RA 138 (#41126) 4/12/82
Computer Science 135 pages

THE RESEARCH QUEUEING PACKAGE VERSION 2 INTRODUCTION AND EXAMPLES

Charles H. Sauer, Edward A. MacNair and James F. Kurose

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract: Queueing networks are important as performance models of systems where performance is principally affected by contention for resources. Such systems include computer systems, communication networks, office systems and manufacturing lines. In order to effectively use queueing networks as performance models, appropriate software is necessary for definition of the networks to be solved, for solution of the networks (by simulation and/or numerical methods) and for examination of the performance measures obtained.

The Research Queueing Package, Version 2 (RESQ) is a system for constructing and solving extended queueing network models. We refer to the class of RESQ networks as "extended" because of characteristics which allow effective representation of system detail. RESQ incorporates a high level language to concisely describe the structure of the model and to specify constraints on the solution. A main feature of the language is the capability to describe models in a hierarchical fashion, allowing an analyst to define submodels to be used analogously to use of macros in programming languages. RESQ also provides a variety of methods for estimating accuracy of simulation results and determining simulation run lengths.

Acknowledgement: We are grateful to P. Heidelberger, E. Jaffe, P. Rosenfeld, M. Reiser, S. Salza, S. Tucci and P.D. Welch for their contributions to RESQ.

This document supplements the CMS and TSO RESQ Users Guides, the primary documents for RESQ usage. Corrections, comments, criticisms and suggestions for improvement of these documents and/or RESQ will be welcomed.

PREFACE

Queueing networks are useful as performance models of systems where performance is principally affected by contention for resources. Such systems include computer systems, communication networks, office systems and manufacturing lines. The Research Queueing Package, Version 2 (hereafter referred to as RESQ) is a system for constructing queueing network models and solving queueing network models. Simulation methods, including state of the art statistical analysis, are provided for the full class of queueing networks allowed in the RESQ language. Numerical methods are provided for a subset of the queueing networks allowed by the RESQ language.

This document introduces usage of RESQ and gives examples of simple models of computer and communication systems constructed and solved using RESQ. The RESQ user should also be familiar with either

> C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package Version 2: CMS Users Guide," IBM Research Report RA-139, Yorktown Heights, New York (April 1982).

or

> C.H. Sauer, E.A. MacNair and J.F. Kurose, "The Research Queueing Package Version 2: TSO Users Guide," IBM Research Report RA-140, Yorktown Heights, New York (April 1982).

whichever is appropriate to the operating system being used. These guides also include a few examples which are more complex than those presented in this document.

This document has the following sections:

"Section 1: Introduction" introduces some of the features and capabilities of RESQ.

"Section 2: Computer System Model - Numerical Solution" illustrates interactive usage of the two basic RESQ commands, SETUP and EVAL, using numerical solution of a model discussed in Section 1.

"Section 3: Dialogue Files - Model Parameters" illustrates the batch mode of the SETUP command and parameters defined with the EVAL command.

"Section 4: Simultaneous Resource Possession - Simulation" discusses simulation of the second example of Section 1.

"Section 5: Confidence Interval Methods" discusses the three methods available in RESQ for statistical analysis of simulation results and automated control of run length.

"Section 6: Sources and Sinks" discusses the RESQ elements for arrival of jobs in the network and departure of jobs from the network.

"Section 7: Chains" discusses the RESQ approach to representing groups of heterogeneous jobs.

"Section 8: Job, Chain and Global Variables" describes variables available during simulation for purposes analogous to variables in the programming language sense.

"Section 9: Routing" discusses the definition of routing between network elements.

"Section 10: Passive Queues" describes in more detail the RESQ elements for explicitly acquiring and freeing units of a resource.

"Section 11: Split, Fission and Fusion Nodes" discusses the RESQ elements used by jobs to generate other jobs and to synchronize activities with these jobs.

"Section 12: Queue Types" discusses a macro facility for queue definition.

"Section 13: Submodels" discusses a macro facility for subnetwork definition.

"Section 14: PL/I Embedding" discusses access to RESQ from PL/I procedures for plotting graphs and constructing hierarchical solutions.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Models are used to estimate the performance of systems when measurement of system performance is impossible (e.g., because the system is not yet operational) or impractical (e.g., because of the human and machine resources required). Queueing networks have become important as performance models of a variety of systems where system performance is usually significantly affected by contention for resources. Queueing network models can be used from the early design stages of a system on throughout the life of the system to estimate system performance.

We will not attempt a general discussion of queueing networks here, but will try to make our discussion self-contained. The reader seeking additional background may wish to refer to special issues of *Computing Surveys* (September 1978) and *Computer* (April 1980), to Sauer and Chandy [SAUE81a] and to other books listed in the Bibliography. Our examples will be of queueing network models of computer systems and communication systems. However, queueing models have been used for decades in examining a wide variety of other systems. Much of our discussion applies directly to performance issues in office equipment, manufacturing lines and other systems. Our emphasis will also be on performance, but the modeling techniques we present also apply to analysis of other issues such as reliability and correctness (e.g., deadlock analysis).

The basic problems in using queueing network models are to (1) determine the resources and their characteristics which will most affect performance, (2) formulate a model representing these resources and characteristics and (3) determine (algebraically, numerically or by simulation) the values for performance measures (e.g, mean response time) in the model. The first of these problems, though often difficult, is highly system specific. We will not address this problem directly. The Research Queueing Package (RESQ) is a software *tool* for building queueing network models. We emphasize "tool" because RESQ is not a model in itself. As a tool, it can be of great value in dealing with the second and third basic problems just cited. This document introduces some of the features of RESQ and their usage. For a thorough discussion of RESQ, see either Users Guide cited in the preface.

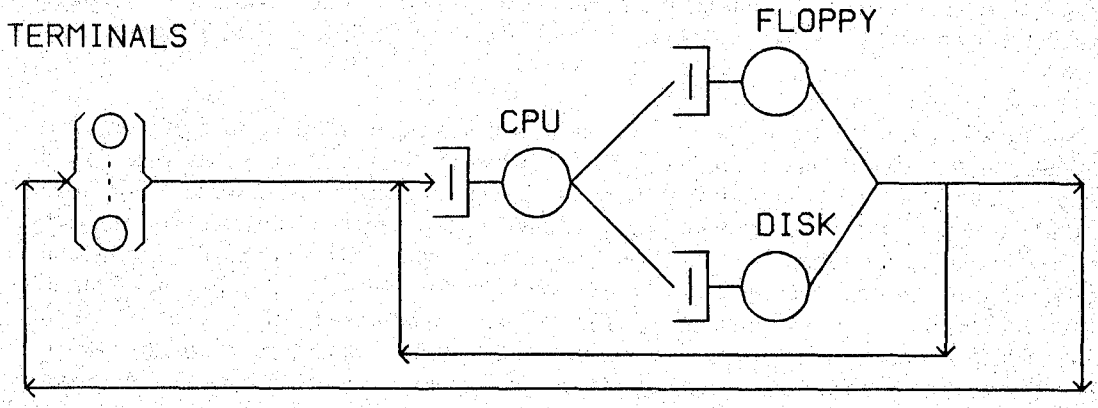


Figure 1.1 - Queueing Network Model

Figure 1.1 illustrates a very simple queueing network model of an interactive computer system. (This network is a simplification of networks used as computer system models since the mid sixties.) The model considers contention for three resources of the system, the CPU, a floppy disk and a hard disk. Users of the system are represented by *jobs* in the queueing network. A user spends part of his or her time thinking at the terminal and keying in commands. This part of the user's time is represented by service times of a job (representing the user) at the terminals "queue." The model assumes there are as many terminals as users, so

there is no waiting for a terminal; we will still refer to the model representation of the terminals as a queue. After keying in a command, the user waits for a response. The job representing the user alternates between computation and I/O activities until the command processing is finished and the user receives the response. The user then begins another thinking/keying time.

We have made this model simple because it is our first example, but we have also made it simple so that a numerical solution of the model will be feasible. For exact numerical solution to be feasible, we must make a number of assumptions. One of these assumptions is that command processing does not require more than one resource at a time. This is likely an unreasonable assumption since command processing will require memory as well as the resources pictured; the assumption is reasonable only if there is negligible contention for memory. Similarly, I/O activity in most architectures will require resources not mentioned, e.g., channels and controllers; the assumption that a single resource is required is only reasonable if there is negligible contention for these other resources. A second assumption is that scheduling is limited to a fairly restricted set of algorithms. In particular, priority scheduling is excluded. Other restrictive assumptions will be considered as we discuss and expand upon this model below.

Without RESQ one would likely have two choices with regard to this model and these assumptions: (1) Accept the model and its results without knowing how much impact the assumptions have on the results. (2) Reject the model and build a detailed simulation model in a conventional discrete event simulation language. This second choice would entail new problems, most notably (a) expense of building and running the model and (b) doubt about the accuracy of the results (due to the statistical variability of simulation). In the past there has been very little middle ground between these choices. advantage of the best features of numerical and simulation solution.

Two of the principal objectives of RESQ have been (1) to bridge this gap between numerical and simulation methods and (2) to encourage analysts to use a solution method appropriate to the case at hand. RESQ has succeeded at these objectives partly because of the solution methods it provides and partly because of its characterizations of queueing networks. RESQ is effective because of its solution methods, because of its characterizations of queueing networks, and because of its user interfaces, which have been engineered to maximize user productivity.

RESQ provides the "state of the art" in numerical solution methods, so that restrictive assumptions can be avoided where possible. RESQ provides simulation solutions with special features not found in most simulation languages. Most important of these are statistical output analysis techniques which provide error estimates (in the form of confidence intervals) for simulation results and stopping rules for determining when the simulation should end. (Statistical output analysis techniques are discussed in Chapter 7 of Sauer and Chandy [SAUE81a], Chapter 4 of Kobayashi [KOBA78] and Chapter 6 of Lavenberg *et al* [LAVE82].) The presence of multiple solution methods in one tool makes it possible to use the method most appropriate to a given model and to test the impact of model assumptions such as the ones discussed above. Presence of multiple solution methods also makes feasible the use of several methods in a hybrid solution of one model.

We refer to the networks of RESQ as "extended" because of characteristics absent from most queueing models. Perhaps the most important of the extensions is the "passive" queue, which allows convenient representation of simultaneous resource possession as in the discussion above. Traditional queues are "active" queues in RESQ terminology. A job's activity is typically focused on the resources of active queues. A job typically has no interaction with other model elements while at an active queue. A job typically acquires units of a passive

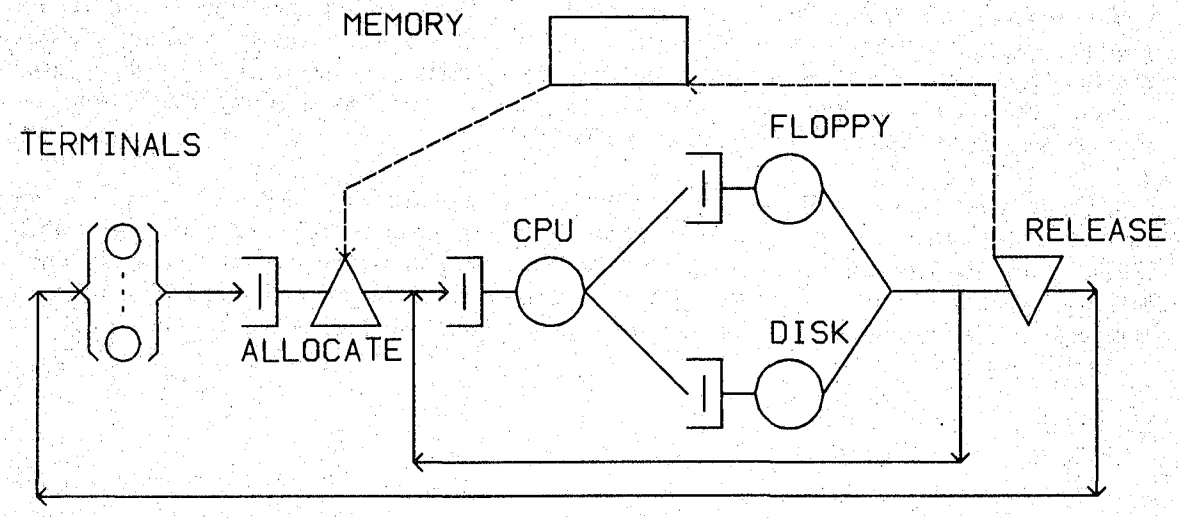


Figure 1.2 - Network with Passive Queue

queue resource and holds on to them while visiting other queues (including other passive queues) and model elements. The job explicitly releases the units of resource when it no longer needs them. Figure 1.2 shows the addition of a passive queue representing memory to the network of Figure 1.1. Inclusion of the passive queue allows us to avoid making the assumption that command processing does not require more than one resource at a time. This assumption is made with the model of Figure 1.1 to make numerical solution feasible; avoidance of the assumption precludes exact numerical solution. Models with passive queues are solved either by simulation or by approximate numerical methods. (Exact numerical solution for networks with passive queues is possible, in principle, but usually not practical.)

Note that in the figure the passive queue resource is held by the job during I/O activity as well. Additional passive queues could be added to the model to represent the channel and/or controller contention mentioned above. As well as representing simultaneous resource possession, passive queues are particularly useful for representing complex mechanisms in a simple manner. For example, contention for a channel may cause I/O devices to experience extra revolutions prior to transfer. Communication network protocols and algorithms are additional examples of mechanisms conveniently represented by passive queues. A third use of passive queues is in measuring response times in subnetworks. The "queueing time" (response time) for a passive queue is defined as the time between a job's request for units of the passive queue resource and that job's freeing of the units of resource. Thus in Figure 1.2 the queueing time for the passive queue corresponds to the response time seen by the terminal users.

The RESQ user interfaces are based on interactive dialogues which serve to educate new users, yet are designed to accommodate sophisticated users and large models. The dialogues provide optional tutorials to clarify prompts. The translator automatically provides for immediate correction of syntactic errors. If a RESQ user discovers a semantic error in prior portions of the dialogue, he or she may temporarily suspend the dialogue, correct the error and then resume the dialogue at the point of suspension. A transcript (a "dialogue file") of a model definition dialogue is kept for the user. The user may edit this transcript and then have it translated again, with or without additional interactive dialogue. In addition to the model definition dialogue and translator, there is a model evaluation dialogue associated with the solution components. This dialogue allows the user to selectively obtain performance measures. Models may be defined with parameters so that solutions of several related models may be obtained in a single evaluation, without retranslation of the model. It is also possible to

embed model evaluation in a PL/I program. The PL/I embedding mechanism is useful for producing graphs or tables of model results for different parameter values. An analyst may use the PL/I embedding mechanism to provide preprocessing and postprocessing for a given model. With such an approach the model may be conveniently used by others who are interested in the modeled system, but not in RESQ.

There are versions of RESQ for both MVS/TSO and for VM/CMS. Most of what we say applies to either version. However, where there are differences, we assume that CMS is being used. Our examples are presented as if a typewriter-type terminal is being used. However, RESQ is insensitive to the type of terminal used and is typically used with a display terminal. The assumption of a typewriter-type terminal simplifies the formatting of the examples.

2. COMPUTER SYSTEM MODEL - NUMERICAL SOLUTION

In this section we will consider numerical solution of the example of Figure 1.1. In doing so we will have to make further assumptions in addition to the ones already discussed.

Let us consider the floppy disk queue. We assume that the queueing discipline for the disk is First-Come-First-Served (FCFS). (We will usually refer to scheduling algorithms as queueing disciplines.) Further, we assume that a job's service time at the disk has an exponential distribution independent of the current state of the disk. Actually, a job's service time will be the sum of several times, including seek time, latency and transfer (and possibly others). The seek and latency times will be dependent on the current position of the arm and the rotational position of the platters. The following is a possible RESQ description of the floppy disk queue:

```
QUEUE:floppyq
   TYPE:fcfs
   CLASS LIST:floppy
      SERVICE TIMES:floppytime
   CLASS LIST:
```

This is a fragment of a RESQ interactive dialogue; we will show the entire dialogue shortly. In this definition we use upper case for the RESQ prompts and lower case for replies to those prompts. The prompts are terminated by a colon (":").

The first prompt is asking for the name of the queue. We use the name "floppyq" rather than "floppy" because we want to save the name "floppy" for another purpose and because "floppyq" is easily pronounced. The name we use may be any legal RESQ identifier (see Appendix 2 of the Users Guide.). ("Floppydiskq" would not be a legal identifier because it has more than ten characters.)

The second prompt is asking for the type of the queue. The type specified may be a general type, i.e., "active" or "passive," or a specialized type, e.g., "fcfs" as in the example. A general type allows specification of all queue characteristics, while a specialized type assumes certain default specifications and thus allows an abbreviated dialogue. The specialized type fcfs results in a single server queue with the FCFS queueing discipline. Further, the server has a fixed service rate of one (1). (If we want the server to have a different fixed rate, we can divide the mean service time by that rate. If we want to explicitly define server rates, we must use the general active dialogue. The general active dialogue is described in Section 4 of the Users Guide.) We will defer until later discussion of some of the other characteristics assumed by the specialized fcfs type. Generally, the assumptions result in a simpler specification than might otherwise be made.

The third prompt is asking for a list of (job) "classes" at the queue. In general, an active queue may have many classes. The classes of a queue serve as "nodes" in the routing description of the network; having multiple classes at a queue allows specification of different routing paths for different jobs leaving the queue. Different classes at a queue may also have different service requirements, priorities and other characteristics we will describe later. In this case there is only one class, which we give the name "floppy." (A class name may be any legal identifier.) We use "floppy" for the class (rather than the queue) because we will use the class name in our routing definition and because of the pronouncability of "floppyq."

The fourth prompt is for the service time distribution of jobs at the queue. The service time is the amount of time needed during one visit to the queue. The identifier "floppytime" is assumed to have been previously defined. Assuming floppytime has been defined to have a

scalar numeric value (another possibility will be discussed later), it is taken to be the mean of an exponential distribution.

The fifth prompt is for more class names. The null reply terminates the definition of floppyq. If the prompt had not been null, there would have been another prompt for service times and yet another class list prompt.

The definition of the hard disk queue can be essentially the same, e.g.,

```
QUEUE:diskq
   TYPE:fcfs
   CLASS LIST:disk
      SERVICE TIMES:disktime
   CLASS LIST:
```

The definition of the cpu queue is similar, but uses the "ps" specialized type:

```
QUEUE:cpuq
   TYPE:ps
   CLASS LIST:cpu
      SERVICE TIMES:cputime
   CLASS LIST:
```

The ps specialized type uses the Processor-Sharing (PS) queueing discipline; otherwise it is the same as the fcfs specialized type. The PS discipline is defined as the limiting case of a Round-Robin discipline with no overhead as the quantum (time slice) goes to zero. With PS and n jobs in the queue, each job gets $1/n^{th}$ of the server, i.e., the server is shared equally among all of the jobs in the queue.

The final queue definition is essentially the same as the other definitions except that we use the "is" special type, which gives a queue with an Infinite-Server (IS) discipline, i.e., the queue always has a server for each job in the queue.

```
QUEUE:terminalsq
   TYPE:is
   CLASS LIST:terminals
      SERVICE TIMES:thinktime
   CLASS LIST:
```

(We use the identifier "thinktime" because "terminalstime" would be too long. Note that "thinktime" should include the keying time and any other times associated with the terminals.) This concludes the queue definitions for this model.

The other principal part of the model definition is that of the routing. The routing is defined in terms of the transitions between nodes; in this model the only nodes are the classes. In general, the nodes of a model may be partitioned into "chains" such that a job at a node in one chain can never get to a node in another chain. In this model there is only one chain. The following is a possible routing definition for this model:

```
   CHAIN:interactive
**ERROR** IDENTIFIER BEGINNING "INTERACTIV" TRUNCATED TO 10 CHARACTERS
      TYPE:closed
      POPULATION:users
      :terminals->cpu
```

```
      :cpu->floppy disk;.1 .9
      :floppy->terminals cpu;1/cpiocycles 1-1/cpiocycles
      :disk->terminals cpu;1/cpiocycles 1-1/cpiocycles
      :
   CHAIN:
```

The first prompt is for the name of a chain. We have intentionally used an identifier longer than ten characters. Note that RESQ gives a warning message informing the user of the truncated identifier it will actually store in its symbol table.

The second prompt is for the type of chain, "open" or "closed." Open chains have "sources" of jobs and "sinks" for jobs. Closed chains do not have sources and sinks. Usually the number of jobs in a closed chain is fixed, though we will see exceptions. The chains of both Figures 1.1 and 1.2 are closed.

The third prompt is for the number of jobs in the closed chain, i.e., its "population." We assume that the identifier "users" has been previously defined to have a scalar numeric value.

The fourth and subsequent prompts, consisting of only colons (":") are for "routing transitions," i.e., descriptions of where a job can go when it leaves a node and how it decides where to go. The first routing transition means that jobs leaving node (class) terminals always go to node cpu. The second routing transition means that jobs leaving node cpu go to node floppy with probability .1 and to node disk with probability .9. The third routing transition means that jobs leaving node floppy go to node terminals with probability 1/cpiocycles and to node cpu otherwise. We assume cpiocycles has been defined to have a scalar numeric value. The fourth transition has the same effect for the disk. Together, the third and fourth transitions mean that the number of CPU-I/O cycles a job experiences will have a geometric distribution (starting at one) with mean cpiocycles. The four transitions have completely described the routing. A null reply to the next colon prompt terminates the chain description. A null reply to the next CHAIN: prompt terminates the routing description.

Having shown the description of the queues and the routing, we now show how these fit into a complete model description. The model definition dialogue is invoked by the command SETUP. The following shows a possible use of SETUP for this model:

```
setup
MODEL:csm

RESQ2 Translator V2.04 (03/02/82)  Time: 21:57:48  Date: 03/10/81

MODEL IS CSM
   METHOD:how
SPECIFY SOLUTION METHOD.
SOLUTION METHODS ARE NUMERICAL AND SIMULATION.
   METHOD:numerical
   NUMERIC PARAMETERS:
   NUMERIC IDENTIFIERS:floppytime disktime cputime thinktime users
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
      THINKTIME:5
      USERS:15
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
```

```
   NUMERIC IDENTIFIERS:
   QUEUE TYPE:
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:floppy
         SERVICE TIMES:floppytime
      CLASS LIST:
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
      CLASS LIST:
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:cpu
         SERVICE TIMES:cputime
      CLASS LIST:
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
      CLASS LIST:
   QUEUE:
   SUBMODEL:
   CHAIN:interactive
**ERROR** IDENTIFIER BEGINNING "INTERACTIV" TRUNCATED TO 10 CHARACTERS
      TYPE:closed
      POPULATION:users
      :terminals->cpu
      :cpu->floppy disk;.1 .9
      :floppy->terminals cpu;1/cpiocycles 1-1/cpiocycles
      :disk->terminals cpu;1/cpiocycles 1-1/cpiocycles
      :
   CHAIN:
END

 NO FATAL ERRORS DETECTED DURING COMPILATION.
R; T=0.73/1.53  22:11:31
```

SETUP will accept a single argument, the model name. If no argument is given to SETUP, it prompts the user for a model name.

Once the model name is established, SETUP prompts for the solution method. As with all RESQ prompts, a reply of "how" causes RESQ to produce a brief explanation of possible replies.

Next is a prompt for the names of numeric parameters, whose values would be supplied when the model is solved. For the moment we assume no parameters, but will return to this feature in the next section.

In the queue and chain definitions we assumed that certain identifiers had been previously defined with numeric values. The next prompt gives an opportunity for definition of such identifiers which have not been declared as parameters. It expects a list of identifiers. After that prompt come prompts for the values of the identifiers and another prompt for more

identifiers. We give another identifier (cpiocycles) and are prompted for its value. We are then given one more prompt for numeric identifiers. A null reply terminates prompting for numeric identifiers.

In this example the values given for times are in units of seconds. This is only implicitly defined, however. As far as RESQ is concerned, the meaning of the time unit is unimportant; the numerical values produced by RESQ would be the same whether we intended the time units to be microseconds, seconds or hours. It is up to the user to decide upon a time unit and be consistent in using it; e.g., the user may choose the time units to be seconds. It is then up to the user to provide all input in units of seconds and to interpret all times in the RESQ output in units of seconds.

Next we are prompted for the name of a user defined queue type. (As discussed in Section 12, we may define our own queue types which may be used in a manner similar to the usage of the predefined queue types fcfs, ps and is.) A null reply indicates we are not defining any queue types. (It is always safe to give a null reply to a prompt. Usually SETUP will accept a null reply as indicating a default value, e.g., no queue type definitions. Occasionally SETUP will insist on some other reply; in those cases the user can use "how" to find out what is expected.)

The queue definitions are the same as the fragments we have already shown. We are then prompted for definition of a submodel; a null reply indicates we are not defining any submodels. The remainder of the model definition is the chain definition already discussed.

Model solutions are obtained with the EVAL command. EVAL may be issued without arguments, in which case it prompts for a model name. If arguments are given to EVAL, the first one is used as a model name. We defer discussion of the interpretation of other arguments. Following is an EVAL dialogue for model csm:

```
eval
RESQ2 EXPANSION AND SOLUTION PROGRAM.
MODEL:csm
RESQ2 VERSION DATE: MARCH 9, 1981 -  TIME:  22:08:11  DATE: 03/10/82
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:all


ELEMENT         UTILIZATION
FLOPPYQ         0.37943
DISKQ           0.29492
CPUQ            0.86234
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
FLOPPYQ         1.72469
DISKQ           15.52218
CPUQ            17.24686
TERMINALSQ      2.15586


ELEMENT         MEAN QUEUE LENGTH
FLOPPYQ         0.58712
```

```
DISKQ          0.40804
CPUQ           3.22555
TERMINALSQ     10.77929


ELEMENT        MEAN QUEUEING TIME
FLOPPYQ        0.34042
DISKQ          0.02629
CPUQ           0.18702
TERMINALSQ     5.00000

WHAT:how
CODES ARE: XXXX XXXX(ELEMENT LIST)
           XXXXCI XXXXCI(ELEMENT LIST)
           XXXXBO XXXXBO(ELEMENT LIST)
WHERE XXXX IS ONE OF THE FOLLOWING:
  UT - UTILIZATION (OF SERVER OR TOKEN)
  TP - THROUGHPUT (DEPARTURES)
  QL - MEAN QUEUE LENGTH
SDQL - STANDARD DEVIATION OF QUEUE LENGTH
 QLD - QUEUE LENGTH DISTRIBUTION
  QT - MEAN QUEUEING TIME
SDQT - STANDARD DEVIATION OF QUEUEING TIME
 QTD - (CUMULATIVE) QUEUEING TIME DISTRIBUTION
  TU - MEAN TOKENS IN USE
 TUD - DISTRIBUTION OF TOKENS IN USE
  TT - MEAN TOTAL TOKENS IN POOL
 TTD - DISTRIBUTION OF TOKENS IN POOL
MXQL - MAXIMUM QL
MXQT - MAXIMUM QT
  PO - OPEN CHAIN POPULATION
 RTM - OPEN CHAIN RESPONSE TIME
 ALL - ALL OF THE ABOVE
XXXX WITHOUT CI OR BO GIVES POINT ESTIMATES ONLY,
XXXXCI GIVES CONFIDENCE INTERVALS ONLY AND
XXXXBO GIVES BOTH POINT ESTIMATES AND CONFIDENCE INTERVALS.
UNLESS AN ELEMENT LIST IS GIVEN, ONLY QUEUE VALUES ARE PRODUCED.
AN ELEMENT LIST IS A LIST OF QUEUES AND NODES.
THE FOLLOWING CODES ARE NOT INCLUDED IN "ALL":
 SIM - GIVES SIMULATION SUMMARY AGAIN
  ND - NUMBER OF DEPARTURES
  ST - MEAN SERVICE TIMES (ACTIVE QUEUES AND CLASSES ONLY)
 LNG - FINAL LENGTHS
  JV - FINAL JV VALUES FOR JOBS STILL IN NETWORK
  CV - FINAL CV VALUES
  GV - FINAL VALUES OF GLOBAL VARIALBES
ND, ST, LNG AND JV MAY BE GIVEN WITH A LIST OF QUEUES AND NODES.
GV MAY BE GIVEN WITH A LIST OF VARIABLE NAMES.
TRY AGAIN-
WHAT:qt(cpuq,cpu)


ELEMENT        MEAN QUEUEING TIME
CPUQ           0.18702
```

```
 CPU              0.18702

WHAT:qtd


ELEMENT          QUEUEING TIME DISTRIBUTION

WHAT:

EXPANSION FINISHED.
R; T=0.45/0.97   22:13:43
```

After obtaining the model name, EVAL prints a version date and the current date and time. EVAL then prints any error messages or the "NO ERRORS ..." message. After that it prompts the user with "WHAT:" meaning "What results do you want to see?" A reply of "all" causes all results to be printed. Where queues consist of a single node (e.g., all of the queues of this example have exactly one class each), only measures for the queues are produced since the node measures will be the same as the queue measures.

The utilization is per server. In the case of the terminals, since the number of servers is "infinite," the utilization of each server is zero. The queue length at a queue is defined to include jobs in service, and the queueing time is defined to include service time.

Perhaps the most interesting performance measures for this model are estimates of response time, but such estimates are not given directly for this model. (We use "estimate" here to emphasize that we are dealing with a model and usually do not obtain the values for the actual system. For this model RESQ provides exact values for performance measures within the limits of numerical error.) By "response time" we know we mean the time from leaving terminalsq to returning to terminalsq, but RESQ has no way of knowing this. In Section 4 we will see how characteristics of response times can be directly estimated in models solved by simulation.

We can easily estimate mean response time from the values we already know in at least two ways: One way is to sum the mean queueing times at cpuq, floppyq and diskq, weighted by the mean number of visits to each queue per response time. I.e., a response time consists (on the average) of 8 queueing times of .18702 seconds at cpuq, .8 queueing times of .34042 seconds at floppyq and 7.2 queueing times of .02629 seconds at diskq, so the mean response time estimate is 1.958 seconds. An easier way is to apply Little's Rule: mean number of jobs = throughput × mean response time. We know that the mean number of jobs *not* at the terminals is 15 − 10.77929 = 4.22071, so the mean response time is 4.22071/2.15586 = 1.958 seconds.

Exact numerical solution for the response time distribution is not feasible. A commonly used *heuristic* is to assume the response time has an exponential distribution, i.e., in this case to assume the probability distribution function has the form $F(t) = 1 - \exp(-t/1.958)$. With that assumption, we would estimate that the probability the response time is at most 1 second would be $1 - \exp(-1/1.958) = .400$, that the probability the response time is at most 3 seconds would be $1 - \exp(-3/1.958) = .784$ and that the probability the response time is at most 5 seconds would be $1 - \exp(-5/1.958) = .922$.

As in SETUP, a reply of "how" causes a tutorial to be printed. In the tutorial above, note that there are many performance measures which were not produced by the "all" reply. These measures are only available with the simulation solution.

Any of the codes for performance measures may be qualified by a list of queues and nodes. Only measures for those elements will be given.

If a code for an unavailable measure, e.g., queueing time distribution in the example, is given, then only the heading is printed. Prompting for measures is terminated by a null reply.

We have only shown the terminal output. A transcript of the EVAL dialogue is preserved for the user in a file with file name the same as the model name and file type RQ2PRNT. This transcript omits errors and "how" output. Thus CSM RQ2PRNT would be

```
RESQ2 VERSION DATE: MARCH 9, 1981 -  TIME:  22:08:11  DATE: 03/10/82
MODEL:CSM
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:all


ELEMENT         UTILIZATION
FLOPPYQ         0.37943
DISKQ           0.29492
CPUQ            0.86234
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
FLOPPYQ         1.72469
DISKQ           15.52218
CPUQ            17.24686
TERMINALSQ      2.15586


ELEMENT         MEAN QUEUE LENGTH
FLOPPYQ         0.58712
DISKQ           0.40804
CPUQ            3.22555
TERMINALSQ      10.77929


ELEMENT         MEAN QUEUEING TIME
FLOPPYQ         0.34042
DISKQ           0.02629
CPUQ            0.18702
TERMINALSQ      5.00000

WHAT:how
WHAT:qt(cpuq,cpu)


ELEMENT         MEAN QUEUEING TIME
CPUQ            0.18702
 CPU             0.18702

WHAT:qtd
```

ELEMENT QUEUEING TIME DISTRIBUTION

WHAT:

3. DIALOGUE FILES - MODEL PARAMETERS

Suppose we wish to evaluate the csm model for a variety of think times and numbers of users. It would be tedious to change the values of thinktime and users and then issue the SETUP and EVAL commands for each pair of values of interest. This is why we provide for numeric parameters declared in SETUP but not defined until EVAL is issued. In the last section when we invoked SETUP we declined to list any numeric parameters. Instead of listing thinktime and users as numeric identifiers we could have listed them as numeric parameters.

We wish to do so now, but we wish to avoid going through the entire SETUP dialogue again. We can avoid this effort by use of dialogue files. While using the SETUP command as illustrated above, it automatically generated a transcript of the dialogue (prompts and replies) on a file with file name CSM (same as the model name) and file type RQ2INP.

This transcript is verbatim with the following exceptions: (1) The "RESQ Translator ..." and "MODEL IS CSM" messages are omitted. (2) Prompts which were given a reply of "how", the how reply and the how tutorial are omitted. (3) Prompts which were repeated because of erroneous replies and the erroneous replies are omitted. (4) Error messages are omitted. (5) Prompts with null replies are omitted. (6) The "NO ERRORS ..." message is omitted. Thus CSM RQ2INP is as follows:

```
MODEL:CSM
   METHOD:numerical
   NUMERIC IDENTIFIERS:floppytime disktime cputime thinktime users
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
      THINKTIME:5
      USERS:15
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:floppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:cpu
         SERVICE TIMES:cputime
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   CHAIN:interactive
      TYPE:closed
      POPULATION:users
      :terminals->cpu
      :cpu->floppy disk;.1 .9
      :floppy->terminals cpu;1/cpiocycles 1-1/cpiocycles
```

```
      :disk->terminals cpu;1/cpiocycles 1-1/cpiocycles
END
```

The following illustrates editing of the dialogue file with the CMS EDIT command:

```
edit csm rq2inp
EDIT:
case m
locate/METHOD:
   METHOD:numerical
input    NUMERIC PARAMETERS:thinktime users
next
   NUMERIC IDENTIFIERS:floppytime disktime cputime thinktime users
change/thinktime users//
   NUMERIC IDENTIFIERS:floppytime disktime cputime
locate/THINKTIME
      THINKTIME:5
delete 2
locate/interactive
   CHAIN:interactive
change/ve/v/
   CHAIN:interactiv
file
R; T=0.06/0.32 12:49:28
```

In this edit session, we first tell the editor that we want mixed lower and upper case. This is necessary because the dialogue file has preserved the user's lower case input and the editor assumes upper case only as its default. There is one exception to the preservation of lower case input: If a model is defined without a dialogue file, then the model name is translated to upper case. Next we locate the solution method prompt and add a line declaring the numeric parameters. Next we go to the numeric identifier prompt with thinktime and users and erase them from the reply. Then we delete the prompts and replies for values of thinktime and users. Finally, we find the "interactive" name which was too long and remove the final "e."

We can now let the SETUP command translate this dialogue file in a batch mode. No interactive dialogue is necessary if we give the SETUP command the model name as part of the command.

```
SETUP csm
MODEL IS CSM
CONTINUING WITH MODEL DEFINITION...
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.43/0.96 12:52:07
```

Several questions may arise in the reader's mind: (1) Why did SETUP not begin an interactive dialogue instead of translating the dialogue file? The answer is that, once given the model name, SETUP will always try to use a dialogue file if it can find one. As we will see in the next section, it is possible to switch back and forth between interactive dialogue and dialogue file within a single issuance of the SETUP command. (2) What about the prompts that would have had null replies in an interactive dialogue but are not present in the dialogue file? In this case there are several instances: there would have been another numeric parameter prompt, there would have been additional CLASS LIST: prompts, there would have been another QUEUE: prompt and there would have been another colon prompt for a routing transition. The answer is that any prompts with null replies can be removed from a dialogue file and

SETUP will still produce the same results as if the prompts with null replies had been there. (3) What about error messages and error handling? The answer is that SETUP will write error messages on the terminal and attempt to continue processing the dialogue file. However, SETUP does not write the offending lines to the terminal. SETUP produces a listing file with file name equal to the model name and file type RQ2LIST. This file corresponds to the listing file a compiler would produce and includes error messages after incorrect lines. CSM RQ2LIST is as follows:

```
RESQ2 Translator V2.04 (03/02/82)  Time: 12:50:48  Date: 03/10/81

*   1*   0*  MODEL:CSM
*   2*   0*      METHOD:numerical
*   3*   0*      NUMERIC PARAMETERS:thinktime users
*   4*   0*      NUMERIC IDENTIFIERS:floppytime disktime cputime
*   5*   0*         FLOPPYTIME:.22
*   6*   0*         DISKTIME:.019
*   7*   0*         CPUTIME:.05
*   8*   0*      NUMERIC IDENTIFIERS:cpiocycles
*   9*   0*         CPIOCYCLES:8
*  10*   0*      QUEUE:floppyq
*  11*   0*         TYPE:fcfs
*  12*   0*         CLASS LIST:floppy
*  13*   0*            SERVICE TIMES:floppytime
*  14*   0*      QUEUE:diskq
*  15*   0*         TYPE:fcfs
*  16*   0*         CLASS LIST:disk
*  17*   0*            SERVICE TIMES:disktime
*  18*   0*      QUEUE:cpuq
*  19*   0*         TYPE:ps
*  20*   0*         CLASS LIST:cpu
*  21*   0*            SERVICE TIMES:cputime
*  22*   0*      QUEUE:terminalsq
*  23*   0*         TYPE:is
*  24*   0*         CLASS LIST:terminals
*  25*   0*            SERVICE TIMES:thinktime
*  26*   0*      CHAIN:interactiv
*  27*   0*         TYPE:closed
*  28*   0*         POPULATION:users
*  29*   0*         :terminals->cpu
*  30*   0*         :cpu->floppy disk;.1 .9
*  31*   0*         :floppy->terminals cpu;1/cpiocycles 1-1/cpiocycles
*  32*   0*         :disk->terminals cpu;1/cpiocycles 1-1/cpiocycles
*  33*   0*  END

 NO FATAL ERRORS DETECTED DURING COMPILATION.
```

Now we are ready to use EVAL again:

```
eval2 csm
RESQ2 EXPANSION AND SOLUTION PROGRAM.
RESQ2 VERSION DATE: MARCH 9, 1981 -  TIME:  13:08:11  DATE: 03/10/82
THINKTIME:10
USERS:15
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.
```

```
WHAT:ut

ELEMENT     UTILIZATION
FLOPPYQ     0.23670
DISKQ       0.18398
CPUQ        0.53796
TERMINALSQ  0.00000

WHAT:
THINKTIME:10
USERS:20
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:ut

ELEMENT     UTILIZATION
FLOPPYQ     0.30661
DISKQ       0.23832
CPUQ        0.69685
TERMINALSQ  0.00000

WHAT:
THINKTIME:10
USERS:30
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:ut

ELEMENT     UTILIZATION
FLOPPYQ     0.40896
DISKQ       0.31787
CPUQ        0.92944
TERMINALSQ  0.00000

WHAT:ql(terminalsq)

ELEMENT     MEAN QUEUE LENGTH
TERMINALSQ  23.23608

WHAT:tp(terminalsq)

ELEMENT     THROUGHPUT
TERMINALSQ  2.32361

WHAT:
THINKTIME:
EXPANSION FINISHED.
R; T=0.50/1.20 14:10:13
```

Note that we gave the model name to EVAL as part of the command. EVAL then prompts us for a value for thinktime and a value for users. We first try twice the previous think time and the same number of users. As we would expect, the CPU utilization goes down considerably. When we give a null reply to WHAT: we are prompted for more parameter values. With 20

users the CPU utilization goes up some, and with 30 users the CPU approaches saturation again. When we give a null reply for a parameter value the EVAL command terminates.

Using the Little's Rule approach, the mean response time estimate is (30 − 23.23608)/2.32361 = 2.911 seconds. Using the exponential assumption the probability the response time is at most 4 seconds is .747, the probability the response time is at most 6 seconds is .873 and the probability the response time is at most 8 seconds is .936.

A file with file name the same as the model name and file type RQ2RPLY may be used instead of the terminal to give replies to the prompts from EVAL. The RQ2RPLY file may include comments using the PL/I comment convention, i.e., comment may be any string enclosed by "/*" and "*/" which does not contain "*/". However, comments must be entirely contained on one line, and a line consisting of only a comment is treated as a blank line. For example, the following RQ2RPLY file could be used with model csm to get the same results as in the above EVAL dialogue:

```
/*THINKTIME:*/ 10
/*USERS:*/ 15
/*WHAT:*/ uncl
/*WHAT:*/ ut
/*WHAT:*/
/*THINKTIME:*/ 10
/*USERS:*/ 20
/*WHAT:*/ ut
/*WHAT:*/
/*THINKTIME:*/ 10
/*USERS:*/ 30
/*WHAT:*/ ut
/*WHAT:*/ ql(terminalsq)
/*WHAT:*/ tp(terminalsq)
/*WHAT:*/
/*THINKTIME:*/
```

In this file the fifth, ninth, fifteenth and sixteenth records have the effect of a null reply.

To summarize the files we have for this model, let us use the CMS LISTFILE command:

```
listfile csm
CSM       RQ2INP    A1
CSM       RQ2LIST   A1
CSM       RQ2COMP   A1
CSM       RQ2RPLY   A1
CSM       RQ2PRNT   A1
R; T=0.02/0.05 14:21:30
```

The RQ2INP file is the dialogue file we have been manipulating. The RQ2LIST file is a listing file produced by SETUP; it is primarily useful when errors are encountered in the RQ2INP file or submodels (see Section 13) are used. The RQ2COMP file is the file passed from SETUP to EVAL. The RQ2RPLY file contains the responses to be stacked for EVAL, as we just discussed. The RQ2PRNT file contains a transcript of the EVAL dialogue.

4. SIMULTANEOUS RESOURCE POSSESSION - SIMULATION

One of the principal limitations of the model of the last two sections is that it ignores simultaneous resource possession, i.e., that jobs must have memory in order to use the processor or a device. We can think in terms of a job passively holding memory while it actively uses the processor or a device. In this section we show how we can add a passive queue to model csm as in Figure 1.2. In order to do so, we edit the dialogue file from the last section:

```
edit csm rq2inp
EDIT:
case m
fname csmwm
next
MODEL:CSM
change/CSM/csmwm
MODEL:csmwm
input   /*Computer System Model with Memory*/
next
   METHOD:numerical
change/numerical/simulation
   METHOD:simulation
locate/CHAIN:/
   CHAIN:interactiv
delete *
EOF:
file
R; T=0.13/0.71 15:02:11
```

First we change the file name so that the old model will be preserved, and change the model name within the file. (It is not strictly necessary to change the model name within the file; the file name is always used as the model name by RESQ commands.)

We also insert a comment explaining the model name. SETUP uses the PL/I comment convention. Comments may be included in replies, where they are treated as blanks, or may be inserted on separate lines as above. As with RQ2RPLY, each comment must be confined to a single line. (This is because the end of a line has meaning in the dialogue file language. A comment too long for one line should be broken into several comments on successive lines.) Then we change the solution method to simulation.

We then delete everything after the queue definitions, leaving an incomplete dialogue file:

```
MODEL:csmwm
   /*Computer System Model with Memory*/
   METHOD:simulation
   NUMERIC PARAMETERS:thinktime users
   NUMERIC IDENTIFIERS:floppytime disktime cputime
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE:floppyq
      TYPE:fcfs
```

```
      CLASS LIST:floppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:cpu
         SERVICE TIMES:cputime
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
```

Now we can use SETUP to translate the partial dialogue file. When SETUP reaches the end of the partial dialogue file, it will switch to interactive mode. SETUP will start prompting us to continue the queue definition, since the last line in the file gives the service times for the terminals.

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
      CLASS LIST:
   QUEUE:memory
      TYPE:passive
      TOKENS:edit
EDIT:
case m
locate/PARAMETERS
   NUMERIC PARAMETERS:thinktime users
change/users/users partitions
   NUMERIC PARAMETERS:thinktime users partitions
file
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
      TOKENS:partitions
      DSPL:fcfs
      ALLOCATE NODE LIST:getmemory
         NUMBERS OF TOKENS TO ALLOCATE:1
      ALLOCATE NODE LIST:
      RELEASE NODE LIST:freememory
      RELEASE NODE LIST:
      DESTROY NODE LIST:
      CREATE NODE LIST:
   QUEUE:
   SET NODES:
   FISSION NODES:
   FUSION NODES:
   SUBMODEL:
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->getmemory->cpu->floppy disk;.1 .9
```

```
      :floppy->freememory cpu;1/cpiocycles 1-1/cpiocycles
      :disk->freememory cpu;1/cpiocycles 1-1/cpiocycles
      :freememory->terminals
      :
   CHAIN:
   QUEUES FOR QUEUEING TIME DIST:memory
      VALUES:1 2 3 4 5 6 7 8
   QUEUES FOR QUEUEING TIME DIST:
   QUEUES FOR QUEUE LENGTH DIST:memory
      MAX VALUE:users/2
   QUEUES FOR QUEUE LENGTH DIST:
   NODES FOR QUEUEING TIME DIST:
   NODES FOR QUEUE LENGTH DIST:
   CONFIDENCE INTERVAL METHOD:none
   INITIAL STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      INIT POP:users
   CHAIN:
   RUN LIMITS-
      SIMULATED TIME:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:500
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:10
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=2.22/6.98 15:53:11
```

In the definition of the memory queue we give the type as "passive." There are no predefined special types for passive queues corresponding to fcfs for active queues, but the user can define special types (Section 12).

A passive queue consists of a "pool" of "tokens" and a set of nodes which interact with the pool. (The tokens are analogous to the servers of an active queue.) The prompt "TOKENS:" is asking for the number of tokens in the pool. Let us assume for now that memory in the computer system is organized into fixed homogeneous partitions such that a job needs exactly one partition of memory for processing and/or I/O. Then a token of the passive queue can represent a partition.

Now suppose we realize we want the number of partitions to be a parameter. We have not declared an identifier for this purpose, so we would like to change the dialogue file before we proceed. As illustrated above, *SETUP will allow us to edit the dialogue file by replying "edit" to any prompt.* When SETUP is given the "edit" reply, it places the user in an editor looking at a dialogue file. This dialogue file includes any interactive dialogue since the SETUP command was issued. When the user leaves the editor (e.g., by filing) SETUP reprocesses the dialogue file left by the editor. If the dialogue file is incomplete, then SETUP switches to prompting mode when it reaches the end of the file. (If the file is complete, SETUP exits without further prompting.)

In the example, we used the editor to add "partitions" as a parameter. After leaving the editor with the file command, SETUP retranslates the dialogue file through our reply "passive" to TYPE: and then reissues the TOKENS: prompt. We can now reply "partitions" to that prompt.

In all of our examples, we assume the standard CMS editor. However, other editors may be made available, as discussed in Section 2.2 of the Users Guide.

The next prompt is for the queueing discipline; we use fcfs.

After that we are prompted for a list of allocate nodes. A job goes to an allocate node when it wants to request tokens from the pool. Allocate nodes in passive queues are the counterparts to classes in active queues. A job will wait at an allocate node until it gets the number of tokens it has requested.

The next prompt is for a distribution for the number of tokens a job needs when it comes to the allocate node. The reply of "1" means that a job needs exactly one token. Note that this is different from the active queue case where a scalar value implies an exponential distribution. The rule is that where continuous distributions are expected, a scalar value implies an exponential distribution, but where a discrete distribution is expected, e.g., because the resulting value should be an integer, a scalar value implies a constant distribution.

We are then given the opportunity to list more allocate nodes. After a null reply we are prompted for a list of "release" nodes. A job gives up any tokens it holds (of a specific passive queue) at a release node and is considered to leave the queue when it goes through the release node. We are prompted for more release nodes and give a null reply.

We are then prompted for destroy node and create node lists for this queue; there are no such nodes in this model and we defer discussion of them until a later section.

Then we are prompted for another queue, for a list of set nodes, for a list of fission nodes, for a list of fusion nodes and for a definition of a submodel; null replies indicate there are none of these.

The routing definition is very similar to before but with one new wrinkle: the reply to the first colon is a series of concatenated routing transitions. The concatenation is permissible as long as the final part of the transition does not involve a routing decision (e.g., by probabilities in its "to part," i.e., its right hand side) and the transition does not include certain node types we have not yet discussed in its "from part," i.e., its left hand side.

If we had wished to, we could have avoided completely respecifying the routing chain interactively and revised the previous definition instead. We could have used the CMS COPYFILE command to copy all or part of the previous model definition (RQ2INP) before we edited it. (If we were using an editor such as XEDIT we could have saved the portion of the file we deleted on a new file at the same time we deleted it. In XEDIT this would be done with the PUTD subcommand.) Having this preparation, we could have given the "edit" reply to the CHAIN: prompt. Then, while in the editor, we could have retrieved the old routing definition (using the GETFILE subcommand) from the other file and modified that definition. We would then proceed with the interactive dialogue that follows the routing chain definition after leaving the editor. With the CMS EDIT command we use in our examples, and this simple model, it is easier to just completely respecify the routing. However, with a full screen editor and/or a more complex model, it is more likely to be appropriate to save and modify portions of dialogue from the previous RQ2INP file than to completely respecify them.

After the chain definition we have finished defining the model proper. However, we still must provide some additional information to define the simulation run. This information falls into several categories:

1. Specification of non-standard performance measures to be gathered.

2. Specification of initial state of the system.

3. Specification of confidence interval method, if any, and parameters of the confidence interval method.

4. Specification of stopping criteria.

5. Specification of simulation trace.

In order for the simulation to estimate distributions of measures such as queueing time, it must reserve storage for each point on the distribution. Rather than attempt to guess which points of the distribution should be gathered for each queue and node and reserve a large amount of storage for information that may not be of interest to the user, the simulation requires that the user specify which distributions are to be gathered and what points of the distributions are to be considered. The prompt "QUEUES FOR QUEUEING TIME DIST:" is asking for a list of queues which are to have queueing time distributions gathered. For each queue listed, SETUP will prompt "VALUES:" for a list of points on the distribution for that queue's queueing time. The simulation will produce estimates for the cumulative distribution at those points, e.g., in the example the simulation will produce estimates of the probability the queueing time is less than or equal to 2, less than or equal to 4, etc. The queue length distribution is treated similarly except: (1) The distribution is estimated for each queue length up to some specified maximum (e.g., one half the number of users in the example). (2) The distribution estimated is not cumulative, i.e., estimates of probability of queue length 0, queue length 1, etc. are produced.

With the regenerative method for confidence intervals (Section 4) we must specify a "regeneration" state similar to the initial state. So SETUP asks for the confidence interval method, if any, before asking for the initial state. We defer discussion of confidence interval methods to Section 4.

The initial state definition section defines where jobs are to be placed when the simulation begins. The initial state is described by chains and by nodes within chains. The NODE LIST: prompt is asking for a list of nodes which will have non-zero populations when the simulation begins. The INIT POP: prompt is asking for a list of the corresponding populations; for closed chains the sum of the elements in the list should equal the chain population. In this example we initially place all of the users at the terminals.

The run limits section defines conditions other than CPU time consumed which will terminate the simulation. The default values are intended to be "infinity;" actually the largest representable floating point or fixed point values are used, depending on the particular limit. The simulation will stop when the first limiting value is reached. As we shall see, the limits specified can be increased after examining the results. Simulated time is time in terms of the model execution. Simulated events are defined in Appendix 7 of the Users Guide. In the examples of this section the events correspond to completion of service times, i.e., departures from the active queues. The prompt "QUEUES FOR DEPARTURE COUNTS:" requests a list of queues where departure count limits are to be considered. The DEPARTURES: prompt requests a corresponding list of counts.

The CPU limit is not in the run limits section because the run limits section is replaced by run "guidelines" with the regenerative method (Section 4). The CPU limit is fairly crude for two reasons: (1) The CPU time consumed is only checked occasionally. The frequency of checking is model and processor dependent but is intended to be roughly once a virtual second on a 3033. (All examples in this document were run on a 3033.) (2) The CPU time considered only includes time consumed during the actual simulation and excludes time preparing for the run (e.g., reading files, obtaining parameter values, etc.) and time spent after the run terminates (e.g., calculating and printing results). (If the run is continued after examining results, the CPU time consumed *includes* the time spent calculating and printing results previously, i.e., the CPU time is from the very beginning of simulation until the end of simulation.)

The TRACE: prompt asks whether we wish to trace the actions of the simulation program. If we had said "yes," then we would be prompted for more details (see Section 12 of the Users Guide).

Having defined the model, we can now evaluate it with EVAL. Using the parameters from before, i.e., 10 second think time and 30 users, and four partitions, we get the following:

```
eval csmwm
RESQ2 EXPANSION AND SOLUTION PROGRAM.
RESQ2 VERSION DATE: MARCH 9, 1981 -  TIME:  16:28:11  DATE: 03/10/82
THINKTIME:10
USERS:30
PARTITIONS:4
RUN END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                SIMULATED TIME:      217.73250
                      CPU TIME:           3.77
              NUMBER OF EVENTS:           8754

WHAT:qt(memory)

ELEMENT          MEAN QUEUEING TIME
MEMORY           3.56839

WHAT:
CONTINUE RUN:yes
LIMIT - MEMORY DEPARTURES:how
LARGER VALUE THAN  500
TRY AGAIN-
LIMIT - MEMORY DEPARTURES:1000
LIMIT - CP SECONDS:how
LARGER VALUE THAN  5 OR NULL TO KEEP THAT VALUE
LIMIT - CP SECONDS:
RUN END: MEMORY DEPARTURE LIMIT
RUN END: CPU LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                SIMULATED TIME:      290.88867
                      CPU TIME:           5.17
              NUMBER OF EVENTS:          11507
```

```
WHAT:nd(memory)

ELEMENT          NUMBER OF DEPARTURES
MEMORY           651

WHAT:qt(memory)

ELEMENT          MEAN QUEUEING TIME
MEMORY           3.76333

WHAT:
CONTINUE RUN:yes
LIMIT - MEMORY DEPARTURES:
LIMIT - CP SECONDS:10
RUN END: MEMORY DEPARTURE LIMIT
RUN END: CPU LIMIT
RUN END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      433.38184
                        CPU TIME:             7.31
                NUMBER OF EVENTS:            16860

WHAT:qt(memory)

ELEMENT          MEAN QUEUEING TIME
MEMORY           3.39005

WHAT:all

ELEMENT          UTILIZATION
MEMORY           0.92252
FLOPPYQ          0.40850
DISKQ            0.31400
CPUQ             0.91481
TERMINALSQ       0.00000

ELEMENT          THROUGHPUT
MEMORY           2.30743
FLOPPYQ          1.85287
DISKQ            16.44046
CPUQ             18.29333
TERMINALSQ       2.31666
FREEMEMORY       2.30743

ELEMENT          MEAN QUEUE LENGTH
MEMORY           7.83167
FLOPPYQ          0.62480
DISKQ            0.43723
CPUQ             2.62806
TERMINALSQ       22.16832

ELEMENT          STANDARD DEVIATION OF QUEUE LENGTH
MEMORY           4.16640
```

```
FLOPPYQ         0.89727
DISKQ           0.74855
CPUQ            1.27861
TERMINALSQ      4.13894

ELEMENT         MEAN QUEUEING TIME
MEMORY          3.39005
FLOPPYQ         0.33721
DISKQ           0.02659
CPUQ            0.14362
TERMINALSQ      9.33247

ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.57104
FLOPPYQ         0.30454
DISKQ           0.02610
CPUQ            0.15563
TERMINALSQ      9.49582

ELEMENT         MEAN TOKENS IN USE
MEMORY          3.69009

ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          4.00000

ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0: 0.01980
                  1: 0.02854
                  2: 0.04084
                  3: 0.06342
                  4: 0.06928
                  5: 0.09986
                  6: 0.09825
                  7: 0.08047
                  8: 0.08460
                  9: 0.09986
                 10: 0.08180
                 11: 0.04505
                 12: 0.03536
                 13: 0.03193
                 14: 0.03103
                 15: 0.04036

ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY           1.00E+00: 0.15900
                 2.00E+00: 0.34100
                 3.00E+00: 0.52400
                 4.00E+00: 0.68400
                 5.00E+00: 0.78200
                 6.00E+00: 0.85500
                 7.00E+00: 0.90900
                 8.00E+00: 0.93500

ELEMENT         DISTRIBUTION OF TOKENS IN USE
```

```
ELEMENT          DISTRIBUTION OF TOTAL TOKENS IN POOL

ELEMENT          MAXIMUM QUEUE LENGTH
MEMORY           19
FLOPPYQ          4
DISKQ            4
CPUQ             4
TERMINALSQ       30

ELEMENT          MAXIMUM QUEUEING TIME
MEMORY           17.08136
FLOPPYQ          2.43811
DISKQ            0.29066
CPUQ             1.60335
TERMINALSQ       101.23787

WHAT:
CONTINUE RUN:no
THINKTIME:
EXPANSION FINISHED.
R; T=8.02/9.51 14:10:18
```

The initial simulation run terminated normally because of the departure limit for the memory queue. (As the model run was specified, the only other reasons the run would stop would be for the CPU time limit or an error.)

Now we have the response time estimates for the model directly available as the queueing time estimates for the memory queue. (Consistent with the definition of queueing time for active queues, the queueing time for passive queues is defined as the time from arrival at the queue to departure from the queue, e.g., release of tokens.) The estimate of mean response time, 3.57 seconds, is 23% higher than the response time estimate for the numerically solved version of this model without memory contention. *Apparently,* the memory contention is having a noticeable effect on response times. We emphasize "apparently" because we have no idea of how much statistical variability has affected the simulation results.

We may be able to get some idea of the variability by letting the run continue to see if there is much change in the results. We continued the run by specifying a larger departure count for the memory queue. EVAL will prompt for larger limits for values not already at "infinity." Larger limits are required for limits that have been reached and are optional for other limits. We then hit the CPU limit after 651 departures. The estimate of mean response time was considerably higher, 3.76 seconds, after only 151 more departures. Then we increased the CPU limit so that we would get the full 1000 departures. With the longer run we got a much smaller estimate of mean response time, 3.39 seconds. The results we got were the same as if we had specified 1000 departures initially. Continuation of runs will produce the same results as if the final limits had been specified initially except possibly for models which stop because of CPU limits. (Two instances of a given run may take slightly different CPU times because of the effects of multiprogramming.)

We could have continued the run further to see if the estimate would change again, but we would rather use one of the formal approaches described in the next section. As we will see, this short a run for this model produces results with great variability.

Now let us suppose the memory is organized in pages instead of partitions. Further, a job's processing requires 16 page frames with probability .25, 32 page frames with probability

.5 and 48 page frames with probability .25. Thus the mean number of frames required is 32. (In this example we are restricting attention to page frames available to users.) We change the model as follows:

```
edit csmwm rq2inp
EDIT:
case m
locate/partitions
   NUMERIC PARAMETERS:thinktime users partitions
change/partitions/pageframes
   NUMERIC PARAMETERS:thinktime users pageframes
locate/partitions
      TOKENS:partitions
change/partitions/pageframes
      TOKENS:pageframes
locate/NUMBERS OF TOKENS TO ALLOCATE
         NUMBERS OF TOKENS TO ALLOCATE:1
change/1/discrete(16,.25;32,.5;48,.25)
         NUMBERS OF TOKENS TO ALLOCATE:discrete(16,.25;32,.5;48,.25)
locate/DEPARTURES:500
         DEPARTURES:500
change/5/10
         DEPARTURES:1000
locate/SECONDS
   LIMIT - CP SECONDS:5
change/5/10
   LIMIT - CP SECONDS:10
file
R; T=0.09/0.34 14:33:41
```

First we changed the name of the parameter specifying the number of tokens. Then we changed the distribution for the number of tokens required from constant at 1 to the above described distribution. The RESQ "discrete" accepts any number of pairs of values and probabilities with the pairs separated by semicolons (";"). In this case there are three pairs. (The commas between values and probabilities are optional. Blanks could be used. Blanks could also appear before and/or after the semicolons.)

Now we use SETUP again:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.70/1.38 14:34:21
```

and EVAL, using the the following RQ2RPLY file:

```
/*Thinktime:*/ 10
/*Users:*/ 30
/*Pageframes:*/ 128
all

/*Continue run:*/ no
/*Thinktime:*/
```

which produces the following RQ2PRNT file:

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 11:47:41  DATE: 03/17/82
MODEL:CSMWM
THINKTIME:/*Thinktime:*/ 10
USERS:/*Users:*/ 30
PAGEFRAMES:/*Pageframes:*/ 128
RUN END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       444.63232
                        CPU TIME:            7.64
                 NUMBER OF EVENTS:           16874

WHAT:all


ELEMENT         UTILIZATION
MEMORY          0.82642
FLOPPYQ         0.41829
DISKQ           0.31052
CPUQ            0.87094
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
MEMORY          2.24905
FLOPPYQ         1.78574
DISKQ           16.05820
CPUQ            17.84845
TERMINALSQ      2.25805
FREEMEMORY      2.24905


ELEMENT         MEAN QUEUE LENGTH
MEMORY          7.02301
FLOPPYQ         0.62368
DISKQ           0.42275
CPUQ            2.30178
TERMINALSQ      22.97699


ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
MEMORY          3.77416
FLOPPYQ         0.87659
DISKQ           0.72183
CPUQ            1.38069
TERMINALSQ      3.77415


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.11431
FLOPPYQ         0.34903
DISKQ           0.02632
```

```
CPUQ            0.12896
TERMINALSQ      9.81125


ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.19806
FLOPPYQ         0.31033
DISKQ           0.02599
CPUQ            0.14421
TERMINALSQ      9.64321


ELEMENT         MEAN TOKENS IN USE
MEMORY          105.78178


ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          128.00002


ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.02748
                  1:0.05196
                  2:0.05272
                  3:0.07851
                  4:0.07436
                  5:0.08615
                  6:0.09493
                  7:0.07985
                  8:0.07594
                  9:0.08690
                 10:0.07881
                 11:0.07982
                 12:0.06958
                 13:0.02964
                 14:0.01516
                 15:0.01056


ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.18300
                2.00E+00:0.34600
                3.00E+00:0.53900
                4.00E+00:0.69700
                5.00E+00:0.83200
                6.00E+00:0.90100
                7.00E+00:0.95200
                8.00E+00:0.97100


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL
```

```
ELEMENT          MAXIMUM QUEUE LENGTH
MEMORY           18
FLOPPYQ          4
DISKQ            5
CPUQ             6
TERMINALSQ       30


ELEMENT          MAXIMUM QUEUEING TIME
MEMORY           13.46424
FLOPPYQ          1.78271
DISKQ            0.29332
CPUQ             1.36720
TERMINALSQ       68.87238

WHAT:
CONTINUE RUN:/*Continue run:*/ no

THINKTIME:/*Thinktime:*/
```

(For the rest of this document, we will usually show RQ2PRNT files rather than actual terminal output.)

With 128 page frames specified, the memory contention is the same on the average in the sense that if each job requires the mean number of frames, at most four jobs can be in memory at once. But now we could have a maximum of two jobs in memory if all the jobs need 48 frames, or up to 8 jobs needing 16 frames. The mean total number of tokens should be exactly 128; the discrepancy is due to numerical error.

The results from the two runs are noticably different, but we do not really know if the difference is due to changes in the model or to statistical variability. One way to consider the statistical variability of simulation is to estimate confidence intervals. The next section discusses methods for estimating confidence intervals that are provided in RESQ.

5. CONFIDENCE INTERVAL METHODS

We have frequently referred to one of the most troublesome problems with simulation: *We need some indication of the accuracy of simulation estimates because of the statistical variability of simulation estimates.* This statistical variability is due to the use of random number streams to drive the simulation. Assuming numerical errors are small, there is no corresponding problem with numerical solution. For example, when we obtained the mean response time estimate of 2.91 seconds for the last parameters of model csm that was an exact value *for the model.* The difference between that value and the mean response time of the modeled system is due entirely to inaccuracies of the model and of parameter estimation, not to inaccuracy of solution. When we obtained the estimate 3.39 seconds for the mean response time of the initial version of model csmwm, we had no idea of how accurate that estimate was *for the mean response time for the model,* much less the modeled system. Though we usually expect the inaccuracies of our models to be the principal source of error in model estimates, it behooves us to attempt some estimate of the error introduced by statistical variability.

The usual method of estimating variability of simulation results is to produce "confidence interval" estimates: given some point estimate p (e.g., 3.39 seconds for mean response time) and other information we produce a confidence interval estimate $(p - \delta, p + \delta)$ and estimate the "true" value (for the model) is contained within the interval with some chosen probability, say .9. This probability, expressed in percent, e.g., 90%, is known as the "confidence level." The quantity δ depends on the confidence level; the higher the confidence level is, the larger δ is. We will use the term "confidence interval" to avoid the mouthful "confidence interval estimate" but it should be remembered that the confidence intervals are only estimates. Note that the true value may lie outside of the confidence interval, but this happens only with a small probability (e.g., $1 - .9 = .1$). If a simulation is not run long enough, or if the performance measure considered is highly variable, then δ may be greater than p and $p - \delta$ may be negative even though the performance measure must be non-negative. Similarly, for performance measures known to be no greater than 1, e.g., utilizations, p and δ may be such that $p + \delta > 1$.

RESQ provides three methods for confidence interval estimation. The methods are implemented to be as transparent to the user as is practical, i.e., to minimize user decision making and to minimize required user understanding of the statistical bases of the methods. No one method is best for all applications.

- The method of independent replications is the preferred method for estimation of transient characteristics. Independent replications may be applied to estimation of equilibrium characteristics, but one of the following two methods will usually be preferable for estimating equilibrium characteristics.

- The regenerative method is the preferred method for estimation of equilibrium behavior in models with regenerative characteristics. Many models constructed with RESQ will have regenerative characteristics, but many other models will not.

- The spectral method is the preferred method for estimation of equilibrium behavior in models without regenerative characteristics. The spectral method may also be applied to models with regenerative characteristics. The regenerative method requires more user sophistication than the spectral method in that the user must be able to define "regeneration states." Definition of a model to use the spectral method is no more difficult than definition of a model to be simulated without confidence intervals.

The regenerative method and the spectral method allow automated run length control based on achieving confidence intervals of a prespecified width. All three methods, independent replications, the regenerative method and the spectral method, are discussed from a statistical point of view in Chapter 6 of Lavenberg *et al* [LAVE82]. Other references in the Bibliography discuss the statistical aspects of the regenerative method and the spectral method in more detail.

The following three subsections are intended to be independent of each other and the remaining sections of this document. The reader may skip one or more (possibly all) of these subsections. Examples in subsequent sections will use the confidence interval methods, but the use of the confidence interval methods is a side issue in the examples.

5.1. Independent Replications

A classical method for obtaining confidence intervals is the method of independent replications. With independent replications we repeat the simulation run several times with everything except the random number streams reset to the original initial state for each replication after the first. The random number streams for the second replication begin where the streams for the first replication ended, the streams for the third replication begin where the streams for the second replication ended, etc.

To use independent replications with csmwm we could first edit the dialogue file:

```
edit csmwm rq2inp
EDIT:
locate/CONFIDENCE
   CONFIDENCE INTERVAL METHOD:none
delete *
EOF:
file
R; T=0.06/0.26 17:00:50
```

and then use SETUP again:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   CONFIDENCE INTERVAL METHOD:replications
   INITIAL STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      INIT POP:users
   CHAIN:
   CONFIDENCE LEVEL:90
   NUMBER OF REPLICATIONS:5
   REPLIC LIMITS-
      SIMULATED TIME:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:1000
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:10
```

```
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=1.35/3.56 17:03:00
```

Usually we are interested in equilibrium behavior of the modeled system. In this case we wish to have the replications long so that the effects of our choice of initial state will not be noticeable. (As we will illustrate later in this subsection, it is possible to discard an initial portion of each replication to reduce the effect of the choice of initial state.) *We prefer a few longer replications to many shorter replications.* Usually we choose the number of replications to be between 5 and 10. The only significant exception is when we want the replications short because we want to notice the effects of our choice of initial state, i.e., we are interested in transient behavior rather than equilibrium behavior. In that case it may be quite reasonable to have many (20 or more) replications.

The CP SECONDS limit is the total for all replications. We have intentionally left the limit too small in the above dialogue so that we can demonstrate how the run continuation mechanism applies to replications.

We used SETUP interactively for clarity. We could have appropriately edited the dialogue file instead, e.g.,

```
edit csmwm rq2inp
EDIT:
locate/CONFIDENCE
   CONFIDENCE INTERVAL METHOD:none
change/none/replications
   CONFIDENCE INTERVAL METHOD:replications
locate/INIT POP/
   INIT POP:users
input    CONFIDENCE LEVEL:90
input    NUMBER OF REPLICATIONS:5
next
   RUN LIMITS-
change/RUN/REPLIC/
   REPLIC LIMITS-
file
R; T=0.11/0.46 17:00:50
```

and then used SETUP again. Now using EVAL we get the following RQ2PRNT file.

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 11:20:35  DATE: 03/17/82
MODEL:CSMWM
THINKTIME:/*Thinktime:*/ 10
USERS:/*Users:*/ 30
PAGEFRAMES:/*Pageframes:*/ 128
REPLICATION   1: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.   6298 DISCARDED EVENTS

  SIMULATED TIME PER REPLICATION:       444.63232
                        CPU TIME:           10.28
NUMBER OF EVENTS PER REPLICATION:           16874
          NUMBER OF REPLICATIONS:               1
```

```
WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.11431

WHAT:
CONTINUE RUN:/*Continue run:*/ yes

LIMIT - CP SECONDS:/*Limit - CP seconds:*/ 50

REPLICATION    1: MEMORY DEPARTURE LIMIT
REPLICATION    2: MEMORY DEPARTURE LIMIT
REPLICATION    3: MEMORY DEPARTURE LIMIT
REPLICATION    4: MEMORY DEPARTURE LIMIT
REPLICATION    5: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

  SIMULATED TIME PER REPLICATION:       445.37769
                        CPU TIME:           38.05
NUMBER OF EVENTS PER REPLICATION:           16966
          NUMBER OF REPLICATIONS:               5

WHAT:allbo


ELEMENT          UTILIZATION
MEMORY           0.84662(0.83293,0.86030) 2.7%
FLOPPYQ          0.40334(0.39382,0.41286) 1.9%
DISKQ            0.30808(0.30467,0.31150) 0.7%
CPUQ             0.89286(0.87936,0.90636) 2.7%
TERMINALSQ       0.00000(0.00000,0.00000)


ELEMENT          THROUGHPUT
MEMORY           2.24855(2.15706,2.34004) 8.1%
FLOPPYQ          1.81734(1.78729,1.84739) 3.3%
DISKQ            16.10393(15.94658,16.26129) 2.0%
CPUQ             17.92303(17.74706,18.09900) 2.0%
TERMINALSQ       2.26418(2.17488,2.35349) 7.9%
FREEMEMORY       6.73585


ELEMENT          MEAN QUEUE LENGTH
MEMORY           7.65754(6.92341,8.39167) 19.2%
FLOPPYQ          0.59750(0.57292,0.62207) 8.2%
DISKQ            0.42267(0.41569,0.42966) 3.3%
CPUQ             2.44248(2.36520,2.51977) 6.3%
TERMINALSQ       22.34245(21.60832,23.07658) 6.6%


ELEMENT          STANDARD DEVIATION OF QUEUE LENGTH
MEMORY           3.81189(3.55682,4.06696) 13.4%
FLOPPYQ          0.85975(0.83129,0.88821) 6.6%
```

```
DISKQ          0.72997(0.72030,0.73963) 2.6%
CPUQ           1.37497(1.34781,1.40212) 3.9%
TERMINALSQ     3.81189(3.55682,4.06696) 13.4%


ELEMENT        MEAN QUEUEING TIME
MEMORY         3.39909(2.96338,3.83480) 25.6%
FLOPPYQ        0.32891(0.31073,0.34709) 11.1%
DISKQ          0.02624(0.02593,0.02656) 2.4%
CPUQ           0.13621(0.13203,0.14040) 6.1%
TERMINALSQ     9.63718(9.33058,9.94379) 6.4%


ELEMENT        STANDARD DEVIATION OF QUEUEING TIME
MEMORY         2.30823(2.06275,2.55372) 21.3%
FLOPPYQ        0.31195(0.29720,0.32671) 9.5%
DISKQ          0.02582(0.02520,0.02645) 4.8%
CPUQ           0.14961(0.14468,0.15454) 6.6%
TERMINALSQ     9.48424(9.16274,9.80575) 6.8%


ELEMENT        MEAN TOKENS IN USE
MEMORY         108.36685(106.61545,110.11826) 3.2%


ELEMENT        MEAN TOTAL TOKENS IN POOL
MEMORY         127.99998(127.99998,128.00000) 0.0%


ELEMENT        QUEUE LENGTH DISTRIBUTION
MEMORY           0:0.01574(0.00823,0.02325) 1.5%
                 1:0.03582(0.02560,0.04605) 2.0%
                 2:0.04874(0.04095,0.05652) 1.6%
                 3:0.06408(0.04862,0.07953) 3.1%
                 4:0.07053(0.05178,0.08928) 3.7%
                 5:0.07709(0.06065,0.09354) 3.3%
                 6:0.08961(0.07677,0.10246) 2.6%
                 7:0.08638(0.07978,0.09298) 1.3%
                 8:0.09105(0.07284,0.10926) 3.6%
                 9:0.09054(0.08325,0.09782) 1.5%
                10:0.08554(0.06862,0.10246) 3.4%
                11:0.07549(0.05956,0.09141) 3.2%
                12:0.06065(0.04926,0.07205) 2.3%
                13:0.03866(0.02522,0.05210) 2.7%
                14:0.02907(0.01547,0.04267) 2.7%
                15:0.01951(0.00754,0.03148) 2.4%


ELEMENT        QUEUEING TIME DISTRIBUTION
MEMORY         1.00E+00:0.15220(0.12760,0.17680) 4.9%
               2.00E+00:0.31960(0.27732,0.36188) 8.5%
               3.00E+00:0.49200(0.41864,0.56536) 14.7%
               4.00E+00:0.65360(0.56817,0.73903) 17.1%
               5.00E+00:0.77840(0.70010,0.85670) 15.7%
```

```
                6.00E+00:0.87260(0.82596,0.91924) 9.3%
                7.00E+00:0.92680(0.89735,0.95625) 5.9%
                8.00E+00:0.95460(0.92936,0.97984) 5.0%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          22
FLOPPYQ         5
DISKQ           5
CPUQ            7
TERMINALSQ      30


ELEMENT         MAXIMUM QUEUEING TIME
MEMORY          16.54839
FLOPPYQ         2.71239
DISKQ           0.31800
CPUQ            2.39180
TERMINALSQ      75.52458

WHAT:
THINKTIME:/*Thinktime:*/
```

With the initial CPU limit, the run stopped during the second replication. When a run stops during a replication, the results are based on the completed replications and the partial replication is discarded if the run is not continued. In this case there is only a single replication (with results the same as our previous run without replications). There must be at least two replications for confidence intervals to be produced. The run continuation mechanism allows us to resume the partial replication where it stopped and continue for the remaining replications. When the run is completed, the point estimate for the mean response time, 3.40, is considerably higher than the estimate given by the single run of the last section, 3.12. However, the confidence interval, (2.96, 3.83), is very wide. Thus the confidence interval has told us that the initial estimate was quite variable. The numbers after the confidence intervals are the widths of the intervals. For the measures which can only have values in the [0, 1] interval, i.e., utilization and the distribution measures, the width specified is absolute width in percent, i.e., $200 \times \delta$, where the confidence interval is $(p - \delta, p + \delta)$. For the other measures the width is relative width in percent, i.e., $200 \times \delta / p$. (Where p is zero, no width is given.) The mean response time confidence interval has a relative width of 26%.

At this point we could either increase the replication length or increase the number of replications to try to get a narrower interval. Usually we would strongly prefer increasing the replication length over increasing the number of replications. It is for this reason that we do not provide the option of specifying that the run be continued by increasing the number of replications.

When using independent replications to obtain confidence intervals, or when making a run without obtaining confidence intervals, is often advisable to discard results from the initial transient phase of a replication or run. Results from the remainder of the run are, presumably,

more representative of the equilibrium behavior to be studied if the effects of the initial system state can be masked. For a formal discussion, see Chapter 6 of Lavenberg *et al* [LAVE82]. Immediately before the REPLIC LIMITS section of a dialogue (RQ2INP) file, a line of the form "INITIAL PORTION DISCARDED: <expression>" may be inserted. This is the first instance we have seen of a portion of the dialogue file language which is not part of the interactive dialogue of SETUP. There are many such instances which we will discuss where appropriate. The reference for the syntax of such instances (and the entire dialogue file language) is the grammar in Appendix 4 of the Users Guide. The expression gives the fraction, in percent, of each replication or run that will be discarded. This fraction applies only to the limits in the REPLIC LIMITS and not to the CP SECONDS limit. For each limit of the section, a temporary limit is established by multiplying the given limit by the fraction. Once one of these temporary limits is reached, the variables used to accumulate performance measures are reset, the original limits are put in effect and the replication or run continues.

In the following we indicate that the first 10% of each replication is to be discarded and that the replications are to be twice as long as before.

```
edit csmwm rq2inp
EDIT:
locate/NUMBER OF REPLICATIONS
   NUMBER OF REPLICATIONS;5
input    INITIAL PORTION DISCARDED:10
locate/DEPARTURES:
         DEPARTURES:1000
change/1/2
         DEPARTURES:2000
locate/SECONDS
   LIMIT - CP SECONDS:10
change/10/100
   LIMIT - CP SECONDS:100
file
R; T=0.11/0.46 17:33:50
```

use SETUP again,

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.85/2.06 17:37:00
```

and get the following results

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 17:38:43  DATE: 03/17/82
MODEL:CSMWM
THINKTIME:/*Thinktime:*/ 10
USERS:/*Users:*/ 30
PAGEFRAMES:/*Pageframes:*/ 128
REPLICATION    1: MEMORY DEPARTURE LIMIT
REPLICATION    2: MEMORY DEPARTURE LIMIT
REPLICATION    3: MEMORY DEPARTURE LIMIT
REPLICATION    4: MEMORY DEPARTURE LIMIT
REPLICATION    5: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.  17418 DISCARDED EVENTS
```

```
  SIMULATED TIME PER REPLICATION:       812.77954
                        CPU TIME:           77.43
NUMBER OF EVENTS PER REPLICATION:           30857
          NUMBER OF REPLICATIONS:               5

WHAT:uncl
WHAT:allbo


ELEMENT         UTILIZATION
MEMORY          0.83975(0.83341,0.84609)  1.3%
FLOPPYQ         0.40552(0.39509,0.41594)  2.1%
DISKQ           0.30475(0.30075,0.30874)  0.8%
CPUQ            0.89125(0.88431,0.89818)  1.4%
TERMINALSQ      0.00000(0.00000,0.00000)


ELEMENT         THROUGHPUT
MEMORY          2.21534(2.17291,2.25777)  3.8%
FLOPPYQ         1.81099(1.76803,1.85395)  4.7%
DISKQ           16.06598(15.95485,16.17711)  1.4%
CPUQ            17.87697(17.73483,18.01910)  1.6%
TERMINALSQ      2.21261(2.17076,2.25446)  3.8%
FREEMEMORY      2.21462


ELEMENT         MEAN QUEUE LENGTH
MEMORY          7.77057(7.35626,8.18487)  10.7%
FLOPPYQ         0.60917(0.58465,0.63369)  8.1%
DISKQ           0.41371(0.40920,0.41821)  2.2%
CPUQ            2.42072(2.38423,2.45721)  3.0%
TERMINALSQ      22.22943(21.81512,22.64372)  3.7%


ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
MEMORY          4.14229(3.93683,4.34775)  9.9%
FLOPPYQ         0.87904(0.85098,0.90709)  6.4%
DISKQ           0.71696(0.71349,0.72042)  1.0%
CPUQ            1.37582(1.34426,1.40738)  4.6%
TERMINALSQ      4.14229(3.93683,4.34775)  9.9%


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.50457(3.27995,3.72919)  12.8%
FLOPPYQ         0.33633(0.32714,0.34553)  5.5%
DISKQ           0.02575(0.02544,0.02606)  2.4%
CPUQ            0.13537(0.13332,0.13742)  3.0%
TERMINALSQ      9.92243(9.72741,10.11744)  3.9%


ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.47966(2.35171,2.60760)  10.3%
FLOPPYQ         0.32228(0.30464,0.33993)  11.0%
DISKQ           0.02513(0.02449,0.02577)  5.1%
```

```
CPUQ            0.15013(0.14653,0.15374) 4.8%
TERMINALSQ      10.07628(9.73916,10.41340) 6.7%

ELEMENT         MEAN TOKENS IN USE
MEMORY          107.48819(106.67644,108.29993) 1.5%

ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          127.99998(127.99998,128.00000) 0.0%

ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.01823(0.01281,0.02364) 1.1%
                  1:0.04016(0.03533,0.04499) 1.0%
                  2:0.05491(0.04938,0.06045) 1.1%
                  3:0.06494(0.05547,0.07442) 1.9%
                  4:0.06867(0.06012,0.07721) 1.7%
                  5:0.07573(0.06814,0.08333) 1.5%
                  6:0.08232(0.07172,0.09293) 2.1%
                  7:0.08424(0.07792,0.09056) 1.3%
                  8:0.08469(0.07457,0.09482) 2.0%
                  9:0.08598(0.07671,0.09524) 1.9%
                 10:0.07246(0.06459,0.08034) 1.6%
                 11:0.06966(0.06011,0.07922) 1.9%
                 12:0.05493(0.04589,0.06397) 1.8%
                 13:0.04574(0.03158,0.05989) 2.8%
                 14:0.03538(0.02575,0.04502) 1.9%
                 15:0.02645(0.01779,0.03511) 1.7%

ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.15578(0.13985,0.17170) 3.2%
                2.00E+00:0.31511(0.28990,0.34032) 5.0%
                3.00E+00:0.48067(0.44496,0.51637) 7.1%
                4.00E+00:0.63233(0.59074,0.67392) 8.3%
                5.00E+00:0.75344(0.71622,0.79067) 7.4%
                6.00E+00:0.85033(0.82113,0.87953) 5.8%
                7.00E+00:0.90844(0.88613,0.93075) 4.5%
                8.00E+00:0.94789(0.93460,0.96118) 2.7%

ELEMENT         DISTRIBUTION OF TOKENS IN USE

ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL

ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          22
FLOPPYQ         5
DISKQ           6
CPUQ            7
TERMINALSQ      30
```

```
ELEMENT          MAXIMUM QUEUEING TIME
MEMORY           18.90594
FLOPPYQ          2.65272
DISKQ            0.29766
CPUQ             1.90131
TERMINALSQ       80.32346

WHAT:
THINKTIME:/*Thinktime:*/
```

The 17418 discarded events are from the initial portions of all 5 replications, i.e., the average number discarded per replication is 3484. The simulated time and events per replication do not count the discarded portions. The point estimate for mean response time has increased slightly, from our previous set of replications, from 3.40 to 3.50, and we have a narrower confidence interval, (3.28, 3.73). If we wish to have a narrower interval, then we should increase the replication length again.

5.2. The Regenerative Method

The regenerative method is a second method provided for confidence interval estimates for equilibrium measures. The principal advantages of the regenerative method over replications are that we can make a single (long) simulation run instead of multiple (shorter) runs and that we need not be concerned about the effects of the choice of initial state. However, there are problems with the regenerative method also.

With the regenerative method we must pick a "regeneration state," similar to the initial state. A regeneration state has the properties that (1) The model periodically returns to the regeneration state. The periods between occurrences of the regeneration state are called "cycles." (2) When the model enters the regeneration state, the future behavior of the model depends only on the regeneration state, i.e., it is independent of the behavior that led to entrance to that state. The most convenient examples of regeneration states are found in Markov and semi-Markov processes. In a "nice" (semi-) Markov process, each state is a regeneration state, and except for practical considerations, all of the states are equally useful. A large subset of the queueing networks allowed by RESQ can be described as (semi-) Markov processes, and these processes will usually be "nice" unless a queue of the network is saturated or a deadlock is possible in the network.

The principal practical consideration is that we would like the regeneration state to occur frequently during a simulation of reasonable length. By "frequently" we mean that there be at least some minimum number of cycles (say 20) during the simulation. If we do not have this property then we cannot reasonably use the regenerative method.

We would also like the state to be one easily detected by the simulation. For this reason, RESQ only allows regeneration states which are specified by the number of jobs at each node with the understanding that additional characteristics of the states are specified implicitly. These implicitly specified characteristics are (1) Where arrival and service distributions are specified by the method of exponential stages (see Appendix 3 of the Users Guide) any arrival and service times in progress are in the first stage in the regeneration state. (2) At active queues where different orderings of the jobs in the queue are important (e.g., FCFS queueing discipline) the ordering of jobs of different classes is the same as at the first occurrence of the required numbers of jobs at all nodes. (3) At passive queues the ordering of jobs of different allocate nodes and different numbers of tokens requested is the same as at the first occurrence of the required numbers of jobs at each node. (4) Chain variables of open chains (see

Section 8) have the value one. In addition to these checks, Aplomb issues warning conditions when an apparently correctly defined regeneration state is not actually a regeneration state.

For further discussion of the regenerative method in general, see Crane and Lemoine [CRAN77], Iglehart and Shedler [IGLE80], Chapter 4 of Kobayashi [KOBA78], Chapter 6 of Lavenberg *et al* [LAVE82] and Chapter 7 of Sauer and Chandy [SAUE81a].

With model csmwm, our choice of initial state is also a regeneration state. We can edit as follows,

```
edit csmwm rq2inp
EDIT:
locate/CONFIDENCE/
   CONFIDENCE INTERVAL METHOD:none
delete *
EOF:
file
R; T=0.06/0.21 15:34:37
```

and then use SETUP:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      REGEN POP:users
      INIT POP:users
   CHAIN:
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:no
   RUN GUIDELINES-
      SIMULATED TIME:
      CYCLES:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:500
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:5
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.86/1.94 15:35:51
```

The initial state definition section has been replaced by a regeneration state definition section. Usually we want to initially place the system in the regeneration state. Occasionally this is not easily done and we define the initial state to be a state other than the regeneration state. In this case the portion of the simulation prior to the first occurrence of the regeneration state is discarded. See Section 12 of the Users Guide for further discusssion.

We will temporarily defer discussion of the sequential stopping rule. The run limits other than the CPU limit have been replaced by run guidelines. Rather than terminate the simulation when the first of these guidelines is reached the simulation continues until either the regeneration state is reached again or the CPU limit is reached.

We could then get the following RQ2PRNT file:

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 20:35:25  DATE: 03/16/82
MODEL:CSMWM
THINKTIME:10
USERS:30
PAGEFRAMES:128
RUN END: MEMORY DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       286.88623
                        CPU TIME:            4.99
                 NUMBER OF EVENTS:           11331
                 NUMBER OF CYCLES:              13

WHAT:nd(memory)


ELEMENT          NUMBER OF DEPARTURES
MEMORY           657

WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.69609(3.35480,4.03737) 18.5%

WHAT:
CONTINUE RUN:yes

GUIDELINE - MEMORY DEPARTURES:1000

LIMIT - CP SECONDS:6

RUN END: MEMORY DEPARTURE GUIDELINE
RUN END: CPU LIMIT
NO ERRORS DETECTED DURING SIMULATION.    1377 DISCARDED EVENTS

                  SIMULATED TIME:       335.31738
                        CPU TIME:            6.33
                 NUMBER OF EVENTS:           12850
                 NUMBER OF CYCLES:              27

WHAT:nd(memory)


ELEMENT          NUMBER OF DEPARTURES
MEMORY           760
```

```
WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.43366(3.06772,3.79960) 21.3%

WHAT:
CONTINUE RUN:yes

LIMIT - CP SECONDS:40

RUN END: MEMORY DEPARTURE GUIDELINE
RUN END: CPU LIMIT
RUN END: MEMORY DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       561.08374
                        CPU TIME:            9.52
                 NUMBER OF EVENTS:           21481
                 NUMBER OF CYCLES:              32

WHAT:nd(memory)


ELEMENT          NUMBER OF DEPARTURES
MEMORY           1249

WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.31099(2.88925,3.73273) 25.5%

WHAT:
CONTINUE RUN:yes

GUIDELINE - MEMORY DEPARTURES:2000

RUN END: MEMORY DEPARTURE GUIDELINE
RUN END: CPU LIMIT
RUN END: MEMORY DEPARTURE GUIDELINE
RUN END: MEMORY DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       980.62939
                        CPU TIME:           16.66
                 NUMBER OF EVENTS:           37328
                 NUMBER OF CYCLES:              57

WHAT:nd(memory)


ELEMENT          NUMBER OF DEPARTURES
MEMORY           2208
```

```
WHAT:qtbo(memory)

ELEMENT          MEAN QUEUEING TIME
MEMORY           3.31633(3.03503,3.59764) 17.0%

WHAT:allbo

ELEMENT          UTILIZATION
MEMORY           0.84297(0.81823,0.86770) 4.9%
FLOPPYQ          0.40977(0.38383,0.43572) 5.2%
DISKQ            0.30904(0.30234,0.31574) 1.3%
CPUQ             0.88643(0.86837,0.90449) 3.6%
TERMINALSQ       0.00000(0.00000,0.00000)

ELEMENT          THROUGHPUT
MEMORY           2.25161(2.19484,2.30839) 5.0%
FLOPPYQ          1.79069(1.71812,1.86325) 8.1%
DISKQ            16.11617(15.75942,16.47290) 4.4%
CPUQ             17.90686(17.53708,18.27663) 4.1%
TERMINALSQ       2.25161(2.19484,2.30839) 5.0%
FREEMEMORY       2.25161

ELEMENT          MEAN QUEUE LENGTH
MEMORY           7.46711(6.90325,8.03097) 15.1%
FLOPPYQ          0.60705(0.55009,0.66402) 18.8%
DISKQ            0.41974(0.40675,0.43273) 6.2%
CPUQ             2.42949(2.31021,2.54877) 9.8%
TERMINALSQ       22.53288(21.96902,23.09673) 5.0%

ELEMENT          STANDARD DEVIATION OF QUEUE LENGTH
MEMORY           3.82926
FLOPPYQ          0.87095
DISKQ            0.72073
CPUQ             1.40257
TERMINALSQ       3.82927

ELEMENT          MEAN QUEUEING TIME
MEMORY           3.31633(3.03503,3.59764) 17.0%
FLOPPYQ          0.33901(0.31533,0.36269) 14.0%
DISKQ            0.02604(0.02554,0.02655) 3.9%
CPUQ             0.13567(0.13013,0.14122) 8.2%
TERMINALSQ       10.00743(9.69447,10.32040) 6.3%

ELEMENT          STANDARD DEVIATION OF QUEUEING TIME
MEMORY           2.26665
FLOPPYQ          0.31497
DISKQ            0.02566
```

```
CPUQ            0.15399
TERMINALSQ      9.77573


ELEMENT         MEAN TOKENS IN USE
MEMORY          107.89966(104.73351,111.06580) 5.9%


ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          127.99998


ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.02189(0.01176,0.03201) 2.0%
                  1:0.03761(0.02409,0.05113) 2.7%
                  2:0.04806(0.03487,0.06125) 2.6%
                  3:0.06976(0.05091,0.08862) 3.8%
                  4:0.07628(0.05659,0.09596) 3.9%
                  5:0.07498(0.06261,0.08735) 2.5%
                  6:0.08536(0.07417,0.09656) 2.2%
                  7:0.07594(0.06764,0.08425) 1.7%
                  8:0.09154(0.07741,0.10566) 2.8%
                  9:0.10343(0.08079,0.12606) 4.5%
                 10:0.08607(0.06737,0.10478) 3.7%
                 11:0.07894(0.05895,0.09894) 4.0%
                 12:0.06137(0.04304,0.07971) 3.7%
                 13:0.03408(0.02542,0.04274) 1.7%
                 14:0.02525(0.01918,0.03132) 1.2%
                 15:0.01416(0.00849,0.01983) 1.1%


ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.16486(0.12866,0.20105) 7.2%
                2.00E+00:0.31884(0.26575,0.37193) 10.6%
                3.00E+00:0.49683(0.43543,0.55823) 12.3%
                4.00E+00:0.65761(0.60555,0.70966) 10.4%
                5.00E+00:0.79303(0.75685,0.82920) 7.2%
                6.00E+00:0.88225(0.85880,0.90570) 4.7%
                7.00E+00:0.93705(0.91911,0.95499) 3.6%
                8.00E+00:0.96558(0.95279,0.97837) 2.6%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          21
FLOPPYQ         5
DISKQ           6
CPUQ            7
TERMINALSQ      30
```

```
ELEMENT          MAXIMUM QUEUEING TIME
MEMORY           13.46424
FLOPPYQ          2.23200
DISKQ            0.29332
CPUQ             1.90131
TERMINALSQ       76.31847

WHAT:
CONTINUE RUN:no

THINKTIME:
```

The simulation had to run an additional 157 memory departures to get back to the regeneration state after the initial departure guideline was reached. There were 13 regeneration cycles during that part of the run. (If there had been fewer than two cycles, confidence intervals would not have been estimated and we would not have been allowed to continue the run.) The point estimate for mean response time, 3.70 seconds, was considerably higher than our previous estimates. The confidence interval, (3.35, 4.04), has a relative width of 19%, which is not so wide as to be useless.

However, we must be cautious. *There is a tendency for the regenerative method to underestimate confidence interval widths for short runs.*

So we doubled the departure guideline. Since we did not increase the CPU limit sufficiently, the run stopped in the midst of a regeneration cycle before reaching the new guideline. When a run stops in the midst of a regeneration cycle because of the CPU limit or because of an error, the partial cycle results are ignored and will be discarded if the run is not continued. Lengthening the run by only 103 departures resulted in an additional 14 regeneration cycles, a noticeably lower point estimate, 3.43 seconds, and a wider confidence interval, (3.07, 3.80). With a sufficient CPU limit, the run took an additional 249 departures to get back to the regeneration state after the guideline was reached. There were only 5 additional regeneration cycles during this part of the run, so it appears that the length of a regeneration cycle, measured in number of departures, is quite variable for this model. The point estimate for mean response time, 3.31, is considerably lower, and the confidence interval, (2.89, 3.73), is considerably wider! (It has a relative width of 25%.) When we increased the departure guideline to 2000, the run went 208 departures past the guideline for a total of 57 cycles. The point estimate, 3.32, is essentially unchanged, but the confidence interval, (3.04, 3.60), is narrower, with a relative width of 17%. *At this point the run is more than three times as long (measured in memory departures) as the initial part, but the respective confidence intervals have comparable widths!* This suggests that the initial run was much too short and that we should probably continue the run further.

So we need a longer run, but how much longer? Rather than proceed in the above manner of lengthening the run and periodically examining the results, we can use the sequential stopping rule, which automates essentially this procedure. The sequential stopping rule allows us to specify the simulation run length in terms of desired widths of confidence intervals, subject to the usual limit on CPU time. The simulation runs for a number of regeneration cycles, e.g., enough for 2000 memory departures, and then confidence intervals are obtained. If the intervals do not meet the width criteria, the simulation continues for more cycles, e.g., enough for 2000 more memory departures. Then new estimates are made and a new decision to terminate or continue is reached. This continues until the criteria are satisfied or the CPU limit is reached. The groups of regeneration cycles will be referred to as "sampling periods."

As before, we can edit the dialogue file,

```
edit csmwm rq2inp
EDIT:
locate/SEQUENTIAL
   SEQUENTIAL STOPPING RULE:no
delete *
EOF:
file
R; T=0.06/0.28 15:45:50
```

and use SETUP again:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:memory cpuq
         MEASURES:qt qt
         ALLOWED WIDTHS:10 10
      QUEUES TO BE CHECKED:
      EXTRA SAMPLING PERIODS:
   SAMPLING PERIOD GUIDELINES-
      SIMULATED TIME:
      CYCLES:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:2000
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:300
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.83/2.06 15:48:02
```

We are asked for a list of queues where we are to obtain and check confidence interval widths at the end of a sampling period. Then we are asked what measures are to be considered. The reply here should be a list of codes, one code per queue just listed. The codes are a subset of those allowed for the "WHAT:" prompt in EVAL: "ut", "tp", "ql", "qld", "qt", "qtd", "tu", "tud", "tt" and "ttd". These correspond to the same measures as in EVAL (see the example in Section 2 or Sections 12 and 13 of the Users Guide). If we want several measures to be checked for a given queue, the queue name should be repeated in the "QUEUES TO BE CHECKED:" prompt. For the distribution measures (qld, qtd, tud and ttd), each gathered point of the measure is checked and must satisfy the width criteria. For the measures which can only have values in the [0, 1] interval, i.e., utilization and the distribution measures, the width specified is absolute width in percent, i.e., the criterion is that $200\times\delta$ be less than the specified width, where the confidence interval is $(p - \delta, p + \delta)$. For the other measures the width is relative width in percent, i.e., the criterion is that $200\times\delta/p$ be less than the specified width. (Where p is zero, the criteria is not satisfied.) In this and most of our examples we use mean queueing time for our measure. In this example we use a relative width of 10%. We are then asked how many extra sampling periods are to be run with the criteria satisfied. The default is 0. The simulation will continue until this number plus one of *successive* sampling periods satisfy the criteria. Extra sampling periods force the simulation to run longer and thus

can help overcome some of the small sample problems of the sequential rule, e.g., on a very short run severe underestimates of the confidence interval width may result in the criteria being accepted. For further discussion of this problem, and the sequential stopping rule in general, see Lavenberg and Sauer [LAVE77].

Using EVAL again, we get

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 20:53:25  DATE: 03/16/82
MODEL:CSMWM6
THINKTIME:10
USERS:30
PAGEFRAMES:128
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      5605.87500
                        CPU TIME:           95.89
                NUMBER OF EVENTS:          214365
                NUMBER OF CYCLES:             247

WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.40792(3.24079,3.57505) 9.8%

WHAT:
CONTINUE RUN:yes

EXTRA SAMPLING PERIODS:1

SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
SAMPLING PERIOD END: MEMORY DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      6521.33203
                        CPU TIME:          111.60
                NUMBER OF EVENTS:          249175
                NUMBER OF CYCLES:             276

WHAT:allbo
```

```
ELEMENT         UTILIZATION
MEMORY          0.84923(0.84019,0.85826) 1.8%
FLOPPYQ         0.40042(0.39030,0.41054) 2.0%
DISKQ           0.30727(0.30470,0.30983) 0.5%
CPUQ            0.89668(0.89112,0.90224) 1.1%
TERMINALSQ      0.00000(0.00000,0.00000)


ELEMENT         THROUGHPUT
MEMORY          2.23528(2.21089,2.25966) 2.2%
FLOPPYQ         1.81558(1.78307,1.84809) 3.6%
DISKQ           16.17137(16.05899,16.28377) 1.4%
CPUQ            17.98695(17.86938,18.10452) 1.3%
TERMINALSQ      2.23528(2.21089,2.25966) 2.2%
FREEMEMORY      2.23528


ELEMENT         MEAN QUEUE LENGTH
MEMORY          7.64376(7.32707,7.96046) 8.3%
FLOPPYQ         0.59513(0.57135,0.61890) 8.0%
DISKQ           0.41651(0.41199,0.42102) 2.2%
CPUQ            2.46919(2.42749,2.51090) 3.4%
TERMINALSQ      22.35623(22.03954,22.67293) 2.8%


ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
MEMORY          3.98528
FLOPPYQ         0.86826
DISKQ           0.71621
CPUQ            1.37777
TERMINALSQ      3.98528


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.41960(3.25787,3.58134) 9.5%
FLOPPYQ         0.32779(0.31884,0.33673) 5.5%
DISKQ           0.02576(0.02558,0.02593) 1.4%
CPUQ            0.13728(0.13547,0.13908) 2.6%
TERMINALSQ      10.00154(9.86644,10.13664) 2.7%


ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.42467
FLOPPYQ         0.31493
DISKQ           0.02496
CPUQ            0.15238
TERMINALSQ      9.90722


ELEMENT         MEAN TOKENS IN USE
MEMORY          108.70111(107.54485,109.85739) 2.1%
```

```
ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          127.99998(127.99998,128.00000) 0.0%


ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.01489(0.01194,0.01783) 0.6%
                  1:0.03278(0.02788,0.03767) 1.0%
                  2:0.05154(0.04538,0.05769) 1.2%
                  3:0.06642(0.06011,0.07273) 1.3%
                  4:0.07467(0.06810,0.08125) 1.3%
                  5:0.08610(0.07853,0.09367) 1.5%
                  6:0.09373(0.08592,0.10154) 1.6%
                  7:0.08709(0.08065,0.09352) 1.3%
                  8:0.08705(0.08114,0.09297) 1.2%
                  9:0.08360(0.07667,0.09053) 1.4%
                 10:0.07682(0.07039,0.08324) 1.3%
                 11:0.06680(0.05996,0.07363) 1.4%
                 12:0.05581(0.04912,0.06250) 1.3%
                 13:0.04234(0.03564,0.04904) 1.3%
                 14:0.03165(0.02536,0.03793) 1.3%
                 15:0.01940(0.01449,0.02431) 1.0%


ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.15689(0.14283,0.17095) 2.8%
                2.00E+00:0.32428(0.30169,0.34686) 4.5%
                3.00E+00:0.50051(0.47290,0.52812) 5.5%
                4.00E+00:0.65123(0.62523,0.67723) 5.2%
                5.00E+00:0.77636(0.75315,0.79957) 4.6%
                6.00E+00:0.86033(0.84071,0.87995) 3.9%
                7.00E+00:0.91583(0.90130,0.93035) 2.9%
                8.00E+00:0.95122(0.94095,0.96150) 2.1%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          21
FLOPPYQ         5
DISKQ           6
CPUQ            7
TERMINALSQ      30


ELEMENT         MAXIMUM QUEUEING TIME
MEMORY          18.26566
FLOPPYQ         2.61646
DISKQ           0.29332
CPUQ            1.94650
TERMINALSQ      112.28362
```

```
WHAT:
CONTINUE RUN:no

THINKTIME:
```

The simulation ran for a total of 6 sampling periods.

The point estimate for mean response time is 3.41 seconds, a little lower than what we got in the last section, and the confidence interval estimate is (3.24, 3.58). The relative width is 9.8%. When we specified that the run was to continue until the stopping criteria had been satisfied for two successive sampling periods, one more sampling period was required. There was a slight increase in the mean response time point estimate and both ends of the interval.

We can reasonably conclude, based on either this run or the last run of Section 5.1, that memory contention has significantly raised the mean response time above the 2.91 second estimate for the model without memory contention.

We will indicate how the regenerative method can be applied to most of the remaining examples in this document, as we discuss those examples.

5.3. The Spectral Method

The spectral method is a third method provided for confidence interval estimates for equilibrium measures. Most methods in classical statistics for estimating confidence intervals depend on having items of data that are "independent and identically distributed." The method of independent replications achieves this "i.i.d." property by the protocol which repeats the simulation. The regenerative method depends on being able to observe the i.i.d. property during the simulation run. The spectral method does not depend on the i.i.d. property. Rather, it explicitly takes into consideration the correlation between data items in the simulation, e.g., the dependencies between successive queueing times for a given queue. This is done without user awareness, other than the availability of confidence intervals, so the dialogue for simulation using the the spectral method is essentially the same as simulation without confidence intervals. A sequential stopping rule is available with the spectral method, a slightly different rule than the one used with the regenerative method. A significant advantage of the spectral method over independent replications is that we can make a single (long) simulation run instead of multiple (shorter) runs and thus we need not be as concerned about the effects of the choice of initial state. The spectral method applies to equilibrium behavior of all models simulated using RESQ, not just those with regenerative properties. For statistical discussion of the spectral method, see Heidelberger and Welch [HEID81].

With model csmwm, we can edit as follows,

```
edit csmwm rq2inp
EDIT:
locate/CONFIDENCE/
   CONFIDENCE INTERVAL METHOD:none
delete *
EOF:
file
R; T=0.06/0.21 15:34:37
```

and then use SETUP:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   CONFIDENCE INTERVAL METHOD:spectral
   INITIAL STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      INIT POP:users
   CHAIN:
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:no
      CONFIDENCE INTERVAL QUEUES:memory memory
         MEASURES:qt qtd
      CONFIDENCE INTERVAL QUEUES:
      CONFIDENCE INTERVAL NODES:
   RUN LIMITS-
      SIMULATED TIME:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:500
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:5
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.86/1.94 15:35:51
```

The differences from the dialogue for simulation without confidence intervals are the SEQUENTIAL STOPPING RULE: prompt and the following section for specifying the queues and nodes which will have confidence intervals determined and the performance measures which will have confidence intervals determined. The only valid codes for the measures are "qt" for mean queueing time and "qtd" for queueing time distribution. We will temporarily defer discussion of the sequential stopping rule.

We could then get the following RQ2PRNT file:

```
RESQ2 VERSION DATE: MARCH 11, 1982 -  TIME: 07:43:26  DATE: 03/17/82
MODEL:CSMWM
THINKTIME:10
USERS:30
PAGEFRAMES:128
RUN END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       222.28853
                        CPU TIME:            3.81
                NUMBER OF EVENTS:            8669

WHAT:qtbo(memory)


ELEMENT          MEAN QUEUEING TIME
MEMORY           3.58395(2.64355,4.52434) 52.5%
```

```
WHAT:
CONTINUE RUN:yes

LIMIT - MEMORY DEPARTURES:1000

RUN END: MEMORY DEPARTURE LIMIT
RUN END: CPU LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       288.99683
                        CPU TIME:            5.16
                NUMBER OF EVENTS:           11361

WHAT:qtbo(memory)


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.68594(3.00150,4.37038) 37.1%

WHAT:nd(memory)


ELEMENT         NUMBER OF DEPARTURES
MEMORY          659

WHAT:
CONTINUE RUN:yes

LIMIT - CP SECONDS:40

RUN END: MEMORY DEPARTURE LIMIT
RUN END: CPU LIMIT
RUN END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:       444.63232
                        CPU TIME:            7.73
                NUMBER OF EVENTS:           16874

WHAT:qtbo(memory)


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.11431(2.45703,3.77160) 42.2%

WHAT:
CONTINUE RUN:yes

LIMIT - MEMORY DEPARTURES:2000

RUN END: MEMORY DEPARTURE LIMIT
RUN END: CPU LIMIT
RUN END: MEMORY DEPARTURE LIMIT
RUN END: MEMORY DEPARTURE LIMIT
```

```
NO ERRORS DETECTED DURING SIMULATION.

                    SIMULATED TIME:       888.56274
                          CPU TIME:           15.21
                  NUMBER OF EVENTS:           33671

WHAT:qtbo(memory)


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.24788(2.86123,3.63452) 23.8%

WHAT:allbo


ELEMENT         UTILIZATION
MEMORY          0.83603
FLOPPYQ         0.40747
DISKQ           0.30796
CPUQ            0.88254
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
MEMORY          2.25082
FLOPPYQ         1.79503
DISKQ           16.02138
CPUQ            17.81641
TERMINALSQ      2.26095
FREEMEMORY      2.25082


ELEMENT         MEAN QUEUE LENGTH
MEMORY          7.32865
FLOPPYQ         0.59840
DISKQ           0.41833
CPUQ            2.41700
TERMINALSQ      22.67134


ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
MEMORY          3.88107
FLOPPYQ         0.85949
DISKQ           0.72055
CPUQ            1.41269
TERMINALSQ      3.88107


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.24788(2.86123,3.63452) 23.8%
FLOPPYQ         0.33337
DISKQ           0.02611
CPUQ            0.13563
TERMINALSQ      9.91472
```

```
ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.27975
FLOPPYQ         0.31128
DISKQ           0.02586
CPUQ            0.15440
TERMINALSQ      9.91142


ELEMENT         MEAN TOKENS IN USE
MEMORY          107.01234


ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          128.00000


ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.02415
                  1:0.04100
                  2:0.05174
                  3:0.07241
                  4:0.08038
                  5:0.07820
                  6:0.08513
                  7:0.07596
                  8:0.09054
                  9:0.09684
                 10:0.08029
                 11:0.07770
                 12:0.05883
                 13:0.03152
                 14:0.02384
                 15:0.01460


ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.17350(0.13270,0.21430) 8.2%
                2.00E+00:0.33300(0.27786,0.38814) 11.0%
                3.00E+00:0.51650(0.44371,0.58929) 14.6%
                4.00E+00:0.67450(0.60723,0.74177) 13.5%
                5.00E+00:0.80150(0.74831,0.85469) 10.6%
                6.00E+00:0.88600(0.85188,0.92012) 6.8%
                7.00E+00:0.93800(0.91297,0.96303) 5.0%
                8.00E+00:0.96600(0.94776,0.98424) 3.6%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          21
```

```
FLOPPYQ         5
DISKQ           6
CPUQ            7
TERMINALSQ      30


ELEMENT         MAXIMUM QUEUEING TIME
MEMORY          13.46424
FLOPPYQ         2.23200
DISKQ           0.29332
CPUQ            1.90131
TERMINALSQ      76.31847

WHAT:
CONTINUE RUN:no

THINKTIME:
```

This run gives the same results as the last run of Section 4 but also provides confidence intervals and results at other run limits. The mean response time confidence interval at 1000 departures, (2.46, 3.77), and the interval at 2000 departures, (2.86, 3.63), contain the value for the numerically solved model without memory contention, 2.91, and are sufficiently wide (respective relative widths of 42% and 24%) that we cannot draw conclusions about memory contention effects.

This suggests that the initial run was much too short and that we should probably continue the run further. We need a longer run, but how much longer? Rather than proceed in the above manner of lengthening the run and periodically examining the results, we can use the sequential stopping rule, which automates essentially this procedure. The sequential stopping rule allows us to specify the simulation run length in terms of desired widths of confidence intervals, subject to the usual limit on CPU time. The simulation runs for an initial length, e.g., 2000 memory departures, and then confidence intervals are obtained. If the intervals do not meet the width criteria, the simulation continues with new limits which increase the total run length by roughly 50%. Then new estimates are made and a new decision to terminate or continue is reached. This continues until the criteria are satisfied or the CPU limit is reached. The parts of the run are referred to as "sampling periods."

As before, we can edit the dialogue file,

```
edit csmwm rq2inp
EDIT:
locate/SEQUENTIAL
   SEQUENTIAL STOPPING RULE:no
delete *
EOF:
file
R; T=0.06/0.28 15:45:50
```

and use SETUP again:

```
SETUP csmwm
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   SEQUENTIAL STOPPING RULE:yes
```

```
      CONFIDENCE INTERVAL QUEUES:memory memory
         MEASURES:qt qtd
         ALLOWED WIDTHS:10 10
      CONFIDENCE INTERVAL QUEUES:
      CONFIDENCE INTERVAL NODES:
      EXTRA SAMPLING PERIODS:edit
EDIT:
case m
locate/ALLOWED WIDTHS:
         ALLOWED WIDTHS:10 10
i    INITIAL PORTION DISCARDED:10 /*percent of initial period*/
file
MODEL IS CSMWM
CONTINUING WITH MODEL DEFINITION...
   INITIAL PERIOD LIMITS-
      SIMULATED TIME:
      EVENTS:
      QUEUES FOR DEPARTURE COUNTS:memory
         DEPARTURES:2000
      QUEUES FOR DEPARTURE COUNTS:
      NODES FOR DEPARTURE COUNTS:
   LIMIT - CP SECONDS:300
   TRACE:no
END
 NO FATAL ERRORS DETECTED DURING THE COMPILATION.
R; T=0.83/2.06 15:48:02
```

We are asked for a list of queues where we are to obtain and check confidence interval widths at the end of a sampling period. Then we are asked what measures are to be considered. Then we are asked what widths are to be allowed. For the queueing time distribution (qtd), each gathered point of the distribution is checked and must satisfy the width criteria. For the queueing time distribution, which can only have values in the [0, 1] interval, the width specified is absolute width in percent, i.e., the criterion is that $200\times\delta$ be less than the specified width, where the confidence interval is $(p - \delta, p + \delta)$. For mean queueing time (qt), the width is relative width in percent, i.e., the criterion is that $200\times\delta/p$ be less than the specified width. (Where p is zero, the criteria is not satisfied.)

We are then asked how many extra sampling periods are to be run with the criteria satisfied. The default is 0. The simulation will continue until this number plus one of *successive* sampling periods satisfy the criteria. Extra sampling periods force the simulation to run longer and thus can help overcome some of the problems of the sequential rule, e.g., on a very short run severe underestimates of the confidence interval width may result in the criteria being accepted. For further discussion of this problem, and the sequential stopping rule in general, see Heidelberg and Welch [HEID81].

Rather than giving a value to the EXTRA SAMPLING PERIODS: prompt, where we are willing to accept the zero default, we give the special reply "edit" so that we can insert an INITIAL PORTION DISCARDED: line in the dialogue file. This portion of the first sampling period will be discarded.

Using EVAL again, we get

```
RESQ2 VERSION DATE: APRIL 3, 1982 -  TIME: 17:56:07  DATE: 04/03/82
MODEL:CSMWM6S
```

```
THINKTIME:10
USERS:30
PAGEFRAMES:128
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.   3786 DISCARDED EVENTS

                  SIMULATED TIME:     6096.76563
                        CPU TIME:        107.39
                 NUMBER OF EVENTS:       232927

WHAT:qtbo(memory)


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.39314(3.24376,3.54253) 8.8%

WHAT:
CONTINUE RUN:yes

EXTRA SAMPLING PERIODS:1

SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
SAMPLING PERIOD END: MEMORY DEPARTURE LIMIT
NO ERRORS DETECTED DURING SIMULATION.   3786 DISCARDED EVENTS

                  SIMULATED TIME:     9200.50391
                        CPU TIME:        160.28
                 NUMBER OF EVENTS:       351644

WHAT:allbo


ELEMENT         UTILIZATION
MEMORY          0.84992
FLOPPYQ         0.39839
DISKQ           0.30647
CPUQ            0.89810
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
MEMORY          2.22836
FLOPPYQ         1.81512
DISKQ           16.18085
```

```
CPUQ            17.99608
TERMINALSQ      2.22803
FREEMEMORY      2.22836


ELEMENT         MEAN QUEUE LENGTH
MEMORY          7.66801
FLOPPYQ         0.59354
DISKQ           0.41624
CPUQ            2.47630
TERMINALSQ      22.33199


ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
MEMORY          3.97375
FLOPPYQ         0.87024
DISKQ           0.71857
CPUQ            1.37433
TERMINALSQ      3.97375


ELEMENT         MEAN QUEUEING TIME
MEMORY          3.43985(3.33988,3.53982) 5.8%
FLOPPYQ         0.32700
DISKQ           0.02572
CPUQ            0.13760
TERMINALSQ      10.01154


ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
MEMORY          2.42777
FLOPPYQ         0.31059
DISKQ           0.02501
CPUQ            0.15214
TERMINALSQ      10.01072


ELEMENT         MEAN TOKENS IN USE
MEMORY          108.78951


ELEMENT         MEAN TOTAL TOKENS IN POOL
MEMORY          128.00000


ELEMENT         QUEUE LENGTH DISTRIBUTION
MEMORY            0:0.01557
                  1:0.03242
                  2:0.04968
                  3:0.06637
                  4:0.07511
                  5:0.08407
                  6:0.08908
                  7:0.08593
```

```
                8:0.09090
                9:0.08693
               10:0.07914
               11:0.06877
               12:0.05468
               13:0.04200
               14:0.03094
               15:0.01930


ELEMENT         QUEUEING TIME DISTRIBUTION
MEMORY          1.00E+00:0.15520(0.14791,0.16250) 1.5%
                2.00E+00:0.32216(0.30920,0.33513) 2.6%
                3.00E+00:0.49410(0.47693,0.51127) 3.4%
                4.00E+00:0.64569(0.62781,0.66358) 3.6%
                5.00E+00:0.77144(0.75471,0.78817) 3.3%
                6.00E+00:0.85875(0.84547,0.87202) 2.7%
                7.00E+00:0.91494(0.90398,0.92589) 2.2%
                8.00E+00:0.95137(0.94339,0.95935) 1.6%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
MEMORY          21
FLOPPYQ         6
DISKQ           6
CPUQ            7
TERMINALSQ      30


ELEMENT         MAXIMUM QUEUEING TIME
MEMORY          19.19725
FLOPPYQ         2.61646
DISKQ           0.29332
CPUQ            1.94650
TERMINALSQ      112.28362

WHAT:
CONTINUE RUN:no

THINKTIME:
```

The simulation ran for a total of 6 sampling periods. The initial 3786 events of the first sampling period were discarded. The point estimate for mean response time is 3.39 seconds and the confidence interval estimate is (3.24, 3.54). The relative width is 8.8%. When we specified that the run was to continue until the stopping criteria had been satisfied for two successive sampling periods, one more sampling period was required. There was an increase in the mean response point estimate and the lower end of the interval.

We can reasonably conclude, based on either this run or the last runs of Sections 5.1 and 5.2, that memory contention has significantly raised the mean response time above the 2.91 second estimate for the model without memory contention.

6. SOURCES AND SINKS

All of the models we have considered so far have a fixed population of jobs with no mechanisms for external arrivals of jobs at the network or departures of jobs from the network. The mechanisms provided for these purposes are nodes called "sources" and "sinks," respectively. As we said before, routing chains with sources and sinks are "open" chains.

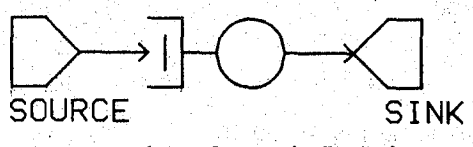


Figure 6.1 - Queue in Isolation

The simplest possible open queueing network is a single queue in isolation, as shown in Figure 6.1. Let us use RESQ to examine the classical "M/M/1" queue, i.e., a fcfs queue with exponential arrival and service times.

```
MODEL:MM1
   METHOD:numerical
   QUEUE:q
      TYPE:fcfs
      CLASS LIST:c
         SERVICE TIMES:4
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:5
      :s->c->sink
END
```

All of the above dialogue should be familiar up to the prompt for the chain type. After giving the type as open, there is a prompt for a list of source names and then a prompt for a list of arrival time distributions. The name "sink" is predefined as the only sink. The same sink is shared by all open chains. It is illegal to have a routing transition with a source on the right hand side or a sink on the left hand side.

Now we can get the results from EVAL:

```
RESQ2 VERSION DATE: OCTOBER 2, 1981
MODEL:MM1
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:all


ELEMENT          UTILIZATION
Q                0.80000


ELEMENT          THROUGHPUT
Q                0.20000
```

```
ELEMENT          MEAN QUEUE LENGTH
Q                4.00000


ELEMENT          MEAN QUEUEING TIME
Q                19.99998


ELEMENT          OPEN CHAIN POPULATION
CH               4.00000


ELEMENT          OPEN CHAIN RESPONSE TIME
CH               19.99998

WHAT:
```

The open chain population is the mean number of jobs in the open chain and the open chain response time is the mean time spent in the chain by a job.

Though the queueing time distribution for the M/M/1 queue is known to be exponential (see Kobayashi [KOBA78]), it is not available from the numerical solution component of RESQ. We can use simulation to obtain estimates of the queueing time distribution, as would be necessary if we were dealing with a system without known solution for the queueing time distribution. The following dialogue file would be adequate:

```
MODEL:mm1
   METHOD:simulation
   QUEUE:q
      TYPE:fcfs
      CLASS LIST:c
         SERVICE TIMES:4
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:5
      :s->c->sink
   QUEUES FOR QUEUEING TIME DIST:q
      VALUES:10 20 30 40 50
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:q
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:q
         DEPARTURES:10000
   LIMIT - CP SECONDS:100
   TRACE:no
END
```

We have not given an explicit definition of the regeneration and initial states. If we do not give an explicit definition of these states for a chain, then there will be no jobs in the chain in these states. (Thus we must give explicit definitions for these states for closed chains.) It can be shown that the empty state is the most frequently occurring state for the M/M/1 queue and for many open networks. Thus it is reasonable as well as convenient to use the empty state as we have done.

Though the M/M/1 queue is very simple to solve algebraically, it can require what seem to be very long simulation runs for reasonable results. Knowing this in advance, we set the departure limit at 10,000 departures.

Now using EVAL we get

```
RESQ2 VERSION DATE: OCTOBER 3, 1981
MODEL:MM1
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:     3.0210E+05
                        CPU TIME:           42.39
                 NUMBER OF EVENTS:          120132
                 NUMBER OF CYCLES:           12268

WHAT:qtbo


ELEMENT         MEAN QUEUEING TIME
Q               18.64873(17.76196,19.53551) 9.5%

WHAT:utbo


ELEMENT         UTILIZATION
Q               0.79771(0.79058,0.80484) 1.4%

WHAT:
CONTINUE RUN:yes

EXTRA SAMPLING PERIODS:1

SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
```

```
NO ERRORS DETECTED DURING SIMULATION.

                SIMULATED TIME:     4.5137E+05
                      CPU TIME:          63.52
               NUMBER OF EVENTS:          180320
               NUMBER OF CYCLES:           18183

WHAT:qtbo

ELEMENT         MEAN QUEUEING TIME
Q               19.19112(18.35539,20.02684) 8.7%

WHAT:allbo

ELEMENT         UTILIZATION
Q               0.79982(0.79388,0.80575) 1.2%

ELEMENT         THROUGHPUT
Q               0.19975(0.19869,0.20081) 1.1%
S               0.19975
SINK            0.19975

ELEMENT         MEAN QUEUE LENGTH
Q               3.83337(3.65683,4.00990) 9.2%

ELEMENT         STANDARD DEVIATION OF QUEUE LENGTH
Q               4.16076

ELEMENT         MEAN QUEUEING TIME
Q               19.19112(18.35539,20.02684) 8.7%

ELEMENT         STANDARD DEVIATION OF QUEUEING TIME
Q               18.49261

ELEMENT         MEAN TOKENS IN USE

ELEMENT         MEAN TOTAL TOKENS IN POOL

ELEMENT         QUEUE LENGTH DISTRIBUTION

ELEMENT         QUEUEING TIME DISTRIBUTION
Q               1.00E+01:0.39427(0.38311,0.40542) 2.2%
                2.00E+01:0.64136(0.62683,0.65589) 2.9%
```

```
                3.00E+01:0.79021(0.77598,0.80443) 2.8%
                4.00E+01:0.87709(0.86450,0.88967) 2.5%
                5.00E+01:0.93015(0.91954,0.94076) 2.1%


ELEMENT         DISTRIBUTION OF TOKENS IN USE


ELEMENT         DISTRIBUTION OF TOTAL TOKENS IN POOL


ELEMENT         MAXIMUM QUEUE LENGTH
Q               35


ELEMENT         MAXIMUM QUEUEING TIME
Q               151.03687


ELEMENT         OPEN CHAIN POPULATION
CH              3.83337(3.65683,4.00990) 9.2%


ELEMENT         OPEN CHAIN RESPONSE TIME
CH              19.19112(18.35538,20.02684) 8.7%

WHAT:
CONTINUE RUN:no
```

Arrivals from sources are events as well as service completions. Thus the initial number of departures is 60066, which seems like a large number to obtain a 10% confidence interval width for the mean queueing time, but this illustrates the variability of the M/M/1 queue at moderately high utilizations. Note that open networks and queues with more variable service times are likely to require even longer runs when utilizations are high. Note also that even with this seemingly long run, the point estimate for mean queueing time is well below the true value and the confidence interval does not contain the true value. Only when we continue the run, requiring that the width criterion be satisfied for two successive sampling periods, do we get a confidence interval which contains the true value. Three additional sampling periods are required before the criterion is satisfied for two successive sampling periods.

Sources and sinks are used in exactly the same manner in general networks as in this example here. We will have more examples with sources and sinks in subsequent sections. It is possible to have the arrival rate of jobs from sources of a chain vary during the simulation, as we shall see in Section 8.

With the regenerative method it is almost always most appropriate to use the empty state for regeneration and initial states for open chains. Though some other state may occur more frequently, it is usually not worth the effort of looking for such a state. If the empty state does not occur frequently enough then it is usually not practical to use the regenerative method.

7. CHAINS

All of our examples so far have had a single routing chain. Also, all of our examples have had at most one node of a given type per queue (the passive queues have had two nodes, an allocate and a release). Usually we use more than one routing chain when we want to distinguish between different types of jobs. A queue must have at least one node for each chain which visits the queue, so more than one chain usually implies more than one node at at least one queue. (Otherwise we would actually have disjoint subnetworks.) We may want to have more than one node (of a given type) per queue even if we have only one routing chain and/or we may want to have a queue with several nodes of the same type which belong to the same chain in a model with several chains. Chains are disjoint in the sense that a node of one chain may not belong to another chain (with the exception that all open chains share the same sink). With models solved by the RESQ numerical component, a job at a source or class of a given chain must be able to reach any class of the chain unless it goes to a sink first. A similar requirement (with nodes in general other than sources and sinks replacing classes in the above) does *not* hold for models to be simulated, but most models will satisfy the condition. (In Section 8.2 we will see a model which does not satisfy this condition.)

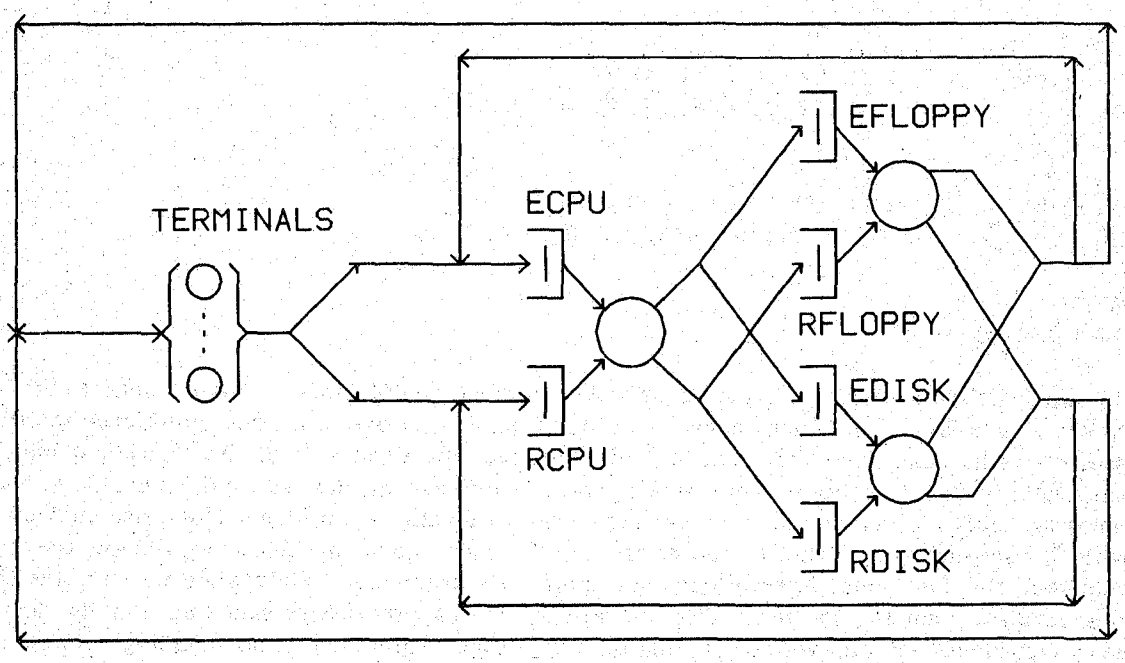


Figure 7.1 - Single Chain Model

Before we consider models with multiple chains, let us consider a model with a single chain but multiple classes at some active queues. Suppose in our original computer system model we wished to distinguish between commands for editing, which represent the bulk of commands in many systems, and other commands, e.g., for compiling and running programs. Figure 7.1 shows a possible modification for this purpose. We have separate classes for editing and "running" at the CPU, floppy disk and hard disk queues. A job leaving the terminals is determined to be either an editing or a running job and the distinction is preserved until the job returns to the terminals. The following is a possible dialogue file for this model:

```
MODEL:csmer
   /*Computer System Model with Editing and "Running" users*/
   METHOD:numerical
   NUMERIC PARAMETERS:thinktime users
   NUMERIC IDENTIFIERS:floppytime disktime ecputime rcputime
```

```
      FLOPPYTIME:.22
      DISKTIME:.019
      ECPUTIME:.05
      RCPUTIME:.075
   NUMERIC IDENTIFIERS:ecycles rcycles
      ECYCLES:4
      RCYCLES:60
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:efloppy rfloppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:edisk
         SERVICE TIMES:disktime
      CLASS LIST:rdisk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:ecpu rcpu
         SERVICE TIMES:ecputime rcputime
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->ecpu rcpu;.95 .05
      :ecpu->efloppy edisk;.1 .9
      :efloppy->terminals ecpu;1/ecycles 1-1/ecycles
      :edisk->terminals ecpu;1/ecycles 1-1/ecycles
      :rcpu->rfloppy rdisk;.2 .8
      :rfloppy->terminals rcpu;1/rcycles 1-1/rcycles
      :rdisk->terminals rcpu;1/rcycles 1-1/rcycles
END
```

The model assumes that the service times are the same for both classes at floppyq and diskq and different at cpuq. With exact numerical solution we must assume the same exponential distribution for all classes at fcfs queues. The model also has different mean numbers of cycles for the editing and running subnetworks. (We dropped "cpio" from the identifiers to keep under 11 characters.) Note that in the definition of floppyq a single service time value is given for all classes in the list. In the definition of diskq we used two class lists (which could each have more than one class if there were more classes at the queue.)

Using EVAL we can get

```
RESQ2 VERSION DATE: MARCH 23, 1982 -  TIME: 15:22:56  DATE: 04/01/82
MODEL:CSMER
THINKTIME:10
USERS:30
NO ERRORS DETECTED DURING NUMERICAL SOLUTION

WHAT:all
```

ELEMENT	UTILIZATION
FLOPPYQ	0.48863
EFLOPPY	0.18947
RFLOPPY	0.29916
DISKQ	0.25062
EDISK	0.14727
RDISK	0.10335
CPUQ	0.94055
ECPU	0.43061
RCPU	0.50993
TERMINALSQ	0.00000

ELEMENT	THROUGHPUT
FLOPPYQ	2.22105
EFLOPPY	0.86122
RFLOPPY	1.35982
DISKQ	13.19032
EDISK	7.75102
RDISK	5.43930
CPUQ	15.41136
ECPU	8.61224
RCPU	6.79912
TERMINALSQ	2.26638

ELEMENT	MEAN QUEUE LENGTH
FLOPPYQ	0.92724
EFLOPPY	0.35954
RFLOPPY	0.56770
DISKQ	0.33208
EDISK	0.19514
RDISK	0.13694
CPUQ	6.07688
ECPU	2.78219
RCPU	3.29469
TERMINALSQ	22.66380

ELEMENT	MEAN QUEUEING TIME
FLOPPYQ	0.41748
EFLOPPY	0.41748
RFLOPPY	0.41748
DISKQ	0.02518
EDISK	0.02518
RDISK	0.02518
CPUQ	0.39431
ECPU	0.32305
RCPU	0.48458
TERMINALSQ	10.00000

WHAT:
THINKTIME:

With numerical solution, RESQ does not provide utilization estimates for classes at queues with multiple servers and/or queue dependent service rates. We can estimate mean response times as before, e.g., by Little's Rule the mean editing response time is (2.78219 + .35954 + .19514)/(2.26638 × .95) = 1.550 seconds, the mean running response time is (3.29469 + .56770 + .13694)/(2.26638 × .05) = 35.293 seconds and the mean overall response time is (30 − 22.66380)/2.26638 = 3.237 seconds.

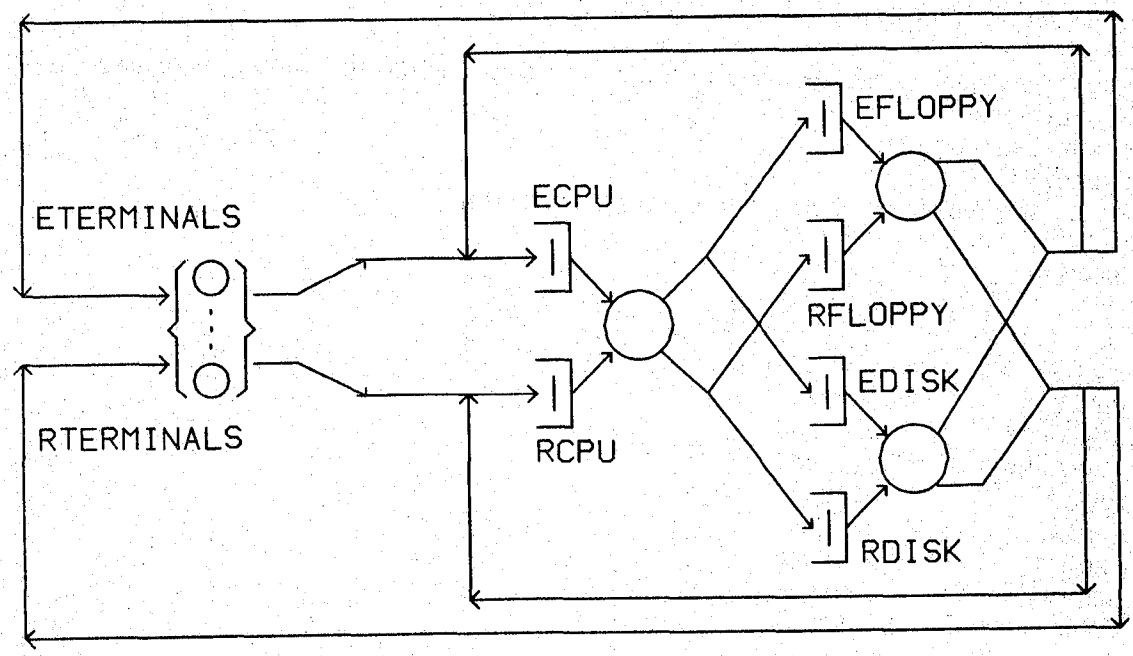


Figure 7.2 - Model with Two Closed Chains

In the model with the single chain, it is unlikely that a user's successive commands will be for "running." Usually an editing command will follow a command for "running." Suppose that this is not realistic, that users stay in an editing or running mode for quite a while so that it seems as if there are editing and running *users* rather than merely editing and running commands. Then it might be more appropriate to have two chains as in Figure 7.2. (Another possibility would be to have infrequent transitions between the chains of Figure 7.2; then the "chains" would actually be subchains, not chains.) Suppose we want to consider the system when one seventh of the users are in the running mode and the rest are in the editing mode. Then we would have the following definitions for terminalsq and the chains:

```
QUEUE:terminalsq
   TYPE:is
   CLASS LIST:eterminals rterminals
      SERVICE TIMES:thinktime
CHAIN:editing
   TYPE:closed
   POPULATION:users-ceil(users/7)
   :eterminals->ecpu
   :ecpu->efloppy edisk;.1 .9
   :efloppy->eterminals ecpu;1/ecycles 1-1/ecycles
   :edisk->eterminals ecpu;1/ecycles 1-1/ecycles
CHAIN:running
   TYPE:closed
   POPULATION:ceil(users/7)
   :rterminals->rcpu
```

```
      :rcpu->rfloppy rdisk;.2 .8
      :rfloppy->rterminals rcpu;1/rcycles 1-1/rcycles
      :rdisk->rterminals rcpu;1/rcycles 1-1/rcycles
END
```

The dialogue file before the definition of terminalsq is the same as before. The "ceil" function gives the smallest integer which is at least as large as the given argument. Now we could get the following:

```
RESQ2 VERSION DATE: MARCH 23, 1982 -  TIME: 16:03:15  DATE: 04/01/82
MODEL:CSMER
THINKTIME:10
USERS:30
NO ERRORS DETECTED DURING NUMERICAL SOLUTION

WHAT:all


ELEMENT         UTILIZATION
FLOPPYQ         0.50873
 EFLOPPY         0.18965
 RFLOPPY         0.31907
DISKQ           0.25764
 EDISK           0.14741
 RDISK           0.11023
CPUQ            0.97491
 ECPU            0.43103
 RCPU            0.54388
TERMINALSQ      0.00000
 ETERMINALS      0.00000
 RTERMINALS      0.00000


ELEMENT         THROUGHPUT
FLOPPYQ         2.31240
 EFLOPPY         0.86207
 RFLOPPY         1.45034
DISKQ           13.55993
 EDISK           7.75859
 RDISK           5.80134
CPUQ            15.87234
 ECPU            8.62066
 RCPU            7.25168
TERMINALSQ      2.27603
 ETERMINALS      2.15516
 RTERMINALS      0.12086


ELEMENT         MEAN QUEUE LENGTH
FLOPPYQ         0.97826
 EFLOPPY         0.37487
 RFLOPPY         0.60339
DISKQ           0.34472
 EDISK           0.19748
```

```
 RDISK          0.14724
CPUQ            5.91675
 ECPU           2.87599
 RCPU           3.04076
TERMINALSQ      22.76025
 ETERMINALS     21.55165
 RTERMINALS     1.20861


ELEMENT         MEAN QUEUEING TIME
FLOPPYQ         0.42305
 EFLOPPY        0.43485
 RFLOPPY        0.41604
DISKQ           0.02542
 EDISK          0.02545
 RDISK          0.02538
CPUQ            0.37277
 ECPU           0.33362
 RCPU           0.41932
TERMINALSQ      10.00000
 ETERMINALS     10.00001
 RTERMINALS     10.00000

WHAT:
THINKTIME:
```

Now the mean response time for the editing jobs would be (25 – 21.55165)/2.15516 = 1.600 seconds, the mean response time for the running jobs would be (5 – 1.20861)/.12086 = 31.370 seconds and the mean overall response time would be (30 – 22.760)/2.27603 = 3.181 seconds.

With multiple closed chains, execution times may become quite large with numerical solution when the chain populations are substantial.

Now let us suppose that we return to our original model without distinctions between interactive users and wish to add a batch workload to that model. The batch jobs could be submitted by terminal commands, for example. Figure 7.3 shows an open chain added to the original figure to represent the batch jobs. The following dialogue file could be used for Figure 7.3:

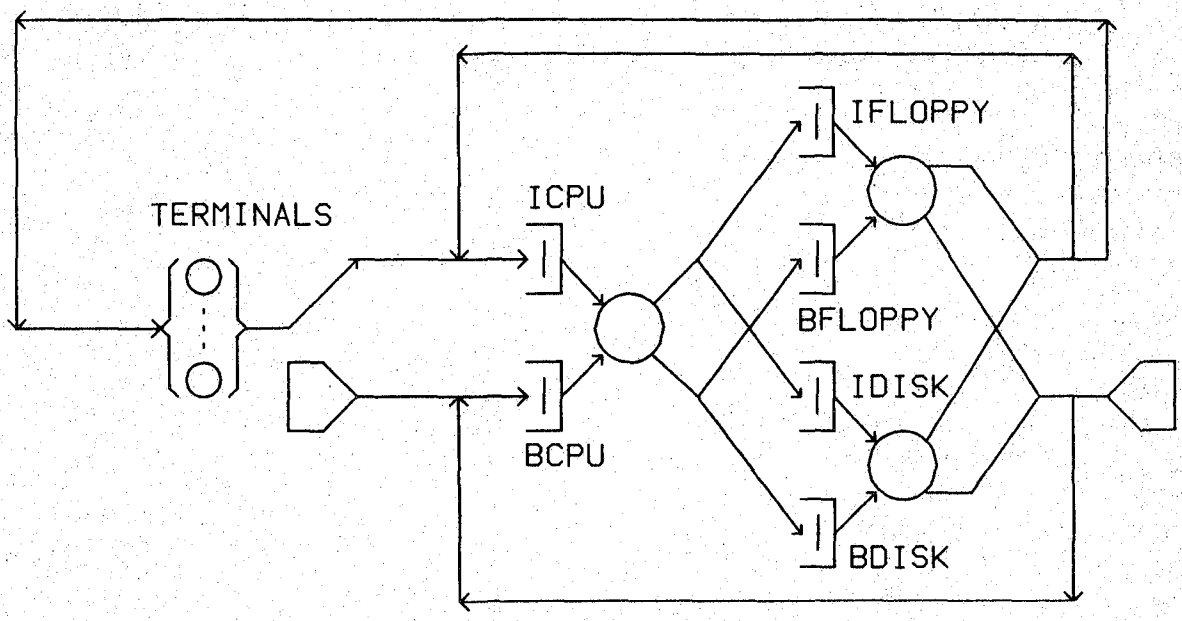


Figure 7.3 - Model with Open and Closed Chains

```
MODEL:csmib
   /*Computer System Model with Interactive users and Batch jobs*/
   METHOD:numerical
   NUMERIC PARAMETERS:thinktime users brate
   NUMERIC IDENTIFIERS:floppytime disktime icputime bcputime
      FLOPPYTIME:.22
      DISKTIME:.019
      ICPUTIME:.05
      BCPUTIME:.075
   NUMERIC IDENTIFIERS:icycles bcycles
      ICYCLES:8
      BCYCLES:100
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:ifloppy bfloppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:idisk bdisk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:icpu bcpu
         SERVICE TIMES:icputime bcputime
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
```

```
      :terminals->icpu
      :icpu->ifloppy idisk;.1 .9
      :ifloppy->terminals icpu;1/icycles 1-1/icycles
      :idisk->terminals icpu;1/icycles 1-1/icycles
   CHAIN:batch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:1/brate
      :s->bcpu
      :bcpu->bfloppy bdisk;.2 .8
      :bfloppy->sink bcpu;1/bcycles 1-1/bcycles
      :bdisk->sink bcpu;1/bcycles 1-1/bcycles
END
```

With numerical solution, RESQ requires that closed chains be defined before open chains. We assume batch jobs are submitted at rate brate. Using EVAL we can get

```
RESQ2 VERSION DATE: MARCH 23, 1982 -  TIME: 16:20:21  DATE: 04/01/82
MODEL:CSMIB
THINKTIME:10
USERS:30
BRATE:.01
NO ERRORS DETECTED DURING NUMERICAL SOLUTION

WHAT:all


ELEMENT         UTILIZATION
FLOPPYQ         0.43299
 IFLOPPY         0.38899
 BFLOPPY         0.04400
DISKQ           0.31755
 IDISK           0.30235
 BDISK           0.01520
CPUQ            0.95906
 ICPU            0.88406
 BCPU            0.07500
TERMINALSQ      0.00000


ELEMENT         THROUGHPUT
FLOPPYQ         1.96812
 IFLOPPY         1.76812
 BFLOPPY         0.20000
DISKQ           16.71306
 IDISK           15.91307
 BDISK           0.80000
CPUQ            18.68118
 ICPU            17.68118
 BCPU            0.99999
TERMINALSQ      2.21015
```

```
ELEMENT          MEAN QUEUE LENGTH
FLOPPYQ          0.75205
 IFLOPPY          0.67496
 BFLOPPY          0.07709
DISKQ            0.46135
 IDISK            0.43914
 BDISK            0.02221
CPUQ             7.41557
 ICPU             6.78440
 BCPU             0.63116
TERMINALSQ       22.10149


ELEMENT          MEAN QUEUEING TIME
FLOPPYQ          0.38212
 IFLOPPY          0.38174
 BFLOPPY          0.38545
DISKQ            0.02760
 IDISK            0.02760
 BDISK            0.02777
CPUQ             0.39695
 ICPU             0.38371
 BCPU             0.63117
TERMINALSQ       10.00000


ELEMENT          OPEN CHAIN POPULATION
BATCH            0.73047


ELEMENT          OPEN CHAIN RESPONSE TIME
BATCH            73.04666

WHAT:
THINKTIME:
```

Now the interactive mean response time is $(30 - 22.10149)/2.21015 = 3.574$ seconds. We could obtain the overall mean response time again, but this seems uninteresting for this model.

Open chains have negligible effect on the execution time of RESQ numerical solutions.

8. JOB, CHAIN AND GLOBAL VARIABLES

Job, Chain and Global Variables are available only in simulation models. These variables are used in the sense of programming language variables. Assignment statements for these variables are performed by set nodes. Expressions containing these variables may be used in defining service times, arrival times, routing, priorities, numbers of tokens to be allocated, and in most other places where numeric expressions are allowed. (The Users Guide indicates which expressions do not allow use of these variables. Except in those places where the variables are explicitly prohibited, expressions may use these variables.)

8.1. Job Variables

Job variables are used to store numeric data with individual jobs during a simulation run. Job variables are identified by the subscripted keyword "JV." The subscripts begin at 0 and may range up to a maximum specified by the user. (Job variables and chain variables are unlike RESQ arrays in that the lower bound is 0 instead of 1.) The maximum subscript is specified in response to the prompt

```
MAX JV:
```

In a dialogue file, this prompt and the reply would be inserted following the identifier declarations section. If no maximum is specified, only JV(0) and JV(1) may be used. All job variables are initialized to 0 when a job is created except for job variables of jobs created by split and fission nodes (Section 11). Job variables are represented internally as double precision floating point numbers.

Job variables are assigned values at "set" nodes. A set node performs assignment statements corresponding to assignment statements in a programming language. After queue definitions are completed the interactive mode of SETUP will prompt for a list of set nodes with the prompt "SET NODES:" and will then prompt for the assignment statements for those nodes with the prompt "ASSIGNMENT LIST:". After the ASSIGNMENT LIST: prompt there will be another SET NODES: prompt. In dialogue files the SET NODES: and ASSIGNMENT LIST: lines are inserted after the queue definitions. If more than one assignment statement is to be associated with a single set node, then this set node should be defined by a separate SET NODE and ASSIGNMENT LIST section. Some examples:

```
SET NODES:       alpha                      beta
ASSIGNMENT LIST:jv(leng)=be(1,0; 1,1)   jv(3)=jv(3)+1
SET NODES:       gamma
ASSIGNMENT LIST:jv(stime)=users/10
SET NODES:       delta
ASSIGNMENT LIST:jv(stime)=users/10 jv(p)=jv(p)+1
```

Note that use of identifiers for subscripts can improve readability. Subscripts for job variables may be expressions requiring simulation time evaluation, e.g., may involve other job variables. In this example, nodes alhpa, beta and gamma would each perform a single assignment. Set node delta would perform two assignments. A set node may perform any number of assignments, but if a set node is to perform more than one assignment, it must be defined by a separate pair of SET NODE: and ASSIGNMENT LIST: lines. The assignments are performed in the order listed. Assignments for job, chain and global variables may be mixed at a single set node.

Job Variables are very useful in service time expressions, e.g.,

```
CLASS LIST:transmit
   SERVICE TIMES:propagate+standard(jv(leng),0)/capacity
```

"Standard" is the name of a RESQ distribution specified by the mean and coefficient of variation. Coefficient of variation zero results in a constant value. The above expression would be taken as the mean of an exponential distribution if the name of a distribution was not present.

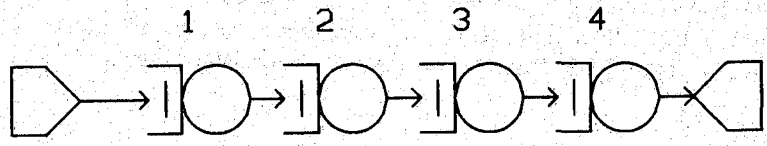


Figure 8.1 - Series Queues with Independence Assumption

The classic application of job variables is to avoid making Kleinrock's "independence assumption" in modeling communication networks. (This assumption was originally proposed in Kleinrock's Ph.D. thesis. It is discussed further in [KLEI76].) Consider Figure 8.1 and suppose that the queues represent communication links. For simplicity let us assume the processing between links is negligible. The transmission times for a given message at each link will be proportional to each other, with the proportionality determined by the link capacities (rates). In order to make analytic solution of such a network feasible, Kleinrock conjectured that one could assume the transmission times were independent and demonstrated by simulation that this was a reasonable assumption for some networks. Whether the assumption is reasonable or not depends on a number of factors, including the traffic intensity and the network topology.

The series topology of Figure 8.1 is such that the independence assumption is not appropriate. Suppose in Figure 8.1 the queues are FCFS queues with (independent) exponential service times with mean .125 second and that the arrival times are exponential with mean .25 second. Then each queue may be treated as an M/M/1 queue in isolation. The queueing times at each queue are exponential with mean .25 second. The response times from source to sink are the sum of four independent exponential times with mean .25 second, so they have a four stage Erlang distribution with mean 1 second.

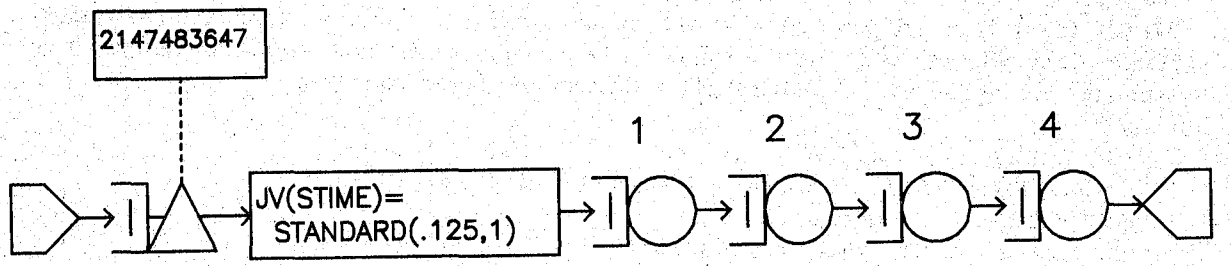


Figure 8.2 - Series Queues with Interdependence

Now suppose that we wish to test the effects of the independence assumption. Assuming that the links have the same capacities, then a message will have the same service time at each of the four queues. We can no longer treat the last three queues as M/M/1 queues because their arrival and service times are dependent and because the interarrival times at the last two are not necessarily exponential. We can simulate the system using job variables. Figure 8.2 shows a passive queue and a set node added to Figure 8.1. The passive queue is used to measure response times. It has an "infinite" number of tokens, i.e., $2^{31}-1$. A release node is not necessary; jobs holding tokens release them when they go to a sink. The numeric identifier msg__stime is given the value 0 and used in JV subscripts to indicate the JV is used for service times. The set node is used to put the service time value in JV(msg__stime). The

standard distribution with coefficient of variation one results in an exponential distribution. "JV(msg__stime)=.125" would result in a constant, not an exponential, value. The following dialogue file could be used:

```
MODEL:fourlink
   METHOD:simulation
   NUMERIC IDENTIFIERS:msg_stime
      MSG_STIME:0 /*JV to be used*/
   QUEUE:rtq
      TYPE:passive
      TOKENS:2147483647
      DSPL:fcfs
      ALLOCATE NODE LIST:beginrt
         NUMBERS OF TOKENS TO ALLOCATE:1
   QUEUE:q1
      TYPE:fcfs
      CLASS LIST:c1
         SERVICE TIMES:standard(jv(msg_stime),0)
   QUEUE:q2
      TYPE:fcfs
      CLASS LIST:c2
         SERVICE TIMES:standard(jv(msg_stime),0)
   QUEUE:q3
      TYPE:fcfs
      CLASS LIST:c3
         SERVICE TIMES:standard(jv(msg_stime),0)
   QUEUE:q4
      TYPE:fcfs
      CLASS LIST:c4
         SERVICE TIMES:standard(jv(msg_stime),0)
   SET NODES:set_stime
   ASSIGNMENT LIST:jv(msg_stime)=standard(.125,1)
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:.25
      :s->beginrt->set_stime->c1->c2->c3->c4->sink
   QUEUES FOR QUEUEING TIME DIST:rtq
      VALUES:.5 1 1.5 2 2.5
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:rtq
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:rtq
         DEPARTURES:10000
   LIMIT - CP SECONDS:100
   TRACE:no
END
```

EVAL will give us the following:

```
RESQ2 VERSION DATE: OCTOBER 9, 1981
MODEL:FOURLINK
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      2512.58301
                        CPU TIME:           29.56
                 NUMBER OF EVENTS:           50285
                 NUMBER OF CYCLES:             806

WHAT:utbo


ELEMENT          UTILIZATION
RTQ              2.2010E-09(2.1094E-09,2.2926E-09) 0.0%
Q1               0.50886(0.49823,0.51949) 2.1%
Q2               0.50886(0.49823,0.51949) 2.1%
Q3               0.50886(0.49823,0.51949) 2.1%
Q4               0.50886(0.49823,0.51949) 2.1%

WHAT:tpbo(rtq)


ELEMENT          THROUGHPUT
RTQ              4.00265(3.94119,4.06411) 3.1%

WHAT:qlbo


ELEMENT          MEAN QUEUE LENGTH
RTQ              4.72664(4.52998,4.92329) 8.3%
Q1               1.00657(0.95876,1.05437) 9.5%
Q2               1.06901(1.02551,1.11251) 8.1%
Q3               1.25355(1.20049,1.30662) 8.5%
Q4               1.39750(1.33610,1.45890) 8.8%

WHAT:qtbo


ELEMENT          MEAN QUEUEING TIME
RTQ              1.18087(1.13909,1.22266) 7.1%
Q1               0.25148(0.24081,0.26214) 8.5%
Q2               0.26708(0.25784,0.27631) 6.9%
Q3               0.31318(0.30191,0.32445) 7.2%
Q4               0.34914(0.33591,0.36238) 7.6%

WHAT:sdqt


ELEMENT          STANDARD DEVIATION OF QUEUEING TIME
RTQ              0.72737
Q1               0.24087
Q2               0.19031
Q3               0.19855
```

```
Q4              0.20232

WHAT:qtdbo


ELEMENT         QUEUEING TIME DISTRIBUTION
RTQ             5.00E-01:0.17590(0.16287,0.18892) 2.6%
                1.00E+00:0.46107(0.43827,0.48388) 4.6%
                1.50E+00:0.71473(0.69109,0.73836) 4.7%
                2.00E+00:0.86944(0.85158,0.88731) 3.6%
                2.50E+00:0.94094(0.92910,0.95277) 2.4%

WHAT:
CONTINUE RUN:no
```

Note the increasing mean queue lengths and mean queueing times as we progress from queues 1 to 4. The response time has a mean 18% higher than with the independence assumption and is more variable than if it had a four stage Erlang distribution. The queueing times at queues 2, 3 and 4 are less variable than the exponential distribution.

We will have more examples of use of job variables in subsequent sections.

8.2. Chain Variables

Chain variables are analogous to job variables except that the numeric data is associated with chains rather than individual jobs. *Chain variables have only one unique function, to control the rates of sources of the chains. Though chain variables can be used for other purposes, it will usually be more appropriate to use global variables (Section 8.3) for these purposes.*

Chain variables are identified by the subscripted keyword "CV." The subscripts begin at 0 and may range up to a maximum specified by the user. Only chain variable 0 affects sources. The maximum subscript can be specified only in dialogue files. A line of the form

```
MAX CV: "max subscript"
```

is inserted after the corresponding job variable definition (or after the identifier definitions if there is no corresponding job variable definition.) If no maximum is specified, only CV(0) may be used. All chain variables are initialized to 1. Chain variables are represented internally as double precision floating point numbers.

Chain variables are assigned values at set nodes, as with job variables. If CV(0) for an open chain has a value other than 0, samples from the arrival time distributions are divided by CV(0) to obtain actual interarrival times. If there are pending source events for an open chain when CV(0) is changed to a value other than 0, those events are rescheduled. The new time until an event is obtained by multiplying the old time until an event by the old value of CV(0) and dividing that result by the new value of CV(0). Setting CV(0) to 0 shuts off all sources for that chain; any pending source events for the chain are deleted from the event list and no new events will be scheduled, even if CV(0) should later become non-zero.

As an example of the use of CV(0) to change arrival rates, let us suppose we want to look at the behavior of our hypothetical computer system as the number of users at the terminals varies during the day. The top part of Figure 8.3 shows users arriving at the terminals and alternating between thinking at the terminals and waiting for command process-

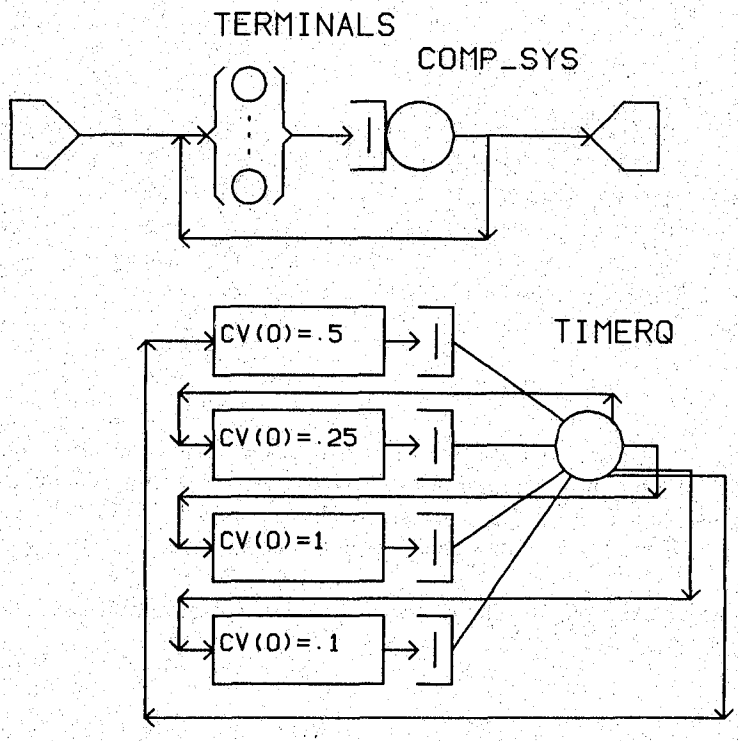


Figure 8.3 - Arrivals Dependent on Time of Day

ing until they are finished and leave. The computer system is represented by a single queue with queue dependent service rates. Queue dependent rates are discussed in Section 15. The rates we will use were obtained by standard approximation techniques assuming partitioned memory with four partitions. For discussion of approximate solutions, see [CHAN78b, SAUE79, SAUE81a, LAVE82]. Let us assume the peak arrival rate of users is from 1 to 5 in the afternoon, that the arrival rate is one half the peak rate from 9 to 12 in the morning, that the arrival rate is one fourth the peak rate during the lunch hour and that the arrival rate is one tenth the peak rate during the night. The bottom part of Figure 8.3 shows the set nodes that will be used for this purpose. There will be a single job alternating between set nodes and service times representing the above periods. Even though this part of the network is disjoint from the remainder as far as the jobs are concerned, we will consider the network to consist of only one chain. (This is the example we referred to in defining chains in Section 10.) The following dialogue file could be used.

```
MODEL:oneday
   METHOD:simulation
   NUMERIC PARAMETERS:peakrate
   NUMERIC IDENTIFIERS:thinktime commands
      THINKTIME:10
      COMMANDS:200
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   QUEUE:comp_sysq
      TYPE:active
      SERVERS:1
      DSPL:ps
```

```
      CLASS LIST:comp_sys
         WORK DEMANDS:1
      SERVER -
         RATES:1.40292 1.98614 2.25576 2.38438
   QUEUE:timerq
      TYPE:fcfs
      CLASS LIST:time9to12 time12to1
         WORK DEMANDS:standard(10800,0) standard(3600,0)
      CLASS LIST:time1to5 time5to9
         WORK DEMANDS:standard(14400,0) standard(57600,0)
   SET NODES:set9to12 set12to1 set1to5 set5to9
   ASSIGNMENT LIST:cv(0)=.5 cv(0)=.25 cv(0)=1 cv(0)=.1
   CHAIN:users
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:1/peakrate
      :s->terminals->comp_sys->sink terminals;1/commands 1-1/commands
      :set9to12->time9to12->set12to1->time12to1->set1to5
      :set1to5->time1to5->set5to9->time5to9->set9to12
   QUEUES FOR QUEUEING TIME DIST:comp_sysq
      VALUES:1 2 3 4 5 6 7 8
   CONFIDENCE INTERVAL METHOD:replications
   INITIAL STATE DEFINITION -
   CHAIN:users
      NODE LIST:set9to12
      INIT POP:1
   CONFIDENCE LEVEL:90
   NUMBER OF REPLICATIONS:25
   REPLIC LIMITS -
      SIMULATED TIME:28800
   LIMIT - CP SECONDS:2800
   TRACE:no
END
```

We are examining transient, not equilibrium, behavior with this model. It would not be appropriate to use the regenerative method or the spectral method to obtain confidence intervals with this model. This is an example of a situation where it is appropriate to use independent replications with a large number of replications. For the period 9 to 5 with a peak rate of one arrival per 100 seconds we get the following:

```
RESQ2 VERSION DATE: SEPTEMBER 4, 1981
MODEL:ONEDAY
PEAKRATE:.01
REPLICATION   1: SIMULATED TIME LIMIT
REPLICATION   2: SIMULATED TIME LIMIT
REPLICATION   3: SIMULATED TIME LIMIT
REPLICATION   4: SIMULATED TIME LIMIT
REPLICATION   5: SIMULATED TIME LIMIT
REPLICATION   6: SIMULATED TIME LIMIT
REPLICATION   7: SIMULATED TIME LIMIT
REPLICATION   8: SIMULATED TIME LIMIT
REPLICATION   9: SIMULATED TIME LIMIT
REPLICATION  10: SIMULATED TIME LIMIT
REPLICATION  11: SIMULATED TIME LIMIT
```

```
REPLICATION  12: SIMULATED TIME LIMIT
REPLICATION  13: SIMULATED TIME LIMIT
REPLICATION  14: SIMULATED TIME LIMIT
REPLICATION  15: SIMULATED TIME LIMIT
REPLICATION  16: SIMULATED TIME LIMIT
REPLICATION  17: SIMULATED TIME LIMIT
REPLICATION  18: SIMULATED TIME LIMIT
REPLICATION  19: SIMULATED TIME LIMIT
REPLICATION  20: SIMULATED TIME LIMIT
REPLICATION  21: SIMULATED TIME LIMIT
REPLICATION  22: SIMULATED TIME LIMIT
REPLICATION  23: SIMULATED TIME LIMIT
REPLICATION  24: SIMULATED TIME LIMIT
REPLICATION  25: SIMULATED TIME LIMIT
NO ERRORS DETECTED DURING SIMULATION.
  SIMULATED TIME PER REPLICATION:      2.8800E+04
                        CPU TIME:           805.65
NUMBER OF EVENTS PER REPLICATION:            72556
         NUMBER OF REPLICATIONS:                25

WHAT:utbo


ELEMENT         UTILIZATION
TERMINALSQ      0.00000(0.00000,0.00000)
COMP_SYSQ       0.64892(0.63425,0.66360) 2.9%
TIMERQ          1.00000

WHAT:tpbo(terminalsq,comp_sysq,s,sink)


ELEMENT         THROUGHPUT
TERMINALSQ      1.25608(1.21623,1.29592) 6.3%
COMP_SYSQ       1.25589(1.21606,1.29572) 6.3%
S               7.2278E-03
SINK            6.3986E-03

WHAT:qlbo(terminalsq,comp_sysq)


ELEMENT         MEAN QUEUE LENGTH
TERMINALSQ      12.55767(12.16163,12.95370) 6.3%
COMP_SYSQ       2.16919(1.87909,2.45929) 26.7%

WHAT:qtbo(comp_sysq)


ELEMENT         MEAN QUEUEING TIME
COMP_SYSQ       1.69493(1.53678,1.85308) 18.7%

WHAT:qtdbo
```

```
ELEMENT          QUEUEING TIME DISTRIBUTION
COMP_SYSQ        1.00E+00:0.52675(0.51026,0.54324) 3.3%
                 2.00E+00:0.74146(0.72224,0.76068) 3.8%
                 3.00E+00:0.84435(0.82639,0.86230) 3.6%
                 4.00E+00:0.89907(0.88329,0.91486) 3.2%
                 5.00E+00:0.93120(0.91767,0.94473) 2.7%
                 6.00E+00:0.95121(0.93976,0.96267) 2.3%
                 7.00E+00:0.96450(0.95484,0.97416) 1.9%
                 8.00E+00:0.97365(0.96555,0.98175) 1.6%

WHAT:pobo


ELEMENT          OPEN CHAIN POPULATION
USERS            15.72686(15.06218,16.39154) 8.5%

WHAT:rtmbo


ELEMENT          OPEN CHAIN RESPONSE TIME
USERS            2455.07861(2374.40869,2535.74829) 6.6%

WHAT:
PEAKRATE:
```

Note that the chain population and response time values include the timing job. If we want the mean time users spend in the system we should use Little's Rule, i.e., 14.72686/.0063986 = 2302 seconds.

We will see another example of determining source rates with CV(0) in Section 8.3.

8.3. Global Variables

Global variables provide for storage of values which may change during simulation. (Global variables can be used for values which do not vary during simulation, but it will be more efficient and flexible to use numeric identifiers for these values.) Global variables are used with set nodes and numerical expressions in the same manner as numeric variables are used in programming languages. "Global" is used in contrast to job and chain variables, which are local to jobs and chains, respectively. Global variables may be defined to be local to submodels. (See Section 13 and Sections 3 and 10 of the Users Guide). Global variables may be defined as scalars and as one and two dimensional arrays. Global variables are defined and given initial values in the same manner as numeric and distribution identifiers, following the definition of any of those. For example,

```
GLOBAL VARIABLES:a b(3) c(3;2)
   A:3.1
   B:0
   C:14.1 7 13
```

All of the values for an array are defined on a single line (If necessary, multiple physical lines may be concatenated to form a single logical line. See Section 2 of the Users Guide.) If fewer values are given then the number of elements in an array, the last value given is used for the remaining elements. In the example above, all three elements of b are initially zero. Two

dimensional arrays are stored by rows, so in the example above c(1;1) is initially 14.1; c(1;2) is initially 7 and the remaining elements are initially 13.

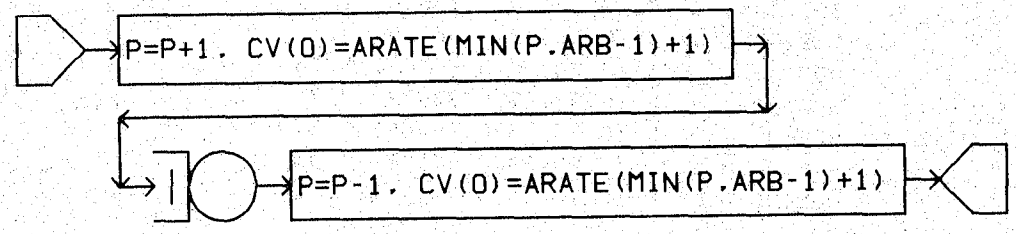


Figure 8.4 - Population Dependent Arrivals

Suppose we wish the arrival rate in an open network to be a function of the network population. We wish to specify the rates up to some population and have the last rate apply to larger populations. We can accomplish this with a global variable to keep track of the population, p, and chain variable scaling of the arrival times. See Figure 8.4. The rates are stored in numeric identifier arate, which has upper bound arb. Since RESQ arrays begin with subscript 1, we use arate(1) for population 0, arate(2) for population 1 and so on with arate(arb) used for population arb−1 and larger populations. The min (minimum) function has exactly two arguments in RESQ. We could use the following dialogue file:

```
MODEL:pda /*Population Dependent Arrivals*/
   METHOD:simulation
   NUMERIC PARAMETERS:atime stime
   NUMERIC IDENTIFIERS:arb arate(arb)
      ARB:4
      ARATE:1 .8 .6 .4
   GLOBAL VARIABLES:p
      P:0
   QUEUE:q
      TYPE:fcfs
      CLASS LIST:c
         SERVICE TIMES:stime
   SET NODES:bef
   ASSIGNMENT LIST:  p=p+1 cv(0)=arate(min(p,arb-1)+1)
   SET NODES:aft
   ASSIGNMENT LIST:  p=p-1 cv(0)=arate(min(p,arb-1)+1)
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:atime
      :s->bef->c->aft->sink
   QUEUES FOR QUEUEING TIME DIST:q
      VALUES:10 20 30 40 50
   QUEUES FOR QUEUE LENGTH DIST:q
      MAX VALUE:10
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:q
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
```

```
      QUEUES FOR DEPARTURE COUNTS:q
         DEPARTURES:10000
   LIMIT - CP SECONDS:50
   TRACE:no
END
```

We can get the following results from EVAL:

```
RESQ2 VERSION DATE: OCTOBER 9, 1981
MODEL:PDA
ATIME:10
STIME:10
WARNING -- MODEL MAY NOT BE TRULY REGENERATIVE
           BECAUSE OF USE OF GLOBAL VARIABLES
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      1.3733E+05
                        CPU TIME:            24.60
                NUMBER OF EVENTS:            20000
                NUMBER OF CYCLES:             3889

WHAT:utbo

ELEMENT         UTILIZATION
Q               0.71541(0.70552,0.72530) 2.0%

WHAT:tpbo(q)

ELEMENT         THROUGHPUT
Q               0.07282(0.07192,0.07372) 2.5%

WHAT:qlbo

ELEMENT         MEAN QUEUE LENGTH
Q               1.48025(1.43406,1.52644) 6.2%

WHAT:qtbo

ELEMENT         MEAN QUEUEING TIME
Q               20.32849(19.66095,20.99605) 6.6%

WHAT:qldbo

ELEMENT         QUEUE LENGTH DISTRIBUTION
Q                 0:0.28459(0.27470,0.29448) 2.0%
                  1:0.28161(0.27473,0.28849) 1.4%
                  2:0.22409(0.21753,0.23065) 1.3%
                  3:0.13062(0.12416,0.13707) 1.3%
```

```
                    4:0.05129(0.04658,0.05601) 0.9%
                    5:0.01830(0.01488,0.02172) 0.7%
                    6:5.6111E-03(4.1301E-03,7.0921E-03) 0.3%
                    7:3.0926E-03(1.4506E-03,4.7346E-03) 0.3%
                    8:5.5150E-04(2.0370E-04,8.9931E-04) 0.1%
                    9:2.4575E-04(-7.3931E-05,5.6542E-04) 0.1%

WHAT:mxql

ELEMENT             MAXIMUM QUEUE LENGTH
Q                   9

WHAT:qtdbo

ELEMENT             QUEUEING TIME DISTRIBUTION
Q                   1.00E+01:0.35100(0.33882,0.36318) 2.4%
                    2.00E+01:0.59710(0.58287,0.61133) 2.8%
                    3.00E+01:0.77060(0.75709,0.78411) 2.7%
                    4.00E+01:0.87060(0.85949,0.88171) 2.2%
                    5.00E+01:0.93380(0.92553,0.94207) 1.7%

WHAT:
CONTINUE RUN:no

ATIME:
```

A model which uses global variables will not be truly regenerative unless the global variables have the same values each time the model is in the "regeneration" state. In this model the global variable p will always have a zero value in the regeneration state. It is required that CV(0) for open chains has value 1 in the regeneration state (Section 5.2).

This example is slightly contrived in that the use of global variables is not strictly necessary. The ql or tq status functions (Appendix 3 of the Users Guide) could be used to avoid the global variable p. That approach could be extended to general networks, but the above approach is likely to be more efficient in general networks. Other examples with global variables are found in Sections 10 and 11 and in Appendix 3 of the Users Guide.

There are a number of global variable identifiers which have special meaning to Aplomb and should not be used for other purposes. These are "clock" and several identifiers including the word "trace" (see Appendix 2 of the Users Guide).

9. ROUTING

All of our examples so far have assumed that routing decisions are made according to probabilities. In this case the format of a routing transition is

from_node -> to_node_1 to_node_2 ... to_node_N ; p_1 p_2 ... p_N.

The semi-colon (";") and probabilities are optional if the probabilities are all equal to the inverse of the number of "to nodes" (e.g., $1/N$). It is entirely permissible to split the above transition into several, e.g.,

from_node -> to_node_1; p_1
from_node -> to_node_2; p_2
...
from_node -> to_node_N; p_N

or

from_node -> to_node_1; p_1
from_node -> to_node_2 to_node_3; p_2 p_3
...
from_node -> to_node_N; p_N

etc. Routing decision may also be made with predicates, i.e., expressions which represent Boolean (true or false) values. In the above examples any or all of the probabilities could be replaced by a predicate of the form

if("Boolean expression").

Assuming that all the probabilities are replaced by predicates then the destination for the "from node" would be chosen by evaluating the predicates in the order given in the dialogue and picking the first destination where the predicate had a true value. (The remaining predicates would not be evaluated.) If none of the predicates had a true value, then an error condition would exist and the simulation would be terminated. (Predicates are not allowed with numerical solution in RESQ.) (Mixtures of probabilities and predicates for the routing from a given node are discussed in Section 9 of the Users Guide.)

Typically, the Boolean expression will consist of one or more relational expressions of the form

"numeric expression" "relational operator" "numeric expression"

where the numeric expressions may be any legal RESQ numeric expression with a scalar value. The relational operator may be any of the following: "=" (equal), "¬=" (not equal), "<" (less than), "<=" (less than or equal), ">" (greater than) and ">=" (greater than or equal). The Boolean operators "not", "and" and "or" may be used with the usual meaning, in that order of precedence. For example,

if(not jv(3)<10 and jv(2)>3 or p=0)

would be true if p had the value 0 or if jv(3) was greater than or equal to 10 and jv(2) was greater than 3. To get other orders of precedence among the operators, we must enclose Boolean expressions in parentheses preceded by "if". For example,

```
if(not if(jv(3)<10 and if(jv(2)>3 or p=0)))
```

would have exactly the reverse order of precedence of the operators as the previous example.

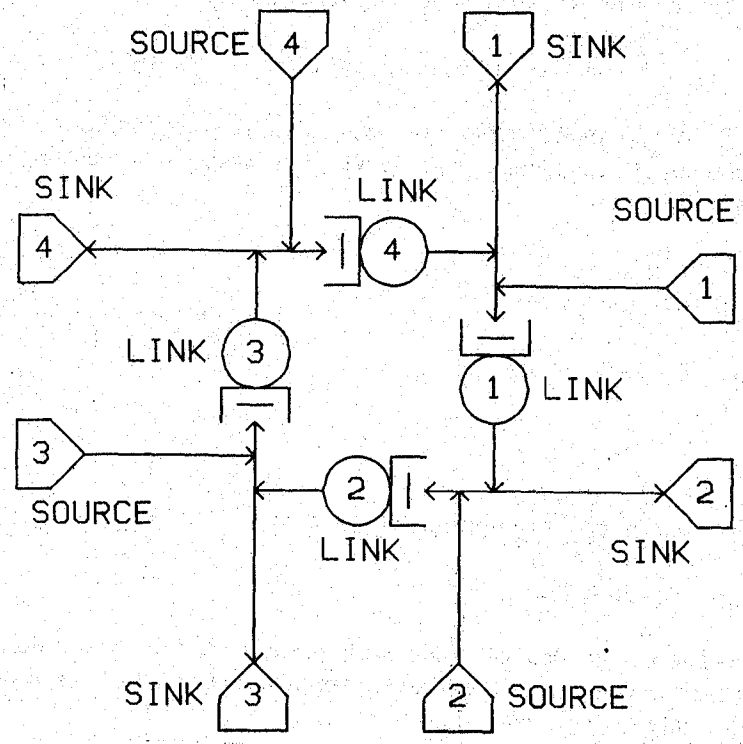


Figure 9.1 - Routing Example

There are many situations where routing predicates are necessary for describing a model. We now describe a case where predicates may be used to simplify routing description in a model. Consider the network of Figure 9.1. Let us suppose that this is a model of a communication network (see Section 8.1). The destination for a message is determined upon arrival of the message. (We will assume each destination is equally likely to be picked.) Since a message has a specified destination, the routing decision after each queue (communication link), whether to proceed to the next queue (message has not reached its destination) or to go to the sink (message has reached its destination), must be deterministic. There are two ways we can represent this: either we have a class at each queue for each possible destination of jobs leaving the queue or we use routing predicates. Let us examine these options in turn. The following dialogue file could be used for the first option:

```
MODEL:loop
   METHOD:numerical
   NUMERIC IDENTIFIERS:mean_atime mean_stime
      MEAN_ATIME:.1
      MEAN_STIME:.15
   QUEUE:q1
      TYPE:fcfs
      CLASS LIST:c1d2 c1d3 c1d4
         SERVICE TIMES:mean_stime
   QUEUE:q2
      TYPE:fcfs
      CLASS LIST:c2d3 c2d4 c2d1
```

```
        SERVICE TIMES:mean_stime
   QUEUE:q3
      TYPE:fcfs
      CLASS LIST:c3d4 c3d1 c3d2
         SERVICE TIMES:mean_stime
   QUEUE:q4
      TYPE:fcfs
      CLASS LIST:c4d1 c4d2 c4d3
         SERVICE TIMES:mean_stime
   CHAIN:c
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:mean_atime
      :s->c1d2 c1d3 c1d4;1/12 1/12 1/12
      :c1d2->sink
      :c1d3->c2d3
      :c1d4->c2d4
      :s->c2d3 c2d4 c2d1;1/12 1/12 1/12
      :c2d3->sink
      :c2d4->c3d4
      :c2d1->c3d1
      :s->c3d4 c3d1 c3d2;1/12 1/12 1/12
      :c3d4->sink
      :c3d1->c4d1
      :c3d2->c4d2
      :s->c4d1 c4d2 c4d3;1/12 1/12 1/12
      :c4d1->sink
      :c4d2->c1d2
      :c4d3->c1d3
END
```

Here we use the name c*id*j for queue *i* jobs with destination *j*. Though the diagram shows four sources and four sinks, we use only one of each. RESQ allows only one source per chain with numerical solution. Since the arrival times for the sources are exponential, i.e., the arrival processes are Poisson, we can combine the sources into a single source with arrival rate equal to the sum of the arrival rates of the individual sources. Or equivalently, we make the mean arrival time the reciprocal of the sum of the reciprocals of the individual arrival times. The arrival probabilities for jobs leaving the single source are the arrival rates of the individual sources normalized so that the probabilities sum to one. In constructing the above dialogue file we assumed equal arrival rates for each of the sources. Thus the probability a job starts at a given location is 1/4. RESQ only allows one sink for the entire network. EVAL gives us the following results:

```
RESQ2 VERSION DATE: OCTOBER 9, 1981
MODEL:LOOP
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:ut


ELEMENT         UTILIZATION
Q1              0.75000
Q2              0.75000
Q3              0.75000
```

```
Q4              0.75000

WHAT:qt


ELEMENT         MEAN QUEUEING TIME
Q1              0.60000
Q2              0.60000
Q3              0.60000
Q4              0.60000

WHAT:po


ELEMENT         OPEN CHAIN POPULATION
C               11.99999

WHAT:rtm


ELEMENT         OPEN CHAIN RESPONSE TIME
C               1.20000

WHAT:
```

By the results of Wong [WONG78], we know that the response time distribution in this network has a distribution representable by the method of exponential stages. The distribution can be represented by the branching Erlang form with three stages, $m_1 = m_2 = m_3 = .6$, $p_1 = 1/3$ and $p_2 = .5$. From equation (A3.2) in the Users Guide we have $C = .8165$ and standard deviation .9798. (This can be seen from Wong's results and the fact that the response time will consist of one, two or three queueing times with equal probability.)

We can represent this same model using job variables to save the destination of a job and predicates which test the value of the job variable. Let us put the destination in JV(0). Let us also store the service time in JV(1); this will give another example of the effects of Kleinrock's independence assumption (see Section 11). Finally, let us measure the response times with a passive queue. We could use the following dialogue file:

```
MODEL:loop
   METHOD:simulation
   NUMERIC IDENTIFIERS:mean_atime mean_stime
      MEAN_ATIME:.1
      MEAN_STIME:.15
   NUMERIC IDENTIFIERS:msg_dest msg_stime
      MSG_DEST:0 /*JV to be used*/
      MSG_STIME:1 /*JV to be used*/
   MAX JV:1
   QUEUE:rtq
      TYPE:passive
      TOKENS:2147483647
      DSPL:fcfs
      ALLOCATE NODE LIST:beginrt
         AMOUNTS:1
   QUEUE:q1
```

```
      TYPE:fcfs
      CLASS LIST:c1
         WORK DEMANDS:standard(jv(msg_stime),0)
   QUEUE:q2
      TYPE:fcfs
      CLASS LIST:c2
         WORK DEMANDS:standard(jv(msg_stime),0)
   QUEUE:q3
      TYPE:fcfs
      CLASS LIST:c3
         WORK DEMANDS:standard(jv(msg_stime),0)
   QUEUE:q4
      TYPE:fcfs
      CLASS LIST:c4
         WORK DEMANDS:standard(jv(msg_stime),0)
   SET NODES:set_stime
   ASSIGNMENT LIST:jv(msg_stime)=standard(mean_stime,1)
   SET NODES:set_dest1
   ASSIGNMENT LIST:jv(msg_dest)=discrete(2,1/3; 3,1/3; 4,1/3)
   SET NODES:set_dest2
   ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 3,1/3; 4,1/3)
   SET NODES:set_dest3
   ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 2,1/3; 4,1/3)
   SET NODES:set_dest4
   ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 2,1/3; 3,1/3)
   CHAIN:c
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:mean_atime
      :s->beginrt->set_stime->set_dest1 set_dest2 set_dest3 set_dest4
      :set_dest1 set_dest2 set_dest3 set_dest4->c1 c2 c3 c4
      :c1->sink c2;if(jv(msg_dest)=2) if(t)
      :c2->sink c3;if(jv(msg_dest)=3) if(t)
      :c3->sink c4;if(jv(msg_dest)=4) if(t)
      :c4->sink c1;if(jv(msg_dest)=1) if(t)
   QUEUES FOR QUEUEING TIME DIST:rtq
      VALUES:.6 1.2 1.8 2.4 3.0 3.6
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:rtq
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:rtq
         DEPARTURES:10000
   LIMIT - CP SECONDS:250
   TRACE:no
END
```

The line

```
      :dest1 dest2 dest3 dest4->c1 c2 c3 c4
```

is equivalent to the four lines

```
:dest1->c1
:dest2->c2
:dest3->c3
:dest4->c4
```

Such a parallel grouping of transitions is allowed when there is only one "to node" for each "from node". The keyword "t" in the predicate "if(t)" represents the constant "true". (The keyword "f" is available to represent the constant "false".)

We said that we were simplifying the routing, yet the above dialogue is much longer! However, the increase in length is due to the use of simulation, to the job variable scaling to avoid the independence assumption and to the passive queue for response times. There are now four classes instead of twelve (though there are four new set nodes) and there are fewer routing transitions. Though the difference is not dramatic, if there were more links (queues) then the difference would be more pronounced. We are *not* saying that this second approach with job variables and predicates is preferable in general; the RESQ user should consider both approaches in developing a model.

We get the following from EVAL:

```
RESQ2 VERSION DATE: OCTOBER 9, 1981
MODEL:LOOP
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      4126.73047
                        CPU TIME:         121.50
                 NUMBER OF EVENTS:         123341
                 NUMBER OF CYCLES:            367

WHAT:utbo


ELEMENT          UTILIZATION
RTQ              5.3685E-09(5.1027E-09,5.6343E-09) 0.0%
Q1               0.73852(0.72578,0.75126) 2.5%
Q2               0.75653(0.74188,0.77118) 2.9%
Q3               0.75569(0.74239,0.76900) 2.7%
Q4               0.74544(0.73237,0.75851) 2.6%

WHAT:tpbo(rtq)


ELEMENT          THROUGHPUT
RTQ              9.95679(9.87664,10.03694) 1.6%

WHAT:qlbo(rtq)
```

```
ELEMENT          MEAN QUEUE LENGTH
RTQ              11.52880(10.95804,12.09957) 9.9%

WHAT:qtbo(rtq)


ELEMENT          MEAN QUEUEING TIME
RTQ              1.15788(1.10514,1.21063) 9.1%

WHAT;sdqt(rtq)


ELEMENT          STANDARD DEVIATION OF QUEUEING TIME
RTQ              0.98273

WHAT:qtdbo


ELEMENT          QUEUEING TIME DISTRIBUTION
RTQ              6.00E-01:0.35635(0.34292,0.36978) 2.7%
                 1.20E+00:0.61885(0.59957,0.63814) 3.9%
                 1.80E+00:0.78792(0.76920,0.80665) 3.7%
                 2.40E+00:0.88318(0.86784,0.89852) 3.1%
                 3.00E+00:0.94042(0.92917,0.95167) 2.2%
                 3.60E+00:0.97121(0.96404,0.97838) 1.4%

WHAT:
CONTINUE RUN:/*Continue run:*/ yes

EXTRA SAMPLING PERIODS:/*Extra periods:*/ 1

SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:     5147.03516
                        CPU TIME:         152.82
                 NUMBER OF EVENTS:         154129
                 NUMBER OF CYCLES:            429

WHAT:qtbo(rtq)


ELEMENT          MEAN QUEUEING TIME
RTQ              1.15198(1.10696,1.19700) 7.8%

WHAT:nd(rtq)


ELEMENT          NUMBER OF DEPARTURES
RTQ              51267
```

```
WHAT:qtdbo


ELEMENT         QUEUEING TIME DISTRIBUTION
RTQ             6.00E-01:0.35526(0.34385,0.36666) 2.3%
                1.20E+00:0.61923(0.60268,0.63578) 3.3%
                1.80E+00:0.79008(0.77403,0.80613) 3.2%
                2.40E+00:0.88712(0.87407,0.90017) 2.6%
                3.00E+00:0.94308(0.93348,0.95268) 1.9%
                3.60E+00:0.97261(0.96650,0.97873) 1.2%

WHAT:
CONTINUE RUN:/*Continue run:*/ no
```

This is an example where Kleinrock's independence assumption seems more appropriate; the mean response time estimates are essentially the same, especially when one considers that the observed throughput is lower than the specified arrival time. The standard deviation of response time in the simulation is very close to the value from Wong's results.

10. PASSIVE QUEUES

Our examples so far have used only part of the generality of the passive queue. We have only used allocate nodes and release nodes, and usually there has been a single pair of allocate and release nodes for a given queue. However, a passive queue may have an arbitrary number of nodes, as long as there is at least one allocate node, and release nodes need not be paired with allocate nodes if they are present at all. Many of our models using a passive queue for measuring response times have no release nodes at all. Note that sinks, and to a lesser extent, fusion nodes (Section 11), can be used to release tokens.

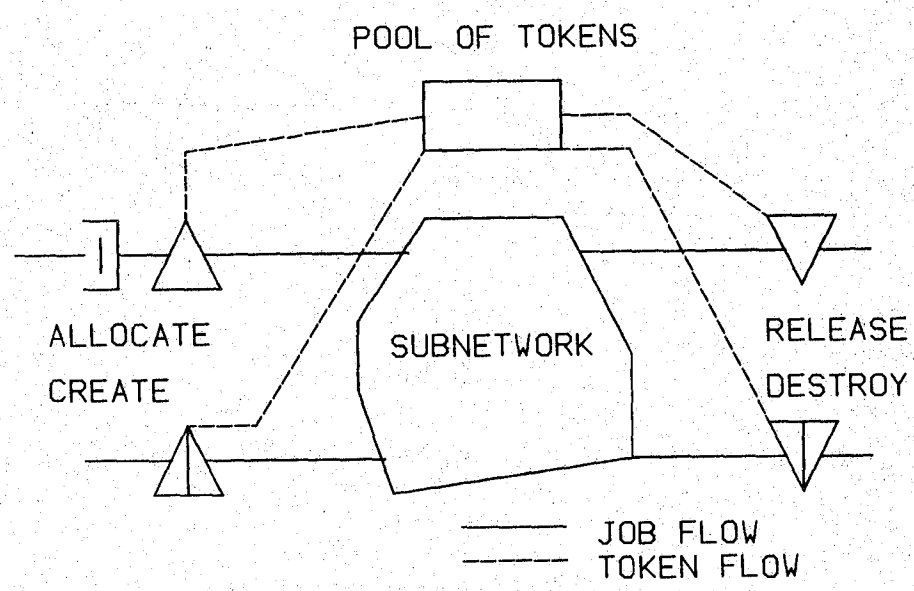


Figure 10.1 - Passive Queue

There are five other kinds of nodes which may be present in passive queues, AND allocate nodes, OR allocate nodes, transfer nodes, destroy nodes and create nodes. AND allocate, OR allocate and transfer nodes are discussed in Section 5 of the Users Guide. We will focus on create and destroy nodes in this section.

A create node is used to add tokens to the pool of tokens of a passive queue. The number of tokens created is specified analogously to specification of the number of tokens requested at an allocate node. A create node behaves in the same manner whether or not a job holds tokens of the queue, i.e., a job need not hold tokens to create tokens. A create node has no effect on the job going through it; the job passes through without delay. The effect on jobs waiting for tokens, if any are waiting, is the same as if the tokens became available through another job releasing tokens. A destroy node is similar to a release node, in that a job gives up all tokens it holds of the queue, if it holds any, but the tokens are destroyed rather than made available to other jobs.

As an example of the use of create and destroy nodes, let us consider printer spooling in a simple computer system model. A potential problem with the models we have used so far is that they ignore spooling of disk files to slower speed input/output devices. Let us assume that there is a 300 line per minute printer supported by the computer system and that there are two tasks constantly present which handle the spooling. One task fills buffers from the disk for the printer and the other dumps the buffers to the printer. There are two buffers for the printer and each buffer contains 30 lines. Thus the transfer time for one buffer is 6 seconds (30/(300/60)).

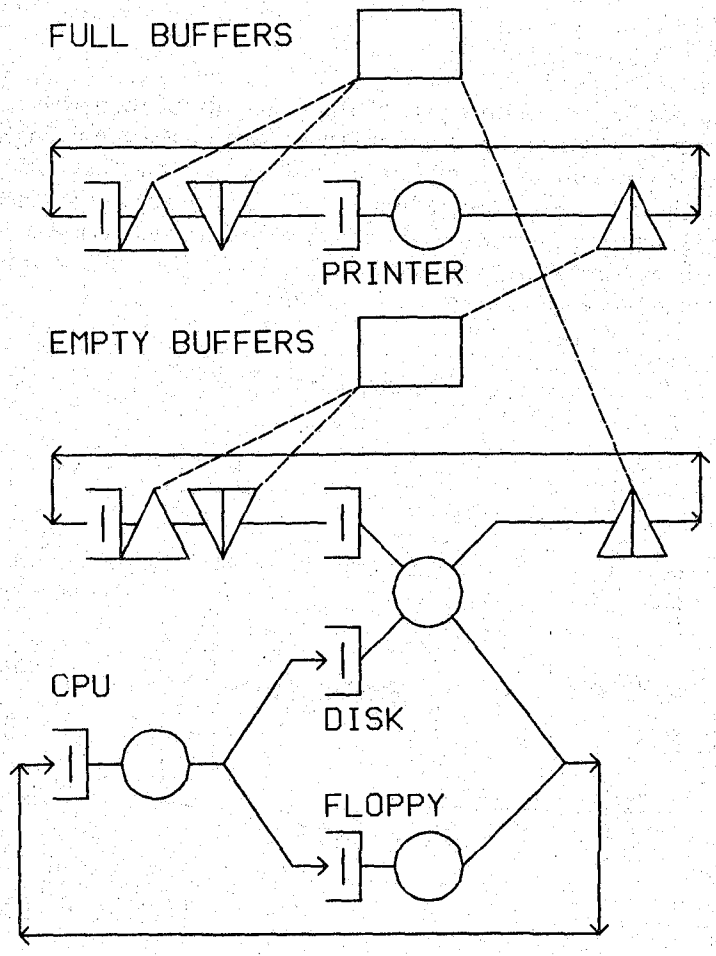


Figure 10.2 - Printer Spooling

To represent the printer spooling we have two chains, one for each task, and two passive queues, one for full buffers and one for empty buffers. The passive queues will be used, in part, to represent communication between the tasks, corresponding to the use of semaphores and similar process communication primitives in operating systems. See Figure 10.2. The number of tokens of each pool will fluctuate between zero and two, because of create and destroy nodes, and the total number of tokens will usually be less than two. The task which empties the buffers acquires a token representing a full buffer, destroys it, transfers the buffer contents to the printer and creates a token of the pool representing empty buffers. Similarly the task which fills the buffers acquires an "empty buffer" token, destroys it, transfers from the disk to the buffer and creates a token of the "full buffer" pool. The buffer emptying task waits at the full buffer allocate node when no full buffers are available, and the buffer filling task waits at the empty buffer allocate node when no empty buffers are available.

The following dialogue file could be used for this model.

```
MODEL:csmwsp
   METHOD:simulation
   NUMERIC IDENTIFIERS:floppytime disktime cputime dmp
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
      DMP:4
   NUMERIC IDENTIFIERS:buffers initfulbuf
```

```
   BUFFERS:2
   INITFULBUF:2
NUMERIC IDENTIFIERS:lpm /*lines/minute*/ lpb /*lines/buffer*/
   LPM:300
   LPB:30
QUEUE:floppyq
   TYPE:fcfs
   CLASS LIST:floppy
      SERVICE TIMES:floppytime
QUEUE:diskq
   TYPE:fcfs
   CLASS LIST:disk diskspool
      SERVICE TIMES:disktime
QUEUE:cpuq
   TYPE:ps
   CLASS LIST:cpu
      SERVICE TIMES:cputime
QUEUE:printerq
   TYPE:fcfs
   CLASS LIST:printer
      SERVICE TIMES:standard(lpb/(lpm/60),0)
QUEUE:fullbuffer
   TYPE:passive
   TOKENS:initfulbuf-1
   DSPL:fcfs
   ALLOCATE NODE LIST:getfullbuf
      NUMBERS OF TOKENS TO ALLOCATE:1
   DESTROY NODE LIST:destfulbuf
   CREATE NODE LIST:genfullbuf
      NUMBERS OF TOKENS TO CREATE:1
QUEUE:empbuffer
   TYPE:passive
   TOKENS:buffers-initfulbuf
   DSPL:fcfs
   ALLOCATE NODE LIST:getempbuf
      NUMBERS OF TOKENS TO ALLOCATE:1
   DESTROY NODE LIST:destempbuf
   CREATE NODE LIST:genempbuf
      NUMBERS OF TOKENS TO CREATE:1
CHAIN:csm
   TYPE:closed
   POPULATION:dmp
   :cpu->disk floppy;.9 .1
   :disk floppy->cpu
CHAIN:emptying
   TYPE:closed
   POPULATION:1
   :getfullbuf->destfulbuf->printer->genempbuf->getfullbuf
CHAIN:filling
   TYPE:closed
   POPULATION:1
   :getempbuf->destempbuf->diskspool->genfullbuf->getempbuf
CONFIDENCE INTERVAL METHOD:regenerative
REGENERATION STATE DEFINITION -
```

```
   CHAIN:csm
      NODE LIST:cpu
      REGEN POP:dmp
      INIT POP:dmp
   CHAIN:emptying
      NODE LIST:printer
      REGEN POP:1
      INIT POP:1
   CHAIN:filling
      NODE LIST:getempbuf
      REGEN POP:1
      INIT POP:1
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:floppyq diskq printerq
         MEASURES:ut ut ut
         ALLOWED WIDTHS:10 10 10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:cpuq
         DEPARTURES:10000
   LIMIT - CP SECONDS:250
   TRACE:no
END
```

We could get the following results from EVAL.

```
RESQ2 VERSION DATE: OCTOBER 16, 1981
MODEL:CSMWSP
WARNING -- SOME PASSIVE QUEUE QT PROCESSES MAY
           NOT BE TRULY REGENERATIVE BECAUSE OF
           QUEUEING TIMES IN PROGRESS
WARNING -- MODEL MAY NOT BE TRULY REGENERATIVE
           BECAUSE OF NON-ZERO POPULATION AT CLASS
           WITH DIST. OTHER THAN BRANCHING ERLANG
SAMPLING PERIOD END: CPUQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: CPUQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      1055.25220
                        CPU TIME:           33.36
                 NUMBER OF EVENTS:          40350
                 NUMBER OF CYCLES:           8317

WHAT:utbo(*)


ELEMENT         UTILIZATION
FULLBUFFER      0.00000
EMPBUFFER       0.00000
FLOPPYQ         0.41176(0.36190,0.46162) 10.0%
DISKQ           0.32471(0.31889,0.33053) 1.2%
 DISK            0.32191(0.31611,0.32772) 1.2%
 DISKSPOOL       2.7956E-03(-5.6492E-04,6.1562E-03) 0.7%
CPUQ            0.95834(0.95279,0.96389) 1.1%
```

```
PRINTERQ        1.00000(0.99992,1.00008) 0.0%

WHAT:qlbo(*)


ELEMENT         MEAN QUEUE LENGTH
FULLBUFFER      0.00000
EMPBUFFER       0.99528(0.98976,1.00080) 1.1%
FLOPPYQ         0.63072(0.51490,0.74653) 36.7%
DISKQ           0.46227(0.44977,0.47476) 5.4%
 DISK            0.45754(0.44516,0.46993) 5.4%
 DISKSPOOL       4.7232E-03(-7.9525E-04,1.0242E-02) 233.7%
CPUQ            2.91174(2.86889,2.95459) 2.9%
PRINTERQ        1.00000(0.99992,1.00008) 0.0%

WHAT:tpbo(*)


ELEMENT         THROUGHPUT
FULLBUFFER      0.16584(0.03030,0.30138) 163.5%
EMPBUFFER       0.16584(0.03030,0.30138) 163.5%
FLOPPYQ         1.93792(1.77106,2.10479) 17.2%
DISKQ           17.18071(16.95303,17.40837) 2.7%
 DISK            17.01488(16.78821,17.24153) 2.7%
 DISKSPOOL       0.16584(0.03030,0.30138) 163.5%
CPUQ            18.95280(18.73048,19.17511) 2.3%
PRINTERQ        0.16584(0.03030,0.30138) 163.5%

WHAT:
CONTINUE RUN:yes

EXTRA SAMPLING PERIODS:1

SAMPLING PERIOD END: CPUQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: CPUQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: CPUQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                SIMULATED TIME:      1585.19995
                      CPU TIME:           50.02
               NUMBER OF EVENTS:          60544
               NUMBER OF CYCLES:          12421

WHAT:utbo(*)


ELEMENT         UTILIZATION
FULLBUFFER      0.00000
EMPBUFFER       0.00000
FLOPPYQ         0.41248(0.37124,0.45372) 8.2%
DISKQ           0.32650(0.32175,0.33124) 0.9%
 DISK            0.32351(0.31877,0.32824) 0.9%
 DISKSPOOL       2.9893E-03(8.4241E-05,5.8944E-03) 0.6%
CPUQ            0.95617(0.95123,0.96111) 1.0%
```

```
PRINTERQ        1.00000(0.99993,1.00007) 0.0%

WHAT:
CONTINUE RUN:no
```

In this dialogue we see two warning messages about the regeneration state. The second one is consistent with our previous discussion in Sections 5 and 8.3. Because we have a job at the printer class in the regeneration state and because the service times at that class are not represented by the branching Erlang distribution, the state we have chosen is not truly a regeneration state. In fact, this model is not truly regenerative, i.e., there is no state which is truly a regeneration state. However, RESQ allows us to proceed as if the model were regenerative. (We are not especially interested in queueing times with this example, so we use this approximation of the regenerative method to obtain confidence intervals rather than using the spectral method. The method of independent replications would also be a reasonable choice.)

The first warning message has a fairly subtle explanation. Nearly all of our discussion of regeneration states has implicitly been restricted to the queue length processes underlying the model and the performance measures obtainable from the queue length process. Most of the performance measures we have considered, including mean queueing time, are obtainable from the queue length process. The warning message and the following discussion do not apply to these measures. (Since we are not focusing on queueing times in this example, we can comfortably ignore the warning.)

In defining a regeneration state for a queueing time process, one must be much more careful than in defining a regeneration state for a queue length process. For a detailed discussion, see Iglehart and Shedler [IGLE80]. If we are to be rigorous, we must not allow queueing times in progress in our regeneration state if we wish to have defensible confidence intervals for queueing time distribution points or standard deviations. This is one of the reasons we used the state with all jobs at the terminals for model csmwm. With a few exceptions, we cannot have queueing times in progress and have those queueing times truly regenerate. Aplomb gives the warning message whenever there are passive queues with jobs (job copies) at allocate nodes in the specified regeneration state. (A similar warning would usually apply to active queues as well, but there is no such warning issued because (1) a regeneration state must have some jobs at classes, so the warning would always appear, and (2) active queues are not used to measure response times the way passive queues are.) Though it is theoretically possible in some circumstances to have jobs (job copies) at allocate nodes in the specified regeneration state and obtain meaningful confidence intervals, this will usually not be practical.

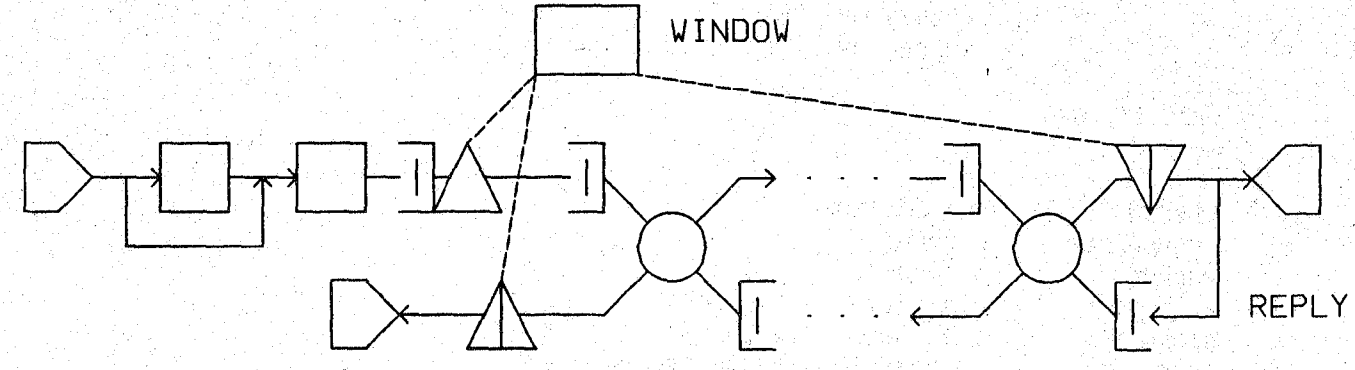


Figure 10.3 - Window Flow Control

Create and destroy nodes are also very important in representing communication network protocols. We consider an example of a window flow control mechanism sometimes referred to as "pacing." The essence of the pacing mechanism is that there is a limit, called the "window," to the number of messages which may be sent from one point in the network to another before the recipient says that more messages may be sent. Usually the first message of the window is marked to indicate to the recipient that a reply should be sent back to the originator. Upon receipt of the reply, the originator may then send another window of messages. Figure 10.3 depicts a pacing mechanism added to model fourlink of Section 13. The passive queue initially has the number of tokens equal to the window size. Each message allocates a token before it can proceed. When it gets to the destination, it destroys the token. If the message was the first of a window, it turns around and goes back to the create node at the origin where it creates a new window of tokens. These may be allocated to any waiting messages. A global variable, wcount, is used to count the arriving messages modulo the window size. If an arriving message finds wcount to be zero, that indicates that it is the first message of the window, and that fact is recorded by setting its JV(1) to one. The following dialogue file could be used.

```
MODEL:pacing
   METHOD:simulation
   NUMERIC PARAMETERS:windowsize
   NUMERIC IDENTIFIERS:msg_stime p_reply
      MSG_STIME:0 /*JV to be used*/
      P_REPLY:1 /*JV to be used*/
   GLOBAL VARIABLE IDENTIFIERS:wcount
      WCOUNT:0
   MAX JV:1
   QUEUE:rtq
      TYPE:passive
      TOKENS:2147483647
      DSPL:fcfs
      ALLOCATE NODE LIST:beginrt
         NUMBERS OF TOKENS TO ALLOCATE:1
      RELEASE NODE LIST:endrt
   QUEUE:windowq
      TYPE:passive
      TOKENS:windowsize
      DSPL:fcfs
      ALLOCATE NODE LIST:getwindow
         NUMBERS OF TOKENS TO ALLOCATE:1
      DESTROY NODE LIST:dropwindow
      CREATE NODE LIST:newwindow
         NUMBERS OF TOKENS TO CREATE:windowsize
   QUEUE:q1
      TYPE:prty
      CLASS LIST:c1r c1l
         SERVICE TIMES:standard(jv(msg_stime),0) standard(.01,0)
         PRIORITIES:2 1
   QUEUE:q2
      TYPE:prty
      CLASS LIST:c2r c2l
         SERVICE TIMES:standard(jv(msg_stime),0) standard(.01,0)
         PRIORITIES:2 1
   QUEUE:q3
      TYPE:prty
```

```
      CLASS LIST:c3r c31
         SERVICE TIMES:standard(jv(msg_stime),0) standard(.01,0)
         PRIORITIES:2 1
   QUEUE:q4
      TYPE:prty
      CLASS LIST:c4r c41
         SERVICE TIMES:standard(jv(msg_stime),0) standard(.01,0)
         PRIORITIES:2 1
   SET NODES:set_stime
   ASSIGNMENT LIST:jv(msg_stime)=standard(.125,1)
   SET NODES:inccount
   ASSIGNMENT LIST:wcount=(wcount+1) mod windowsize
   SET NODES:setreply
   ASSIGNMENT LIST:jv(p_reply)=1
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:.25
      :s->beginrt->set_stime->setreply inccount;if(wcount=0) if(t)
      :setreply->inccount->getwindow->c1r
      :c1r->c2r->c3r->c4r->dropwindow->endrt
      :endrt->sink c41;if(jv(p_reply)=0) if(t)
      :c41->c31->c21->c11->newwindow->sink
   QUEUES FOR QUEUEING TIME DIST:rtq
      VALUES:.5 1 1.5 2 2.5
   QUEUES FOR TOKEN USE DIST:windowq
      MAX VALUE:2*windowsize-1
   QUEUES FOR TOTAL TOKEN DIST:windowq
      MAX VALUE:2*windowsize-1
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:rtq
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:rtq
         DEPARTURES:10000
   LIMIT - CP SECONDS:250
   TRACE:no
END
```

The prty (priority) queueing discpline is used to give the pacing replies priority over the data messages. Note that the measures of total tokens in the passive queue pool become interesting when create and destroy nodes are present. In this model it is possible to have up to 2×windowsize−1 tokens in the pool. In dialogue files, it is possible to specify that distributions of tokens in use and total tokens be gathered, as illustrated above. The syntax is essentially the same as for queue length distributions. The following can be obtained with EVAL:

```
RESQ2 VERSION DATE: OCTOBER 16, 1981
MODEL:PACING
WINDOWSIZE:6
```

```
WARNING -- MODEL MAY NOT BE TRULY REGENERATIVE
           BECAUSE OF USE OF GLOBAL VARIABLES
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      7568.48047
                        CPU TIME:           145.11
                 NUMBER OF EVENTS:          170544
                 NUMBER OF CYCLES:            2149

WHAT:utbo(*)


ELEMENT          UTILIZATION
RTQ              2.6288E-09(2.5138E-09,2.7437E-09) 0.0%
WINDOWQ          0.75093(0.73342,0.76845) 3.5%
Q1               0.50775(0.50135,0.51414) 1.3%
 C1R              0.50112(0.49476,0.50747) 1.3%
 C1L              6.6275E-03(6.5377E-03,6.7173E-03) 0.0%
Q2               0.50775(0.50135,0.51414) 1.3%
 C2R              0.50112(0.49476,0.50747) 1.3%
 C2L              6.6275E-03(6.5377E-03,6.7173E-03) 0.0%
Q3               0.50775(0.50135,0.51414) 1.3%
 C3R              0.50112(0.49476,0.50747) 1.3%
 C3L              6.6275E-03(6.5377E-03,6.7173E-03) 0.0%
Q4               0.50775(0.50135,0.51414) 1.3%
 C4R              0.50112(0.49476,0.50747) 1.3%
 C4L              6.6275E-03(6.5377E-03,6.7173E-03) 0.0%

WHAT:tubo


ELEMENT          MEAN TOKENS IN USE
RTQ              5.64520(5.39842,5.89198) 8.7%
WINDOWQ          4.50559(4.40052,4.61067) 4.7%

WHAT:tudbo


ELEMENT          DISTRIBUTION OF TOKENS IN USE
WINDOWQ            0:0.07033(0.06581,0.07486) 0.9%
                   1:0.09848(0.09344,0.10353) 1.0%
                   2:0.11277(0.10769,0.11785) 1.0%
                   3:0.11347(0.10904,0.11791) 0.9%
                   4:0.11165(0.10789,0.11540) 0.8%
                   5:0.11517(0.11188,0.11847) 0.7%
                   6:0.13371(0.12858,0.13885) 1.0%
                   7:0.09138(0.08688,0.09588) 0.9%
                   8:0.06271(0.05881,0.06662) 0.8%
                   9:0.04551(0.04150,0.04952) 0.8%
                  10:0.02739(0.02430,0.03047) 0.6%
                  11:0.01742(0.01481,0.02003) 0.5%
```

```
WHAT:ttbo

ELEMENT          MEAN TOTAL TOKENS IN POOL
RTQ              2.1475E+09(2.1474E+09,2.1476E+09) 0.0%
WINDOWQ          7.87949(7.84877,7.91022) 0.8%

WHAT:ttdbo

ELEMENT          DISTRIBUTION OF TOTAL TOKENS IN POOL
                   1:1.1777E-03(8.6775E-04,1.4877E-03) 0.1%
                   2:3.0013E-03(2.5671E-03,3.4356E-03) 0.1%
                   3:0.01040(0.00953,0.01127) 0.2%
                   4:0.02528(0.02398,0.02658) 0.3%
                   5:0.06318(0.06140,0.06496) 0.4%
                   6:0.17828(0.17243,0.18412) 1.2%
                   7:0.15969(0.15417,0.16520) 1.1%
                   8:0.15307(0.14791,0.15822) 1.0%
                   9:0.15729(0.15191,0.16266) 1.1%
                  10:0.13858(0.13316,0.14399) 1.1%
                  11:0.11007(0.10471,0.11543) 1.1%

WHAT:qlbo(*)

ELEMENT          MEAN QUEUE LENGTH
RTQ              5.64520(5.39842,5.89198) 8.7%
WINDOWQ          5.64520(5.39842,5.89198) 8.7%
Q1               1.16712(1.12730,1.20694) 6.8%
 C1R              1.13603(1.09643,1.17563) 7.0%
 C1L              0.03109(0.02952,0.03266) 10.1%
Q2               1.03166(1.00930,1.05402) 4.3%
 C2R              0.99569(0.97408,1.01730) 4.3%
 C2L              0.03597(0.03400,0.03794) 11.0%
Q3               1.14772(1.12481,1.17063) 4.0%
 C3R              1.11837(1.09609,1.14066) 4.0%
 C3L              0.02935(0.02778,0.03092) 10.7%
Q4               1.26213(1.23699,1.28727) 4.0%
 C4R              1.25550(1.23040,1.28061) 4.0%
 C4L              6.6275E-03(6.5377E-03,6.7173E-03) 2.7%

WHAT:qtbo(*)

ELEMENT          MEAN QUEUEING TIME
RTQ              1.41964(1.36345,1.47584) 7.9%
WINDOWQ          1.41964(1.36345,1.47584) 7.9%
Q1               0.25158(0.24403,0.25912) 6.0%
 C1R              0.28569(0.27692,0.29446) 6.1%
 C1L              0.04691(0.04509,0.04873) 7.8%
Q2               0.22238(0.21838,0.22638) 3.6%
 C2R              0.25039(0.24587,0.25492) 3.6%
 C2L              0.05428(0.05209,0.05647) 8.1%
```

```
Q3              0.24739(0.24335,0.25144) 3.3%
 C3R             0.28125(0.27665,0.28584) 3.3%
 C3L             0.04429(0.04254,0.04603) 7.9%
Q4              0.27206(0.26759,0.27652) 3.3%
 C4R             0.31573(0.31053,0.32093) 3.3%
 C4L             0.01000

WHAT:qtdbo


ELEMENT         QUEUEING TIME DISTRIBUTION
RTQ             5.00E-01:0.15992(0.15131,0.16854) 1.7%
                1.00E+00:0.40324(0.38651,0.41998) 3.3%
                1.50E+00:0.61713(0.59623,0.63802) 4.2%
                2.00E+00:0.76445(0.74380,0.78511) 4.1%
                2.50E+00:0.86317(0.84620,0.88014) 3.4%

WHAT:
CONTINUE RUN:no

WINDOWSIZE:
```

With this pacing mechanism and the chosen parameter values, mean response is only slightly increased (from 1.18 to 1.42) but we now know that the destination will never need more than 11 buffers (and if there were more origin destination pairs, unnecessary network congestion might be avoided.)

11. SPLIT, FISSION AND FUSION NODES

11.1. Split Nodes

Split nodes allow a job to produce additional independent jobs. Split nodes are often used in models of communication systems to create control messages, e.g., for acknowledgements or flow control mechanisms. A split node has one entrance, an exit for the job that entered and an additional exit for each new job to be created. The created jobs are given the same job variable values as the creating job. The created jobs do not possess tokens, whether or not the creating job possessed tokens.

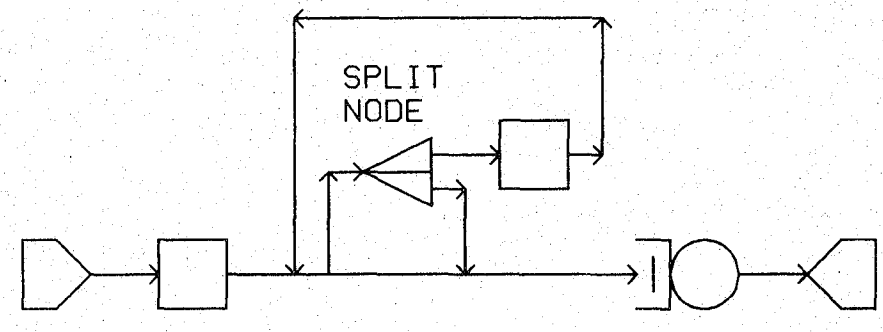


Figure 11.1 - Bulk Arrivals

As an abstract example of use of split nodes, suppose we wish to represent a queue with bulk arrivals, i.e., with several jobs arriving at the same time. A single arriving job can become several by going through a split node. See Figure 11.1. A job arriving from the source goes to a set node where JV(0) is set to the actual number of jobs to arrive. If that number is greater than one, the original job goes to the split node. (Note that the split node is a separate node; i.e., another node, e.g., a class, cannot serve as a split node.) One created job leaves the split node through the second (bottom) exit and goes on to the queue. The original job leaves the split node through the first (top) exit. In our diagrams of split nodes we have exactly one exit from the upper half of the triangle and one exit from the bottom half for each job created. The original job goes to a set node to decrement JV(0). If JV(0) is still greater than one, the original job goes to the split node again; otherwise it goes to the queue. Let the number of arriving jobs be equally likely to be any value from one up to maxjobs. We could use the following dialogue file:

```
MODEL:bulk
   METHOD:simulation
   NUMERIC PARAMETERS:atime maxjobs stime
   QUEUE:q
      TYPE:fcfs
      CLASS LIST:c
         SERVICE TIMES:stime
   SET NODES:       setcount                              deccount
   ASSIGNMENT LIST:jv(0)=ceil(uniform(0,maxjobs,1)) jv(0)=jv(0)-1
   SPLIT NODES:splitnode
   CHAIN:ch
      TYPE:open
      SOURCE LIST:s
      ARRIVAL TIMES:atime
      :s->setcount->splitnode c;if(jv(0)>1) if(t)
      :splitnode->deccount c;split
```

```
      :deccount->splitnode c;if(jv(0)>1) if(t)
      :c->sink
   QUEUES FOR QUEUEING TIME DIST:q
      VALUES:10 20 30 40 50
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION -
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:q
         MEASURES:qt
         ALLOWED WIDTHS:10
      EXTRA SAMPLING PERIODS:0
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:q
         DEPARTURES:10000
   LIMIT - CP SECONDS:300
   TRACE:no
END
```

As with other nodes, the name of a split node may be any legal RESQ name; we use "splitnode" to help clarify the syntax for split nodes.

It is not necessary to give the name of a split node before the routing definition. For example, we could omit the line

```
SPLIT NODES:splitnode
```

from the above dialogue file and the file would still be accepted by SETUP. When SETUP sees the name of a split node in the routing, it does not know whether that name is intended to be that of a new (split) node or whether it it is a misspelling of the name of another node. For this reason, if we omit the above line, when SETUP encounters the following line

```
      :s->setcount->splitnode c;if(jv(0)>1) if(t)
```

it produces the following warning message at the terminal and in the RQ2LIST file

```
**ERROR**  WNG: THE NODE "SPLITNODE " HAS BEEN IMPLICITLY DECLARED
```

SETUP knows that splitnode is a split node from the routing transition with splitnode as the from node:

```
      :splitnode->deccount c;split
```

This transition indicates that splitnode is a split node, with the job which entered the node going to deccount when it leaves and with one new job going to node c from the split node. SETUP does not prompt interactively for names of split nodes. Fission nodes (Section 11.2) and the dummy nodes we discuss below are treated similarly by SETUP.

In general a routing transition for a split node would have the form

```
from_node -> to_node1 to_node2 ... to_nodeN ; split
```

where from_node is a split node, the job which entered from_node would go to to_node$_1$, $N-1$ new jobs would be created by a visit to the split node and they would go to the remain-

ing nodes to the right of the arrow. As with other routing transitions, a node name may be used several times on the right hand side of the routing transition.

Note that the jobs leaving a split node are not allowed to choose a destination. In the example model, the creating job always goes to the set node and the created job always goes to the class. In general, we might wish to make routing decisions for the jobs leaving the split node; this is a purpose for dummy nodes. A dummy node has no effect on a job. We can specify a dummy node as the destination for a job leaving a split node. Then the usual routing decision mechanism is available for jobs leaving the dummy node. For example, suppose that in the bulk arrival model we want the number of arrivals to have a geometric distribution (starting at one) with mean meanjobs. To avoid the warning message, we could use the following line after the split node definition

```
DUMMY NODES:dummynode
```

Then we could use the following routing definition

```
:s->splitnode c;1-1/meanjobs 1/meanjobs
:splitnode->dummynode c;split
:dummynode->splitnode c;1-1/meanjobs 1/meanjobs
:c->sink
```

The name of a dummy node can be any legal RESQ name; we use "dummynode" for clarity.

With model bulk as first defined above the mean number of jobs arriving at one time will be (maxjobs+1)/2. Using EVAL we can get

```
RESQ2 VERSION DATE: OCTOBER 20, 1981
MODEL:BULK
ATIME:55
MAXJOBS:10
STIME:5
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
SAMPLING PERIOD END: Q DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      2.9918E+05
                        CPU TIME:            30.30
                NUMBER OF EVENTS:            35511
                NUMBER OF CYCLES:             2716

WHAT:utbo

ELEMENT         UTILIZATION
Q               0.50105(0.48742,0.51468) 2.7%

WHAT:tpbo

ELEMENT         THROUGHPUT
Q               0.10030(0.09779,0.10281) 5.0%
SETCOUNT        0.01840
```

```
DECCOUNT        0.08190
SPLITNODE       0.08190
S               0.01840
SINK            0.10030

WHAT:qlbo


ELEMENT         MEAN QUEUE LENGTH
Q               4,00728(3.75102,4.26354) 12.8%

WHAT:qtbo


ELEMENT         MEAN QUEUEING TIME
Q               39.95358(38.08224,41.82491) 9.4%

WHAT:qtdbo


ELEMENT         QUEUEING TIME DISTRIBUTION
Q               1.00E+01:0.17989(0.17171,0.18807) 1.6%
                2.00E+01:0.34045(0.32704,0.35386) 2.7%
                3.00E+01:0.48915(0.47173,0.50657) 3.5%
                4.00E+01:0.60762(0.58830,0.62695) 3.9%
                5.00E+01:0.70237(0.68243,0.72231) 4.0%

WHAT:
CONTINUE RUN:no

ATIME:
```

Note that split node throughput is measured in entered jobs, i.e., created jobs are not counted.

11.2. Fission and Fusion Nodes

Fission nodes allow a job to create additional jobs dependent on the creating job. Fusion nodes allow for the destruction of the created jobs in a coordinated manner. Fission and fusion nodes are usually used together in pairs. Fission and fusion nodes are useful for representing synchronized processes (tasks) occurring in operating systems. Similarly, fission and fusion nodes are useful for representing parallel physical activities representing a single logical activity, for example transmission of a message across a communication network as a collection of packets.

A fission node has one entrance, an exit for the job that entered (referred to as the "parent"), and an additional exit for each new job to be created. The created jobs are referred to as "children." Children may themselves enter fission nodes, thus creating hierarchies of jobs (see Section 8 of the Users Guide). Children are given the same job variable values as the parent. The children do not possess tokens, whether or not the parent does. *Jobs are not allowed to go to sinks as long as they have relatives (parents or children).* If this rule is violated, the simulation terminates.

In our diagrams we represent a fission node by a triangle with the entrance at one vertex and the exits on the opposite side. This corresponds to the split node representation except that the triangle is not divided into separate sub-triangles for the parent and children exits. In the dialogue syntax, fission nodes are treated exactly the same as split nodes, except that (1) the keyword "FISSION" is used instead of the keyword "SPLIT," (2) there is an interactive prompt to optionally declare the names of fission nodes, and (3) in dialogue files, if the names of fission nodes are declared before the routing definition they are defined after declarations for split nodes, if any are present.

A fusion node provides a place for jobs to wait for related jobs (a parent or children). (A fusion node acts as a dummy node for jobs without relatives, i.e., such jobs pass through a fusion node without delay or other effect.) No more than one job of a "family" can stay at a fusion node. If a job arrives at a fusion node and it has relatives, but none of its relatives are at this particular fusion node, it waits at the fusion nodes. When a job arrives at a fusion node and it has a relative at this particular fusion node, two things can happen, depending on the relationship between the jobs. If one is the parent and the other is a child, then the offspring is destroyed. If both are children, the one that was created last is destroyed. Before a child is destroyed, any tokens it holds are released. After destruction of one job, if the other job has no remaining relatives, it proceeds from the exit of the fusion node. If the other job still has other relatives, it waits at the fusion node for another relative to arrive.

In our diagrams we represent fusion nodes by a triangle with the exit at one vertex and the entrance(s) on the opposite side. Fusion nodes must be declared immediately before the routing, e.g.,

```
FUSION NODES:fusionnode
```

Fusion nodes appear in the routing without further distinction, i.e., there is no need for a keyword as in the case of split and fission nodes.

A natural application of fission and fusion nodes is to represent messages transmitted as packets in a communications network. In our loop model, an alternative to full-duplex links which might significantly improve performance would be to break long messages into packets, to be transmitted separately. Let us assume that the maximum packet size is to be 240 bits. If a message exceeds 240 bits, it will be broken into two or more packets. We represent this by sending a job with JV(0)>240 to a fission node. The parent leaving the fission node has JV(0) decremented by 240 and the created job has JV(0) set to 240. (We are ignoring the headers and/or trailers which would be necessary on each packet.) If the parent still has JV(0)>240 it goes to the fission node again. When a packet gets to its destination, it goes to a fusion node which represents assembly of the packets into the original message. When the parent and all of its children (if any) have made it to the fusion node, the parent leaves the fusion node. Figure 11.2 shows the fission and fusion nodes but not the set nodes. The following dialogue file could be used:

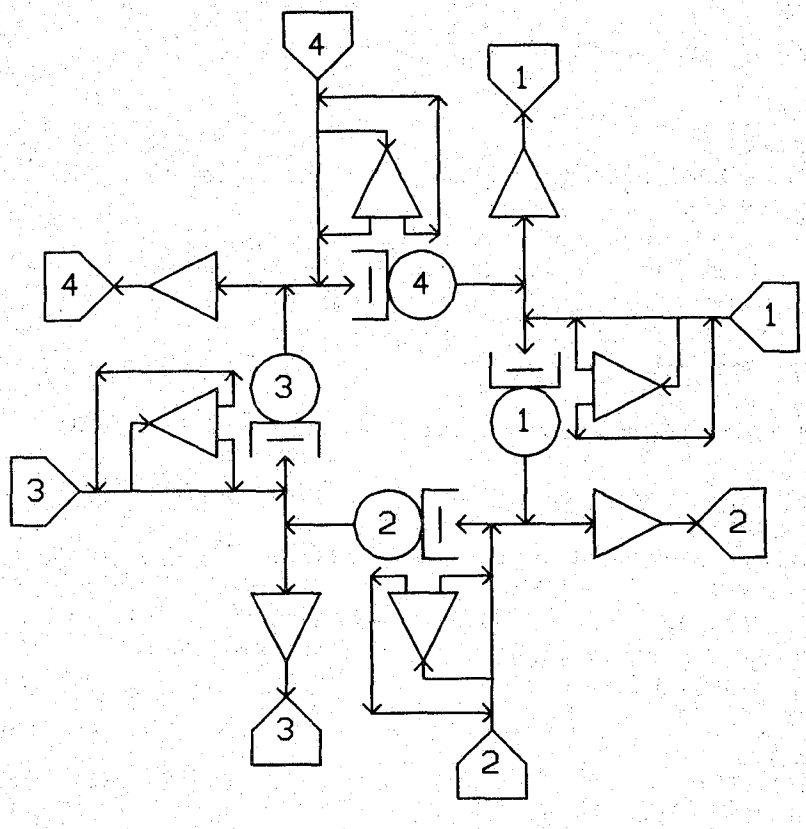


Figure 11.2 - Packetizing of Messages

```
MODEL:loop
   METHOD:simulation
   NUMERIC IDENTIFIERS:mean_atime
      MEAN_ATIME:.1
   NUMERIC IDENTIFIERS:totlength capacity
      TOTLENGTH:720
      CAPACITY:4800
   NUMERIC IDENTIFIERS:msg_dest pkt_leng
      MSG_DEST:0 /*JV to be used*/
      PKT_LENG:1 /*JV to be used*/
   MAX JV:1
   QUEUE:rtq
      TYPE:passive
      TOKENS:2147483647
      DSPL:fcfs
      ALLOCATE NODE LIST:beginrt
         NUMBERS OF TOKENS TO ALLOCATE:1
   QUEUE:q1
      TYPE:fcfs
      CLASS LIST:c1
         WORK DEMANDS:standard(jv(pkt_leng)/capacity,0)
   QUEUE:q2
      TYPE:fcfs
      CLASS LIST:c2
         WORK DEMANDS:standard(jv(pkt_leng)/capacity,0)
   QUEUE:q3
```

```
   TYPE:fcfs
   CLASS LIST:c3
      WORK DEMANDS:standard(jv(pkt_leng)/capacity,0)
QUEUE:q4
   TYPE:fcfs
   CLASS LIST:c4
      WORK DEMANDS:standard(jv(pkt_leng)/capacity,0)
SET NODES:set_msg_1
ASSIGNMENT LIST:jv(pkt_leng)=standard(totlength,1)
SET NODES:dec_msg_11 dec_msg_12 dec_msg_13 dec_msg_14
ASSIGNMENT LIST:jv(pkt_leng)=jv(pkt_leng)-240
SET NODES:set_pkt_11 set_pkt_12 set_pkt_13 set_pkt_14
ASSIGNMENT LIST:jv(pkt_leng)=240
SET NODES:dest1
ASSIGNMENT LIST:jv(msg_dest)=discrete(2,1/3; 3,1/3; 4,1/3)
SET NODES:dest2
ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 3,1/3; 4,1/3)
SET NODES:dest3
ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 2,1/3; 4,1/3)
SET NODES:dest4
ASSIGNMENT LIST:jv(msg_dest)=discrete(1,1/3; 2,1/3; 3,1/3)
FISSION NODES:separate1 separate2 separate3 separate4
FUSION NODES:assemble
CHAIN:c
   TYPE:open
   SOURCE LIST:s
   ARRIVAL TIMES:mean_atime
   :s->beginrt->set_msg_1->dest1 dest2 dest3 dest4
   :dest1->c1 separate1;if(jv(pkt_leng)<=240) if(t)
   :separate1->dec_msg_11 set_pkt_11;fission
   :dec_msg_11->c1 separate1;if(jv(pkt_leng)<=240) if(t)
   :dest2->c2 separate2;if(jv(pkt_leng)<=240) if(t)
   :separate2->dec_msg_12 set_pkt_12;fission
   :dec_msg_12->c2 separate2;if(jv(pkt_leng)<=240) if(t)
   :dest3->c3 separate3;if(jv(pkt_leng)<=240) if(t)
   :separate3->dec_msg_13 set_pkt_13;fission
   :dec_msg_13->c3 separate3;if(jv(pkt_leng)<=240) if(t)
   :dest4->c4 separate4;if(jv(pkt_leng)<=240) if(t)
   :separate4->dec_msg_14 set_pkt_14;fission
   :dec_msg_14->c4 separate4;if(jv(pkt_leng)<=240) if(t)
   :set_pkt_11 set_pkt_12 set_pkt_13 set_pkt_14->c1 c2 c3 c4
   :c1->assemble c2;if(jv(msg_dest)=2) if(t)
   :c2->assemble c3;if(jv(msg_dest)=3) if(t)
   :c3->assemble c4;if(jv(msg_dest)=4) if(t)
   :c4->assemble c1;if(jv(msg_dest)=1) if(t)
   :assemble->sink
QUEUES FOR QUEUEING TIME DIST:rtq
   VALUES:.6 1.2 1.8 2.4 3.0 3.6
CONFIDENCE INTERVAL METHOD:regenerative
REGENERATION STATE DEFINITION -
CONFIDENCE LEVEL:90
SEQUENTIAL STOPPING RULE:yes
   QUEUES TO BE CHECKED:rtq
      MEASURES:qt
```

```
        ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES -
      QUEUES FOR DEPARTURE COUNTS:rtq
         DEPARTURES:10000
   LIMIT - CP SECONDS:1000
   TRACE:no
END
```

We could then get the following from EVAL:

```
RESQ2 VERSION DATE: OCTOBER 20, 1981
MODEL:LOOP
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      5085.87891
                        CPU TIME:         488.95
                 NUMBER OF EVENTS:         408413
                 NUMBER OF CYCLES:           1318

WHAT:utbo


ELEMENT          UTILIZATION
RTQ              4.1214E-09(3.9229E-09,4.3200E-09) 0.0%
Q1               0.75700(0.74489,0.76910) 2.4%
Q2               0.74786(0.73703,0.75869) 2.2%
Q3               0.74743(0.73574,0.75912) 2.3%
Q4               0.73841(0.72680,0.75002) 2.3%

WHAT:tpbo(rtq)


ELEMENT          THROUGHPUT
RTQ              9.96209(9.89882,10.02536) 1.3%

WHAT:qlbo(rtq)


ELEMENT          MEAN QUEUE LENGTH
RTQ              8.85070(8.42426,9.27715) 9.6%

WHAT:qtbo(rtq)


ELEMENT          MEAN QUEUEING TIME
RTQ              0.88844(0.84769,0.92919) 9.2%

WHAT:sdqt(rtq)
```

```
ELEMENT          STANDARD DEVIATION OF QUEUEING TIME
RTQ              0.81473

WHAT:qtdbo


ELEMENT          QUEUEING TIME DISTRIBUTION
RTQ              6.00E-01:0.46939(0.45398,0.48480) 3.1%
                 1.20E+00:0.73667(0.71883,0.75450) 3.6%
                 1.80E+00:0.87060(0.85549,0.88572) 3.0%
                 2.40E+00:0.93901(0.92868,0.94934) 2.1%
                 3.00E+00:0.97377(0.96776,0.97978) 1.2%
                 3.60E+00:0.98899(0.98564,0.99233) 0.7%

WHAT:
CONTINUE RUN:/*Continue run:*/ yes

EXTRA SAMPLING PERIODS:/*Extra periods:*/ 1

SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: RTQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                   SIMULATED TIME:     6108.71094
                         CPU TIME:          583.68
                  NUMBER OF EVENTS:          489485
                  NUMBER OF CYCLES:            1604

WHAT:qtbo(rtq)


ELEMENT          MEAN QUEUEING TIME
RTQ              0.88052(0.84438,0.91665) 8.2%

WHAT:nd(rtq)


ELEMENT          NUMBER OF DEPARTURES
RTQ              60777

WHAT:qtdbo


ELEMENT          QUEUEING TIME DISTRIBUTION
RTQ              6.00E-01:0.47112(0.45714,0.48509) 2.8%
                 1.20E+00:0.73878(0.72295,0.75461) 3.2%
                 1.80E+00:0.87416(0.86082,0.88750) 2.7%
                 2.40E+00:0.94169(0.93257,0.95080) 1.8%
                 3.00E+00:0.97524(0.96993,0.98055) 1.1%
                 3.60E+00:0.98972(0.98678,0.99265) 0.6%
```

```
WHAT:
CONTINUE RUN:/*Continue run:*/ no
```

The mean response time estimate, .88 seconds, is substantially improved over the estimate for the original model, 1.16 seconds.

12. QUEUE TYPES

A queue type is a parameterized macro definition of a queue. Queue types are usually used to create multiple instances of a frequently used type of queue. For example, if fcfs were not a predefined special RESQ queue type, we could define a corresponding queue type using the queue type facility.

There are two distinct operations involved in the use of queue types: the definition of a queue type and the invocation of a queue type. The queue type definition consists of the specification of a parameterized queue template in which some of the queue type characteristics are given explicit values and other queue type characteristics are given parametric values. The explicit values become the default characteristics of the queue type. Once a queue type has been defined, it can later be invoked to create a specific instance of a queue. As we shall see, a set of parameter values is given as part of the invocation. A queue defined by an invocation of a queue type assumes the default characteristics of the queue type and the parametric characteristics given by the set of parameter values in the invocation.

A frequently used queue is a simple passive first come first served (pfcfs) queue which has no create or destroy nodes, a fcfs queueing discipline and a single allocate node at which a single token is allocated. Since such a queue is frequently used, we might want to define it as a special queue type. A definition of a pfcfs queue type is shown in the model below. The following dialogue corresponds to the first version of csmwm in Section 4.

```
MODEL:csmwm
   METHOD:simulation
   NUMERIC PARAMETERS:thinktime users partitions
   NUMERIC IDENTIFIERS:floppytime disktime cputime thinktime users
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE TYPE:pfcfs              /* passive fcfs queue template */
      NUMERIC PARAMETERS:ntokens    /* number of tokens in pool */
      NODE PARAMETERS:alloc releas
      TYPE:passive
      TOKENS:ntokens
      DSPL:fcfs
      ALLOCATE NODE LIST:alloc
         NUMBERS OF TOKENS TO ALLOCATE:1
      RELEASE NODE LIST:releas
   END OF QUEUE TYPE PFCFS
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:floppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:fcfs
      CLASS LIST:cpu
         SERVICE TIMES:cputime
```

```
QUEUE:terminalsq
   TYPE:is
   CLASS LIST:terminals
      SERVICE TIMES:thinktime
QUEUE:memory              /* define the passive memory queue  */
   TYPE:pfcfs             /* by invoking the pfcfs queue type */
   NTOKENS:partitions
   ALLOC:getmemory
   RELEAS:freememory
CHAIN:interactiv
   TYPE:closed
   POPULATIONS:users
   :terminals->getmemory->cpu->floppy disk; .1 .9
...
```

The queue type definitions immediately precede the queue definitions. Note that with the exception of the parameter declarations, the body of a queue type is similar to the body of a queue definition. The parameter declarations themselves are similar to model parameter declarations with the exception of the NODE PARAMETERS: prompt. All nodes and classes used within the body of a queue type must be declared as a node parameter of the queue type.

In the above model, the pfcfs queue type is invoked once to define the memory queue. A queue type is invoked by giving the previously defined queue type name in response to the TYPE prompt of a standard queue definition. In addition to giving the name of the queue type to be invoked, we must also supply values for the parameters of the queue type; this is done immediately following the TYPE prompt. Further discussion of queue types is given in Section 6 of the Users Guide.

13. SUBMODELS

Submodels provide a facility for macro definition of subnetworks. A submodel is a template for a subnetwork which the user wishes to explicitly delineate (1) because this clarifies model structure, (2) because several such subnetworks (with parameterizable differences) appear in a model and/or (3) because this submodel is to be (may be) used in other models.

When one uses ("invokes") a submodel with a set of parameter values, then a set of queues and nodes with the specified values and relationships is added to the network, just as invocation of an assembly language macro causes a set of instructions to be added to the program. It is important for the user to think in terms of macros rather than procedures in properly understanding submodels and how they may be used.

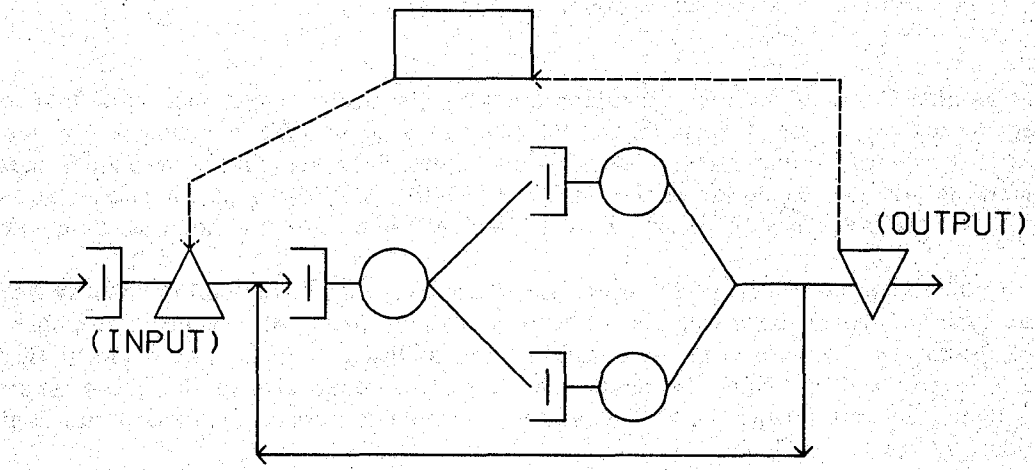


Figure 13.1 - Computer System Submodel

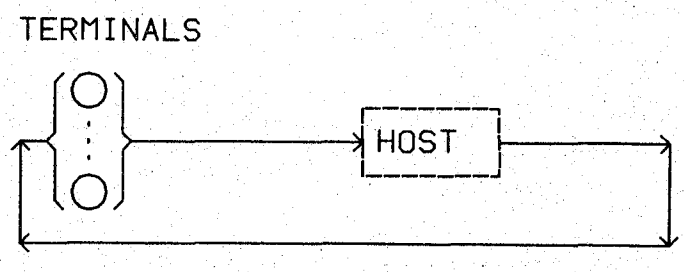


Figure 13.2 - Network with Submodel Invocation

Many of the examples we have given are easily (and appropriately) restated using submodels. For example, consider the computer system model csmwm. Let the entire network except for the terminals queue be considered a submodel, as depicted in Figure 13.1. After specifying the submodel, we can invoke it with parameters in a network corresponding to the previous model (Figure 13.2) and in other networks. The following dialogue file portion could define the submodel.

```
SUBMODEL:cssm /*Computer System SubModel*/
   NUMERIC PARAMETERS:pageframes floppytime disktime cputime
   CHAIN PARAMETERS:chn
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE:floppyq
```

```
      TYPE:fcfs
      CLASS LIST:floppy
         SERVICE TIMES:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIMES:disktime
   QUEUE:cpuq
      TYPE:ps
      CLASS LIST:cpu
         SERVICE TIMES:cputime
   QUEUE:memory
      TYPE:passive
      TOKENS:pageframes
      DSPL:fcfs
      ALLOCATE NODE LIST:getmemory
         NUMBERS OF TOKENS TO ALLOCATE:discrete(16,.25;32,.5;48,.25)
      RELEASE NODE LIST:freememory
   CHAIN:chn
      TYPE:external
      INPUT:getmemory
      OUTPUT:freememory
      :getmemory->cpu
      :cpu->floppy disk;.1 .9
      :floppy->freememory cpu;1/cpiocycles 1-1/cpiocycles
      :disk->freememory cpu;1/cpiocycles 1-1/cpiocycles
END OF SUBMODEL CSSM
```

Notice that the dialogue very closely parallels the dialogue for an entire model. We focus on the differences.

A submodel is only part of a network. For a network including a submodel to be meaningful, there must be at least one chain which is partially defined inside the submodel and partially defined outside the submodel. Such chains must be declared as chain parameters of the submodel. In the current example, there is only one chain. It has the name "chn" inside the submodel and is declared as a parameter.

Additional identifiers (numeric, distribution, global variable) may be defined in a submodel declaration. Identifiers defined outside a submodel may be used within a submodel. Names of identifiers may be reused within submodels. The rules for doing so are exactly the same as in block structured programming languages such as PL/I and PASCAL.

As we said, chains declared as chain parameters are defined partly inside a submodel and partly outside a submodel. The type of a chain parameter is defined as "external" because the usual type, open or closed, is not determined until the chain definition is completed outside of the submodel.

In many situations, submodels can be used with minimal knowledge of the contents of the submodel. To this end, it is possible to give exactly one node of each chain parameter the synonym "input" and to give exactly one node of each chain parameter the synonym "output." When the submodel is invoked, and the chain definition completed, these nodes may be referred to by these synonyms instead of the names used within the submodel. It is intended that node input be the primary (usually the only) entry point seen by the invoking model and that node output be the primary (usually the only) exit point seen by the invoking

model. (The Users Guide discusses and illustrates use of node parameters for multiple entry and exit points per chain. See Section 10 and Appendix 1 of the Users Guide.) In the example, the allocate node and release node are used as the input and output, respectively, of the routing chain.

The following uses submodel cssm to obtain a model definition equivalent to the csmwm definition of Section 6.

```
MODEL:csm
   METHOD:simulation
   NUMERIC PARAMETERS:thinktime users pageframes
   NUMERIC IDENTIFIERS:floppytime disktime cputime
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
   NUMERIC IDENTIFIERS:cpiocycles
      CPIOCYCLES:8
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   SUBMODEL:cssm /*Computer System SubModel*/
      ...
   END OF SUBMODEL CSSM
   INVOCATION:host
      TYPE:cssm
      PAGEFRAMES:pageframes
      FLOPPYTIME:floppytime
      DISKTIME:disktime
      CPUTIME:cputime
      CHN:interactiv
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->host.input
      :host.output->terminals
   QUEUES FOR QUEUEING TIME DIST:host.memory
      VALUES:1 2 3 4 5 6 7 8
   QUEUES FOR QUEUE LENGTH DIST:host.memory
      MAX VALUE:users/2
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      REGEN POP:users
      INIT POP:users
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:host.memory
         MEASURES:qt
         ALLOWED WIDTHS:10
   SAMPLING PERIOD GUIDELINES-
      QUEUES FOR DEPARTURE COUNTS:host.memory
         DEPARTURES:1000
```

```
   LIMIT - CP SECONDS:300
   TRACE:no
END
```

We have not repeated the full definition of the submodel here. In the actual file, it would not be necessary to use the definition if the definition existed as a separate file and were logically inserted at the appropriate point using the INCLUDE statement described in Section 2 of the Users Guide.

An invocation is a specific instance of a submodel with its own parameter values (and its own nodes, queues and "global" variables which were defined locally within the submodel). The INVOCATION prompt requests a name for the invocation. This name will be needed later for qualification of the names of elements of the submodel. The TYPE prompt requests the name of the submodel being invoked. After giving the submodel name, the remaining prompts of the invocation are for parameter values. For numeric and distribution parameters, the values given must be expressions consisting of constants and previously defined identifiers (possibly including model parameter identifiers). The value given for a chain parameter will be the first appearance of that chain name, unless it has previously been used in an invocation or it is a chain array (see Section 3 of the Users Guide).

Subsequent to the invocation, when it is necessary to refer to elements of the invoked submodel, these names are qualified by the invocation name in the form "invocation" "." "element". In the example, in the routing the allocate node is referred to as "host.input" and the release node is referred to as "host.output". In the simulation specific dialogue, the memory queue is referred to as host.memory.

The RQ2LIST indicates the level of nesting of submodel definition in the column after the line number:

```
RESQ Translator V2.04 (10/02/81)  Time: 13:42:23  Date: 10/26/81

*   1*   0*  MODEL:csm
*   2*   0*      METHOD:simulation
*   3*   0*      NUMERIC PARAMETERS:thinktime users pageframes
    ...
*  13*   0*           SERVICE TIMES:thinktime
*  14*   0*      SUBMODEL:cssm /*Computer System SubModel*/
*  15*   1*         NUMERIC PARAMETERS:pageframes floppytime disktime cputime
*  16*   1*         CHAIN PARAMETERS:chn
    ...
*  45*   1*            :disk->freememory cpu;1/cpiocycles 1-1/cpiocycles
*  46*   1*      END OF SUBMODEL CSSM
*  47*   0*      INVOCATION:host
*  48*   0*         TYPE:cssm
    ...
*  80*   0*  END

 NO FATAL ERRORS DETECTED DURING COMPILATION.
```

This definition of csmwm with the submodel produces exactly the same numerical results as the definition used at the end of Section 5.2. The output format has an extra column giving invocation names, as we shall see shortly.

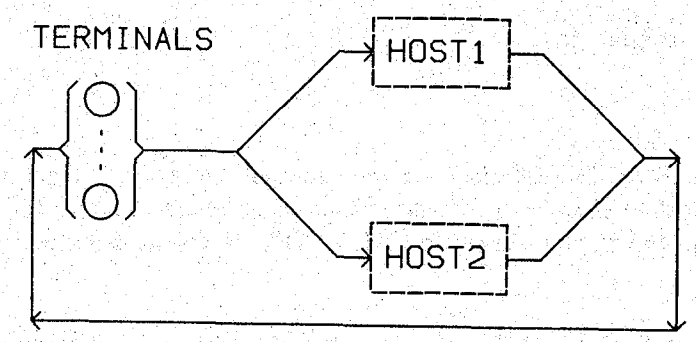


Figure 13.3 - Network with Two Invocations

Having defined submodel cssm, we can now use it several times in a model. For example, if we wished to model a pair of computer systems sharing a common set of terminals as pictured in Figure 13.3, we might use the following model definition.

```
MODEL:csm
   METHOD:simulation
   NUMERIC PARAMETERS:thinktime users pageframes
   NUMERIC IDENTIFIERS:floppytime disktime cputime1 cputime2
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME1:.05
      CPUTIME2:.075
   QUEUE:terminalsq
      TYPE:is
      CLASS LIST:terminals
         SERVICE TIMES:thinktime
   SUBMODEL:cssm /*Computer System SubModel*/
   ...
   END OF SUBMODEL CSSM
   INVOCATION:host1
      TYPE:cssm
      PAGEFRAMES:pageframes
      FLOPPYTIME:floppytime
      DISKTIME:disktime
      CPUTIME:cputime1
      CHN:interactiv
   INVOCATION:host2
      TYPE:cssm
      PAGEFRAMES:pageframes
      FLOPPYTIME:floppytime
      DISKTIME:disktime
      CPUTIME:cputime2
      CHN:interactiv
   CHAIN:interactiv
      TYPE:closed
      POPULATION:users
      :terminals->host1.input host2.input
      :host1.output host2.output->terminals
   QUEUES FOR QUEUEING TIME DIST:host1.memory host2.memory
      VALUES:1 2 3 4 5 6 7 8
      VALUES:1 2 3 4 5 6 7 8
```

```
   CONFIDENCE INTERVAL METHOD:regenerative
   REGENERATION STATE DEFINITION-
   CHAIN:interactiv
      NODE LIST:terminals
      REGEN POP:users
      INIT POP:users
   CONFIDENCE LEVEL:90
   SEQUENTIAL STOPPING RULE:yes
      QUEUES TO BE CHECKED:host1.memory host2.memory
         MEASURES:qt qt
         ALLOWED WIDTHS:10 10
   SAMPLING PERIOD GUIDELINES-
      QUEUES FOR DEPARTURE COUNTS:terminalsq
         DEPARTURES:10000
   LIMIT - CP SECONDS:1000
   TRACE:no
END
```

The dialogue is essentially the same as before, but there are now two invocations. Assuming 50 terminals and the other parameters we have used, we could get the following from EVAL.

```
RESQ2 VERSION DATE: OCTOBER 20, 1981
MODEL:CSM
THINKTIME:10
USERS:50
PAGEFRAMES:128
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:      2.0007E+04
                        CPU TIME:         542.76
                 NUMBER OF EVENTS:        1102672
                 NUMBER OF CYCLES:             13

WHAT:qtbo


INVOCATION      ELEMENT         MEAN QUEUEING TIME
                TERMINALSQ      9.95654(9.88809,10.02499) 1.4%
HOST1           MEMORY          1.67274(1.60467,1.74080) 8.1%
HOST1           FLOPPYQ         0.29691(0.29259,0.30123) 2.9%
HOST1           DISKQ           0.02387(0.02363,0.02411) 2.0%
HOST1           CPUQ            0.10543(0.10329,0.10756) 4.0%
HOST2           MEMORY          9.11460(8.72290,9.50631) 8.6%
HOST2           FLOPPYQ         0.29479(0.28634,0.30324) 5.7%
HOST2           DISKQ           0.02376(0.02362,0.02389) 1.1%
HOST2           CPUQ            0.23773(0.23517,0.24029) 2.2%

WHAT:utbo
```

```
INVOCATION      ELEMENT         UTILIZATION
                TERMINALSQ      0.00000(0.00000,0.00000)
HOST1           MEMORY          0.50065(0.48280,0.51850) 3.6%
HOST1           FLOPPYQ         0.28473(0.27743,0.29203) 1.5%
HOST1           DISKQ           0.22153(0.21569,0.22738) 1.2%
HOST1           CPUQ            0.64912(0.63248,0.66577) 3.3%
HOST2           MEMORY          0.90884(0.90452,0.91316) 0.9%
HOST2           FLOPPYQ         0.28436(0.27804,0.29068) 1.3%
HOST2           DISKQ           0.22144(0.21992,0.22297) 0.3%
HOST2           CPUQ            0.97110(0.96828,0.97392) 0.6%


WHAT:
CONTINUE RUN:yes

EXTRA SAMPLING PERIODS:1

LIMIT - CP SECONDS:2000

SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
SAMPLING PERIOD END: TERMINALSQ DEPARTURE GUIDELINE
NO ERRORS DETECTED DURING SIMULATION.

                  SIMULATED TIME:     2.4276E+04
                        CPU TIME:          660.72
                NUMBER OF EVENTS:         1338170
                NUMBER OF CYCLES:              21

WHAT:qtbo(host1.memory,host2.memory)


INVOCATION      ELEMENT         MEAN QUEUEING TIME
HOST1           MEMORY          1.66722(1.61349,1.72095) 6.4%
HOST2           MEMORY          9.16357(8.82434,9.50280) 7.4%

WHAT:
CONTINUE RUN:no

THINKTIME:
```

As we said before, the output format is essentially the same, but there is an added column to indicate the invocation.

Submodels are used extensively in the examples in Section 1 and Appendix 1 of the Users Guide.

14. PL/I EMBEDDING

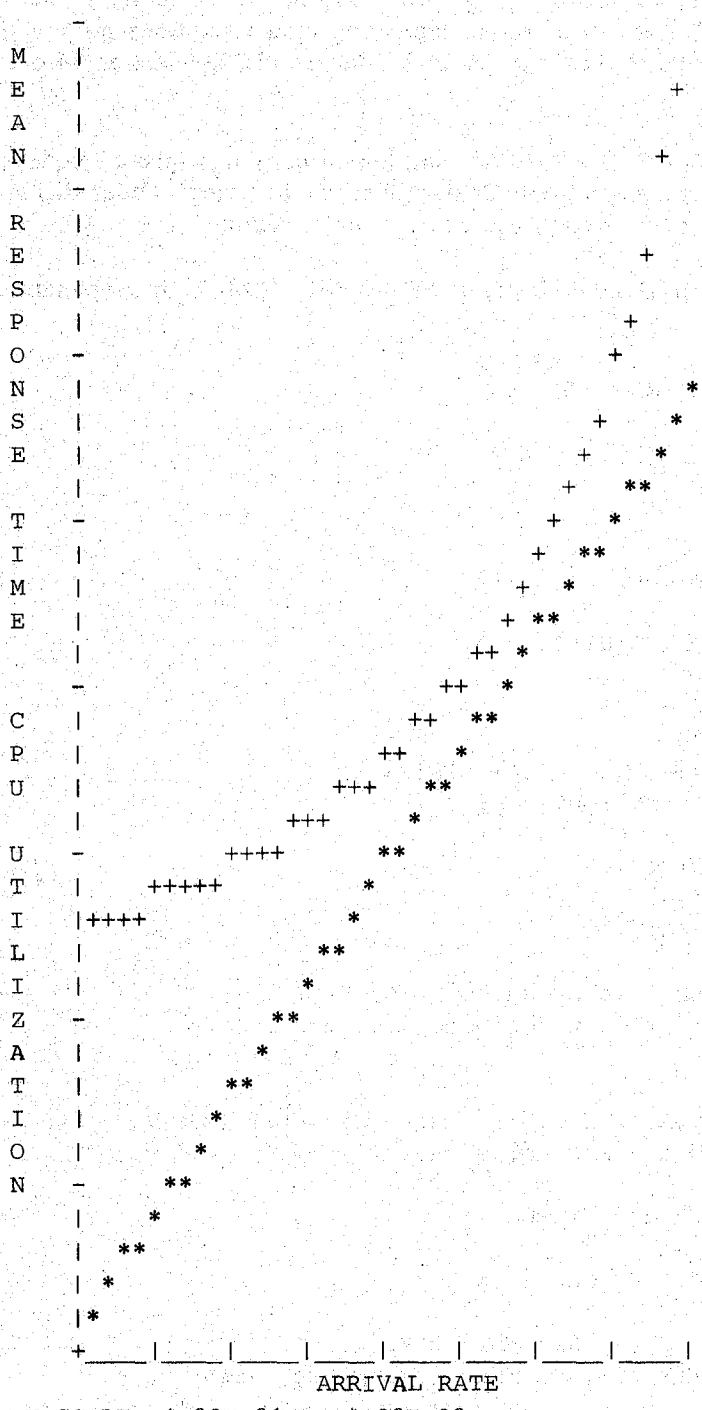
Instead of using the EVAL command after a model has been defined with the SETUP command, model expansion may be embedded within a PL/I program. This may be done in order (1) to produce tables or graphs of results, (2) to coordinate solution of several separate models in a hierarchical solution, (3) to provide a preprocessor for determining model parameters and/or (4) to provide a postprocessor for manipulating model solutions prior to display. We briefly illustrate the first two of these applications. For details, see Section 14 of the Users Guide.

Several procedures are supplied with RESQ for producing low resolution graphs of model results on a terminal, line printer or other appropriate character oriented device. Other PL/I callable graphics packages supplied by the user may be used in a similar manner.

Following is a complete program which could be used with model EXAMP1 in Appendix 1 of the Users Guide:

```
EXAMP1: PROCEDURE OPTIONS(MAIN) REORDER;
  DECLARE
    N FIXED BIN(31),
    (T,DATA(40,3),OP(3)) FLOAT BIN(21),
    FMSG CHAR(80),
    (FLOAT,SUBSTR) BUILTIN,
  /*Entry points for RESQ routines:*/
    READMD ENTRY,
    STPARM ENTRY (CHAR(10),FLOAT BIN(21)),
    RESQ2M ENTRY(FIXED BIN(31)),
    FNLMSG ENTRY(CHAR(80)),
    GTRSLT ENTRY (CHAR(*) VARYING,
                  CHAR(*) VARYING,(3) FLOAT BIN(21)),
  /*Entry points for RESQ plotting routines:*/
    RQSET  ENTRY(FIXED BIN(31),FIXED BIN(31)),
    RQPLOT ENTRY((*,*) FLOAT BIN(21)),
    RQXLBL ENTRY(CHAR(*) VARYING),
    RQYLBL ENTRY(CHAR(*) VARYING),
    RQVIEW ENTRY;
  CALL READMD; /* Reads RQ2COMP file produced by SETUP*/
  CALL STPARM('CPIOCYCLES',8.0); /*Set parameter value*/
  DO N=1 TO 40;
    DATA(N,1)=FLOAT(N)/10.0;
    CALL STPARM('ARVL_RATE',FLOAT(N)/10.0); /*Set parameter value*/
    CALL RESQ2M(0); /* Expands model & solves numerically*/
    CALL FNLMSG(FMSG);
    IF SUBSTR(FMSG,1,9)¬='NO ERRORS' THEN
      STOP;
    CALL GTRSLT('CPUQ','QL',OP); /* Get result */
    T=OP(1);
    CALL GTRSLT('DISKQ','QL',OP); /* Get result */
    DATA(N,2)=(T+OP(1))/(FLOAT(N)/10.0); /*Mean response time
                                           (Little's Rule)    */
    CALL GTRSLT('CPUQ','UT',OP); /* Get result */
    DATA(N,3)=OP(1);
  END;
  CALL RQSET(40,40);
```

```
   CALL RQPLOT(DATA);
   CALL RQXLBL('                ARRIVAL RATE');
   CALL RQYLBL('MEAN RESPONSE TIME  CPU UTILIZATION');
   CALL RQVIEW;
END;
```




```
X SCALE: 1.00E-01 -  4.00E+00
Y SCALE: 2.85E-02 -  1.17E+00
```

Figure 14.1 - Example Graph of Model Results

After compiling this procedure, with the PLIOPT command, we could use the RPLOT EXEC, e.g.,

```
rplot examp1 examp1
```

to get the plot shown in Figure 14.1.

The next example will illustrate a hierarchical model which passes values for the rates of a queue dependent server from the inner model to the outer model. See the example described in Sections 4.3 and 8.3 of [LAVE82]. Figure 14.2 illustrates the outer model, which is a closed model with two resources: (1) an infinite server representing the terminals and (2) a queue dependent server representing the computer system. The following dialogue file can be used as input to SETUP:

```
MODEL:outer
   METHOD:numerical
   NUMERIC PARAMETERS:qrates(4)
   QUEUE:termq
      TYPE:active
      DSPL:is
      CLASS LIST:terminals
         WORK DEMANDS:10
   QUEUE:csq
      TYPE:active
      DSPL:fcfs
      CLASS LIST:comsys
         WORK DEMANDS:1
      SERVER -
         RATES:qrates
   CHAIN:c1
      TYPE:closed
      POPULATION:30
      :terminals->comsys->terminals
END
```

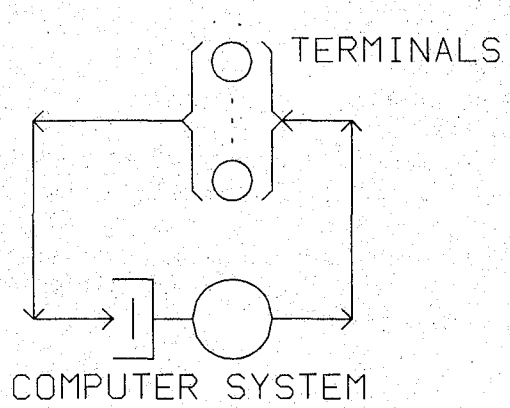


Figure 14.2 - Outer Model

The inner model is a "central server model" (Figure 14.3). The dialogue file for this model can be constructed as follows:

```
MODEL:inner
   METHOD:numerical
```

```
   NUMERIC PARAMETERS:deg_m_p
   NUMERIC IDENTIFIERS:floppytime disktime cputime
      FLOPPYTIME:.22
      DISKTIME:.019
      CPUTIME:.05
   QUEUE:floppyq
      TYPE:fcfs
      CLASS LIST:floppy
         SERVICE TIME:floppytime
   QUEUE:diskq
      TYPE:fcfs
      CLASS LIST:disk
         SERVICE TIME:disktime
   QUEUE:cpuq
      TYPE:fcfs
      CLASS LIST:cpu
         SERVICE TIME:cputime
   CHAIN:multi_prog
      TYPE:closed
      POPULATION:deg_m_p
      :cpu->floppy disk;.1 .9
      :floppy disk->cpu
END
```

The following program can be used to solve these two models hierarchically:

```
INOUT: PROC OPTIONS (MAIN) REORDER;
   DCL READMD ENTRY,   /* DCL'S for RESQ routines. */
       RESQM ENTRY(FIXED BIN(31)),
       GTRSLT ENTRY (CHAR(*) VARYING,
                     CHAR(*) VARYING,(3) FLOAT BIN(21)),
       STPARM ENTRY (CHAR(10),FLOAT BIN(21)),
       STPRMV ENTRY (CHAR(10),(*) FLOAT BIN(21));
   DCL TYPEVL ENTRY;
   DCL RSQ2IP FILE STREAM INPUT;
   DCL (DEG_M_P) FLOAT BIN(21),
       OP(3) FLOAT BIN(21),
       QRATES(4) FLOAT BIN(21);
         /* INNER MODEL */
      CALL READMD;   /* READS INNER RQ2COMP FILE FROM TRANSLATOR */
      DO DEG_M_P=1 TO 4;   /* MULTIPROGRAMMING LEVEL */
         CALL STPARM('DEG_M_P',DEG_M_P);   /* SETS PARAMETER DEG_M_P */
         CALL RESQM(0);   /* EXPANDS INNER MODEL & SOLVES */
         CALL GTRSLT('CPUQ','TP',OP);   /* GET THROUGHPUT FOR CPU */
         QRATES(DEG_M_P)=OP(1)/8.0;
      END;
      CLOSE FILE (RSQ2IP);
         /* OUTER MODEL */
      OPEN FILE (RSQ2IP) TITLE('OUTER');
      CALL READMD;   /* READS OUTER RQ2COMP FILE FROM TRANSLATOR */
      CALL STPRMV('QRATES',QRATES);   /* SETS PARAMETER QRATES */
      CALL RESQM(0);   /* EXPANDS OUTER MODEL & SOLVES  */
      CALL TYPEVL;
END INOUT;
```

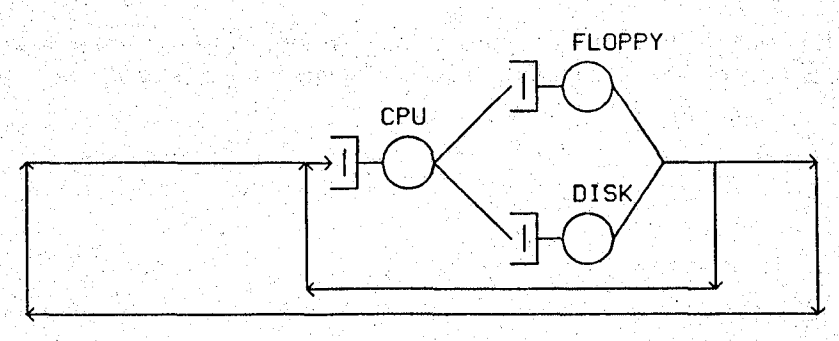


Figure 14.3 - Inner Model

We have two RESQ models to read, one for the inner model and one for the outer model. The inner model contains a parameter DEG__M__P representing the multiprogramming level. The outer model contains a parameter QRATES representing the rates for the queue dependent server. The variable QRATES will be used to store the results of the inner model and will be assigned to the parameter of the outer model. The inner model is read and is solved for DEG__M__P's of one, two, three and four. The CPU throughput for each DEG__M__P is divided by 8.0, the mean number of cycles, and stored in QRATES. The model definition file from SETUP is closed so that it can be opened with a different name, enabling the outer model to be read. The outer model parameter is assigned a value from QRATES, and the outer model is solved. Then the results are displayed interactively.

The following shows the execution of this program (after compilation and appropriate CMS commands):

```
EXECUTION BEGINS...
NO ERRORS DETECTED DURING NUMERICAL SOLUTION.

WHAT:all

ELEMENT     UTILIZATION
TERMQ       0.00000
CSQ         0.98158

ELEMENT     THROUGHPUT
TERMQ       2.27025
CSQ         2.27025

ELEMENT     MEAN QUEUE LENGTH
TERMQ       22.70251
CSQ         7.29747

ELEMENT     MEAN QUEUEING TIME
TERMQ       10.00000
CSQ         3.21439

WHAT:
R; T=2.26/4.40 11:08:17
```

A brief discussion is in order to emphasize the difference between solving the model in this fashion as opposed to using submodels. When submodels are used, the expansion program produces and solves one model. With the above hierarchical approach, the inner model is evaluated four times and results from it are passed to the outer model, which is solved

separately. This is done to allow an approximate solution, as in the above example. Similar approaches may be used with simulation, to reduce simulation run times.

BIBLIOGRAPHY

CHAN78a K.M. Chandy and R.T. Yeh (Editors), *Current Trends in Programming Methodology, Volume III: Software Modeling and Its Impact on Performance.* Prentice-Hall, Englewood Cliffs, New Jersey (1978).

CHAN78b K.M. Chandy and C.H. Sauer, "Approximate Methods for Analysis of Queueing Network Models of Computer Systems," *Computing Surveys 10,* 3 pp. 263-280 (September 1978).

CRAN77 M.A. Crane and A.J. Lemoine, *An Introduction to the Regenerative Method for Simulation Analysis,* Springer-Verlag, New York (1977).

HEID81 P. Heidelberger and P.D. Welch, "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations," *CACM 24* (April 1981) pp. 233-245.

IGLE80 D.L. Iglehart and G.S. Shedler, *Regenerative Simulation of Response Times in Networks of Queues,* Springer-Verlag (1980).

KLEI75 L. Kleinrock, *Queueing Systems Volume I: Theory,* Wiley, New York (1975).

KLEI76 L. Kleinrock, *Queueing Systems Volume II: Computer Applications,* Wiley (1976).

KOBA78 H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Addison-Wesley, Reading, Massachusetts (1978).

LAVE77 S.S. Lavenberg and C.H. Sauer, "Sequential Stopping Rules for the Regenerative Method of Simulation," *IBM J. of Research and Development 21,* (Nov. 1977) pp. 545-558.

LAVE82 S.S. Lavenberg (Editor), E.A. MacNair, H.M. Markowitz, C.H. Sauer, P.D. Welch and G.S. Shedler, *Computer Performance Modeling Handbook,* to appear, Academic Press, New York (1982).

SAUE79 C.H. Sauer and K.M. Chandy, "Approximate Solution of Queueing Models of Computer Systems," RC-7785, IBM Research, Yorktown Heights, N.Y. (July 1979). *Computer 13,* 4 (April 1980) pp. 25-32.

SAUE81 C.H. Sauer and K.M. Chandy, *Computer System Performance Modeling,* Prentice-Hall (1981).

SCHW77 M. Schwartz, *Computer-Communication Network Design and Analysis,* Prentice-Hall (1977).

WONG78 J.W. Wong, "Distribution of End-to-End Delay in Message-Switched Networks," *Computer Networks 2,* 1 (February 1978) pp. 44-49.

INDEX