Table of Contents

1. Hierarchical Model Development
   1.1 Model Development
2. Examples of System Models
   2.1 Example 1: Batch Server System
   2.2 Example 2: Batch Server System With Two Types of Printers
   2.3 Example 3: Interactive Server System
   2.4 Example 4: Interactive Server System With Revisions
   2.5 Example 5: Interactive System With Dual Processors
3. Active Resource Manager Heirarchy
   3.1 Description of Active Server Attributes
   3.2 Active Server Conceptual Models (levels 1 - 3)
4. Passive Resource Manager Heirarchy
   4.1 Description of Passive Resource Manager Attributes
   4.2 Passive Resource Manager Conceptual Models (levels 1 - 3)
5. Shared Memory Manager Heirarchy
   5.1 Description of Shared Memory Manager Attributes
   5.2 Shared Memory Manager Conceptual Models (levels 1 - 3)
6. Code for the Examples
   6.1 Notes on the Example Code
   6.2 Example Code and Results
1. Hierarchical Model Development

1.1 Model Development

A model is a simplified representation of a complex physical system. A model is constructed to gain a better understanding of a system by predicting its performance under a variety of different operating conditions. Most modelers adopt a top-down, iterative approach to system modeling. Initially, the modeler constructs a very simple model of the system to answer fundamental questions about basic alternatives. The initial model contains many simplifications either because details of the system remain undecided, or their inclusion in the model is considered unnecessary. To be useful, this model must be able to evaluate a large number of alternatives quickly and inexpensively. As decisions are made about the system organization, the high-level structure of the model stabilizes. However, each system component still has a very simple representation. In order to obtain more accurate results, the modeler refines the model iteratively. An iteration introduces additional complexity in the model by increasing the amount of detail in the representation of some components. During the implementation of the system, design estimates can often be replaced by measured values. The model is then re-evaluated to examine the impact of inaccuracies in the initial predictions. This process is continued until a complete model with the desired complexity is obtained or the system has been constructed. Once the system becomes operational, the model does not become useless. The model may be used to tune the system to its environment and to predict the impact of proposed changes in the system and its workload.

Due to the high resource requirements of constructing a model, sub-models may be developed concurrently by a group of modelers. Frequently, the same sub-model may be used to represent a number of components in various physical systems. As an example, an active server may be used to model the CPU of a computer system or the cashier at the check-out counter of a grocery store.

To illustrate the ideas presented in this section, we consider the construction of a simulation model of a supermarket. The purpose of the model is to estimate the average time taken by a customer to pass through the check-out counter as a function of the number of items he has purchased. In our model of the supermarket, a customer arrives at the check-out counter with an arbitrary number of items, waits in a queue associated with a cashier, pays the appropriate amount and leaves the store. We have chosen this everyday example because it captures many of the complex scheduling and capacity planning issues involved in the design of computer systems. To illustrate this complexity, consider some fairly typical behavior patterns: a customer waiting in a queue notices another queue being serviced faster, and decides to join the faster queue (customer reneging); the machine being used by a specific cashier may malfunction temporarily (server failure), or the cashier may close his queue if it becomes too long (finite queue capacity); a customer may desire to be serviced at a specific counter, which is not necessarily the
one with the shortest queue (customer affinity for one of many servers); a customer may request priority service (different service disciplines), etc. Rather than attempting to capture the entire complexity of the system, the modeler may initially choose a very simple model -- for instance, a simple FCFS server. In this model, a single server services its queue in a FCFS manner. When a customer needs service, he joins the queue, and waits until he has received the required service. The arrival of customers is modelled by a source entity, and a sink entity models their departure from the system. Since the FCFS server is a frequently used model, it may be selected from a library of models maintained by the modeler. The parameters of the general FCFS server are assigned appropriate values and the model executed. Based on the results of executing his initial model, the modeler may refine the model as follows: two types of service-centers are introduced -- express lanes to service customers with very small service requirements, and general lanes for the remaining customers. The refined model consists of two sub-models, each of which is a FCFS server. The arrival-rate of customers for each FCFS server should be adjusted to match the overall arrival-rate in the earlier model. The two sub-models may be developed concurrently by different modelers. The sub-model for the general lanes can be refined independently of the other sub-model by replacing the FCFS server by a collection of FCFS servers which are fed from a common source. Complex system behavior like customer reneging, server failures, finite queue capacity, and customer affinity may be introduced iteratively in the model. The sub-model for the express lanes may be refined independently to experiment with other service disciplines, like priority service based on number of items purchased by a customer (shortest job next).

There are four important aspects to the modeling process described above:

1. Reusability: A sub-model can be used to represent a variety of different components.

2. Evolution: In general, a model evolves over time from simple to complex. The representation of a system component may change depending on the amount of information available and on the perceived effect of the component on the performance parameter being measured. This implies that the system components may be modeled at increasing levels of complexity during the evolution of the model.

3. Cooperation: A group of modelers cooperate in the model development process.

4. Multiple Copies: Several versions of a model may exist simultaneously. Each version may include details needed to answer a particular question. It follows that different versions of a model may incorporate different levels of complexity in the representation of a given component.

The system described in this paper attempts to capture each of the above aspects
of the modelling process. Reusability has been an important criterion in the design of a number of modeling packages and led to the adoption of a tool-based approach to modeling. In this approach, the system provides a library of modelling tools called entities. Each entity in the library models a specific component. In addition, the library may also contain entities for statistics collection and report generation. This tool-based approach effectively captures the philosophy of top-down decomposition of a model into simpler sub-models. If each component in the system being modelled can be represented by an entity from the model library, the model may be constructed conveniently by selecting the appropriate entities from the library. However, the general tool-based approach does not provide any help in the evolution of a sub-model to introduce more complex behavior patterns. Typically, the model library contains a single entity to represent a group of physical components. This forces the modeler to design his model at the precise level of complexity that has been built into each entity in the library. As a consequence, if the library entity is too complex, the modeler is forced to deal with a large number of parameters that he may be unable to specify, or if it is too simple, use a simpler model than he needs. This has been a classic shortcoming of most library-based systems. We introduce the concept of hierarchical reusability as a solution to some of these problems. The next chapter presents a series of examples which illustrate how hierarchies may be used to develop models of simple computing systems. The following chapters describe three hierarchies for modeling: active-server hierarchy, resource-manager hierarchy, and a memory-manager hierarchy.

2. Examples of System Models

Introduction

This chapter illustrates the application of the hierarchical modeling process to construct a sophisticated model of a computer system. The model is developed iteratively. A simple model of the system is constructed initially. Subsequently, each iteration refines the model either by decomposing a sub-model into a collection of sub-models or by refining a sub-model to introduce complexity in its description.

2.1 Example 1: Batch Server System
As our first example, we choose a simple batch processing system. In the physical system, users submit their jobs to an operator who queues each job for execution by the computer. After a job has been executed, the output is sent to one of several identical line printers where it is printed and is subsequently picked up by the user.

We develop a simple model of the physical system described above. The source entity models the arrival of the batch jobs at the computer. This entity creates jobs at a specified rate which is equal to the average rate at which students submit their jobs to the operator.

The operator and the computer are modeled by a simple entity called *yasl2*. *yasl2* is the simplest entity from a family of entities known as the *active-server* family. This entity (at the first level of the hierarchy) services requests using a *fcfs* service discipline. The entity has a simple parameter called its service rate which specifies the fixed rate at which requests for service are processed.

To model access to a group of line printers, a resource entity, *izmfl*, is chosen from the *passive-resource-manager* family. This entity models exclusive access to one or more of the passive resources (line printers) that it manages. The entity is the simplest entity in this family and models access to a simple type of resource. The entity has a simple parameter *numtok* which specifies the initial number of units of the resource managed by the manager. Upon receipt of a request, the manager services the request if enough units of the resource are available. Otherwise the request is queued. A queued request is serviced in a *fcfs* fashion when sufficient units of the resource become available to the manager.

Finally the sink entity models the departure of jobs from the system (i.e. students picking up hard output).

A schematic diagram for the model described above follows:

```
  source  active server  passive resource manager  sink
```

A special entity called the *job* entity is defined to simulate the interactions between the user and the system. A *job* entity is created by the *source*. After having been created, a job first requests service from the computer and waits for acknowledgement that the service has been completed. It then requests and waits for access to a single line printer. After obtaining access, the job retains access to the printers for a time specified by the job. After its output has been printed, the job relinquishes access to the printers and terminates.

The statistical information generated by the execution of this model is 1) utiliza-
tion of the computer which is defined as the % of time the cpu was servicing requests; 2) utilization of the line printers which is the average number of line printers that were in use over the entire simulation period; and 3) the mean response time for requests which is defined as the average time for a job entity to receive service from the cpu entity and the resource entity. The first two statistical measures are automatically generated by the respective entities. The third statistic is incorporated in the description of the job entity.

2.2 Example 2: Batch Server System With Two Types of Printers

For our second example, we refine the batch processing system introduced in example one to introduce two types of printers. After a job has been executed by the computer, it may be directed to one of two types of printers - laser printers or line printers.

To model this system, the source, sink and computer components and their respective attributes remain as described in the previous example. The printer component and its attributes change as described below.

To model access to two different types of printers, an entity called *iztmf2* is chosen from the second level of the passive-resource-manager family. When the resource entity receives a request for access to the passive resources (printers), if enough units of the type of resource requested is available, the request is granted. Otherwise the request is placed on a time of arrival (First-Come-First-Serve) queue. In general, a request specifies the number of each type of resource managed by the entity.

A schematic diagram for this model follows:

![Diagram of batch server system with two types of printers](image)

The only change in the job entity is that it must now specify which type(s) of passive resources (printers) it requires. The job releases these resources after retaining access for a time specified in its description.

The statistics gathered remain the same as in the first example.
2.3 Example 3: Interactive Server System

In this example, we assume that the system being modelled is an interactive computer system. In the physical system, users at interactive terminals use a computer to create documents and then print them out on laser or line printers. The components of the system are: users of the computer, terminals, the computer, and the printer manager and the printers. Actions performed by the components are described below.

A user logs on to a terminal and sends commands to the computer in order to create a document. The user sends an arbitrary number of editing commands followed by a request to print the final document when he or she is finished. The computer executes the commands and informs the user when the execution is completed. When a computer receives a request to print a document, it forwards the request to the printer manager which controls access to the laser and line printers, ensuring that only one document is being printed on a given printer at any time.

The users of the system are classified as undergraduate students, graduate students, or professors. User commands are executed according to the priority of the user. Professors have the highest priority, then graduates, and then undergraduates. The priority of each user is determinable from the logon id of the user.

A simulation model of the physical system described above is constructed as follows. We assume that each user thinks for a while, then issues a single command to the computer from the terminal. The command requests service from the computer. The computer services the user’s request according to the user’s priority. The number of editing commands requested by a user (before the final print command) is determined according to a probability distribution. Each request of the user is preceded by a thinking phase before entering his or her next command. If the user is finished editing his or her document, then the last request made will be to have it printed on a laser or line printer. A request is sent, containing the user’s id, to the printer manager.

When the printer manager receives a printer request, it releases the type of printer needed to the computer so that the document can be printed. If a printer of the appropriate type isn’t available, the manager queues the request according to the user’s priority and services it when a printer is available. After the user’s document has been printed, the printer is released and the user leaves the system.

For the simulation model, entities are created which model both the components of the physical system (on a very simple level) and the actions taken by a user in editing and printing a document. The model consists of the following entities: the source which models arrivals; three entities to model the three hardware components and the document-creation actions to be modeled (i.e. the interactive terminals, the computer, and the printer manager for the laser and line printers); the users are modeled by an entity called the job entity. When the job departs from the system, it is said to have reached the sink (For this, as in previous examples, the sink does not need to be modeled as an actual entity).

The first entity to consider is the one needed to model the interactive terminals and the process of users thinking and entering commands. Two points need to be con-
sidered in choosing this entity. First, the thinking time of the user must be simulated. Second, the users are able to log on to an available terminal and immediately receive service. For the purpose of this example, the number of users in the system is never greater than the number of available terminals. A delay service center from level 2 of the active server hierarchy, izasdy, will be chosen. The thinking time of the users can be easily simulated as a service request to the delay server. Further, a delay server allows a user to receive immediate service without waiting, no matter how many users are already being serviced. The statistics that will be gathered by the delay server will be: the minimum, maximum and mean think times, the utilization of the server, and the server throughput.

The second entity is used to simulate the computer. In this case, the computer services one user at a time using a priority service discipline. Another active server entity, called izasl2 from level two of the active server hierarchy can be used to simulate the computer. This entity has two parameters, the service rate which is the rate, in service units per simulation unit, that service requests are satisfied; and the service discipline which for this example has been specified as a priority service discipline. The commands received by the computer can be simulated as requests for service from the user to the izasl2 entity. The statistics that will be gathered by the izasl2 server will be: the minimum, maximum and mean response and service times, the utilization of the server, the maximum and mean queue length, and the server throughput.

The third entity simulates the printer manager and the printers. As in example two, a level two passive resource manager, iztml2, can be used, except that the queueing discipline will be specified as priority.

A diagram and complete description of the simulation model just constructed follows:

```
resources

source delay server (level 2) active server (level 2) passive resource manager (level 2) passive resource manager (level 2) sink
```

Finally, the job entity description. The source creates jobs according to the in-
terarrival rate. Each job requests service from the delay server corresponding to the average 'think time' of the user. Upon completion of the service from the delay server, a service request is sent to izasl2. The request is queued according to job priority. Upon completion of the service request, the job either requires more service and again requests service from the delay server followed by a request to izasl2, or a resource request containing the resource type and the job's priority is sent to the passive resource manager. The probability of a job returning to the delay server is \( p_1 \) and the probability of going on to izasl2 is \( p_2 \). If a resource of the correct type can be allocated, it is; otherwise the request is queued. The passive resource manager allocates a resource to the job which the job keeps for a time period simulating resource usage, then returns the resource to the passive resource manager. Finally, the job proceeds to the sink and terminates. At this point the response time statistics for the job are gathered.

### 2.4 Example 4: Interactive System with Revisions

For this example, we refine the model for the interactive system introduced in example 3. The refinements introduced in this example model a computer that services requests using a pre-emptive priority service discipline. The queue length is controlled to limit the number of requests in the queue. A final refinement is that the printer manager now schedules requests for access to the passive resources (printers) using a priority discipline.

In the pre-emptive priority discipline, a server services its queue according to the priority of the request. If a request with a higher priority is received by the server while it is servicing a lower priority request, the request being serviced is pre-empted and the higher priority request is serviced. The pre-empted request is put on the priority queue of the server.

The components or aspects of this system which change from the previous example are the following: the users and the terminals (source); the computer's associated queue of requests which is maintained in a pre-emptive priority scheduling discipline; the queue capacity (maximum number of requests in the queue) is now specifiable; and access to the printers which is given according to a priority discipline. The following components or aspects remain as described in example 3: the computer; the routing (back to terminals or to printer); and the sink.

A schematic diagram of the model follows:
To model the computer with queue capacity and which schedules requests using a pre-emptive priority discipline, a server called izarl3 is chosen from the third level of the active-server family. The statistics gathered by this entity are: the minimum, maximum and mean response times; the utilization of the server; the minimum and maximum queue length; and the server throughput.

An entity izzml2 is chosen from the second level of the passive-resource-manager family to model the printer manager and the queues associated with each type of printer. At this level of the hierarchy, the desired priority scheduling discipline may be specified. The statistics gathered by this entity are the same as for example two.

The description of the source entity changes slightly to accommodate the capability to specify the queue capacity. This change is effectuated by the specification of the entity parameter, quecap, when instantiating the active server, izarl3. The job entity must specify the priority of the requests to both the active and passive servers.

2.5 Example 5: Interactive System With Dual Processor

For this example, there are two significant additions to the system to be modeled. First, the single processor computer is replaced by a dual-processor computer. The order in which service requests are executed remains the same i.e. pre-emptive priority, however there are now two processors servicing requests from a common queue. Second, user documents are retrieved from and saved on disk drives. Two disks are available, both of equal service rates; the CPU accesses the disks to update and store documents. The components of the system are now: users of the computer, terminals, the dual-processor computer, the two FCFS disk drives and the printer manager and the printers.

The entity needed to model this dual-processor computer must have two identical servers to simulate the two processors, the ability to set the queue capacity, and a pre-emptive priority queueing discipline. Again, a level three active server entity, izarl3, is
chosen from the active server hierarchy. The number of servers parameter is set to two. The statistics that will be gathered by izasl3 will be: the minimum, maximum and mean response and service times, the utilization of the server, the maximum and mean queue length, and the server throughput.

The disks will each be modeled by a level 2 active server entity with fcfs service discipline.

A diagram and description of the job entity follows:

The principle change in the job entity from previous examples is that it must define the routing of a transaction to the several possible active servers and the passive resource manager. In order to do this, branching probabilities, \( p_1, p_2, p_3, p_4 \) are defined to determine whether the job proceeds to the level two active servers, the passive resource manager or back to the delay server, respectively. As in the previous examples, the response time statistics gathered by the job entity are a measure of the length of time from the job's creation to its departure from the system.

3. Active Resource Manager Heirarchy

3.1 Description of Active Server Attributes
The active server hierarchy is based upon the increasing capability of the service center to model more diverse and complex servers. The capabilities of the service center at each level of the hierarchy are indicated by attributes and the range of attribute values available to the user at each level.

The simplest attributes of a server, available at level 1, are service rate and service discipline. Service rate determines the fixed rate at which customer requests for service are satisfied. This rate is assumed to be expressed in number of service units per unit of simulation time. For example a service rate of 10000 means that each customer will receive 10000 units of service for each unit of simulation time. The service discipline refers to the order in which requests for service are handled by a service center. The two types available at level 1 are first-come-first-served and priority.

Level 2 of the active server hierarchy extends the scope of service disciplines to include the delay or infinite server. When using this discipline, a customer never waits for service. Level 2 introduces two new attributes, server power and queue status. Server power can be used to vary the service rate of a server. At this level, the power may be set on or off. Queue status allows a server to dynamically close and open its queue. When queue status is off, the server will not accept any further requests.

At level 3, two new service disciplines, and three new attributes, including multiple servers become available to the user. Level 3 introduces two new service disciplines, last-come-first-served, preemptive-resume and preemptive priority. For lcfr: if, when a customer arrives at the manager with a service request, all servers are busy, then the customer that has been serviced the longest is pre-empted and the new customer is allocated that server. For preemptive priority, if a request entering the server's queue has a higher priority than a current customer, then the higher priority request may preempt the lower.

Level 3 also introduces a user-specifiable queue capacity which limits the number of customers that can join the server queue. When the queue capacity is reached, customers can no longer join the queue. In addition, switching overhead to simulate switching service between customers is available at level 3. Finally, at level 3 the number of servers available at the service center becomes user specifiable. Once specified, the number of servers at the center cannot be changed, although specific servers may be turned on or off using the server power attribute, which can now be used to separately modify the power of each server.

Level 4 adds two new service disciplines, round robin and processor sharing. For round robin, each entering customer joins the queue, which is treated as if it were circular. Service is provided to each requestor by going around the queue, giving each a fixed quantum of service. For processor sharing, a variable interval of service time is shared evenly among all requestors currently in the server queue.

Level 5 introduces a modifiable service rate attribute, and two new attributes, server priority and queue inspection by requestors. At level 5, the service rate attribute becomes dynamically modifiable, relative always to the original, user-specified, service rate. When multiple servers are available at a service center, a different service rate can be initially specified and separately modified for each server. Multiple servers at a service center can be assigned priorities for handling incoming requests at level 5, using the new server priority attribute. Server priority indicates which server(s) will handle an incom-
ing request first when more than one server is on and idle. The second new attribute is queue inspection, which allows a customer to inspect the queue and balk (choose not to request service) if, for example, the queue is too long.

Level 6 increases the queue-status-dependent capabilities of a customer by allowing a queued requestor to reneg and leave the queue. Level 6 also allows a customer to express affinity for a particular server(s) the customer would prefer to be serviced by.

At level 7, the basic server attributes of service discipline and service rate become user-specifiable algorithms.

3.2 Active Server Conceptual Models (levels 1 - 3)

3.2.1 Level One

(single server, infinite queue)

A customer arrives at the service center needing a certain number of units of service. The customer immediately joins the queue, and while in the queue receives the service. The customer then departs the service center. Service is provided to one customer at a time by a single server. The server provides service at a fixed rate. The service discipline determines the order in which customers are served.

**Active server center attributes:**

- **service rate**
  
  integer $\geq 0$  
  (service units/time unit)  
  static

- **service discipline**
  
  {fcfs,priority}  
  fcfs  
  static

3.2.2 Level Two

(single server, infinite queue, dynamic server power and queue status)
A customer arrives at the service center needing a certain number of units of service. The customer immediately joins the queue, and while in the queue receives the service. The customer then departs the service center. Service is provided to one customer at a time by a single server. The server provides service at a fixed rate. The service discipline determines the order in which customers are served. Both the queue and the server may be dynamically made inoperable. Customers receive no service if the server is off, although customers can continue to join the queue if it is open. Customers are denied entry to a closed queue, although customers already in the queue continue to receive service if the server is on.

Active server center attributes:

* service rate
  integer >= 0 1 (service units/time unit)

* service discipline
  \{fcfs,priority\} fcfs static

* server power
  \{on,off\} on dynamic

* queue status
  \{open,closed\} open dynamic

3.2.3 Level Two

(delay)

Same as the other service centers at this level, except that service is provided to
each customer as if it were the only customer at the service center. A customer never waits while other customers are being served.

Active server center attributes:

* service rate  
  integer \( \geq 0 \)  
  \( 1 \)  
  (service units/time unit)  
  static

* service discipline  
  \{delay\}  
  static

* server power  
  \{on, off\}  
  on  
  dynamic

3.2.4 Level Three

(multiple identical servers, optional finite queue capacity)

Queue Servers

Service may be provided to customers by multiple identical servers. All servers operate at the same fixed rate, although each server may be independently turned on or off. Additional service disciplines are available. The maximum number of customers that can wait in the queue can be specified by setting the queue capacity. Queue status and queue capacity are independent. A queue of any capacity can be closed at any length.

Active server center attributes:

* service rate  
  integer \( \geq 0 \)  
  \( 1 \)  
  (service units/time unit)  
  static

* service discipline  
  \{fcfs, priority\}  
  fcfs  
  static

* new service disciplines  
  \{lcfs preempt-resume, preemptive priority\}  
  static
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Setting</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>server power</td>
<td>{on, off}</td>
<td>on</td>
<td>dynamic</td>
</tr>
<tr>
<td>queue status</td>
<td>{open, closed}</td>
<td>open</td>
<td>dynamic</td>
</tr>
<tr>
<td>number of servers</td>
<td>integer &gt;= 1</td>
<td>1</td>
<td>static</td>
</tr>
<tr>
<td>queue capacity</td>
<td>integer &gt;= 1</td>
<td>infinite</td>
<td>static</td>
</tr>
</tbody>
</table>

4. Passive Resource Manager Heirarchy

4.1 Description of Passive Resource Manager Attributes

A passive resource center manages a pool of resources available to requesting customers. The attributes of a resource manager are resource types and the resource allocation discipline. Resource types refers to the number of different types of resources managed by a resource manager. The resource allocation discipline refers to the manner in which incoming requests for resources are serviced.

At level 1, one resource type is available, and the initial size of the resource pool is specifiable by the user. The allocation disciplines that can be used are first-come-first-served, priority, and skipping. The skipping discipline allows satisfiable requests queued behind unsatisfiable requests to be allocated resources.

At level 2, the number of resource types becomes specifiable by the user, each type has its own pool, and the initial size of each pool can be set by the user. Additionally, each requestor now includes requests for each type in its resource request.

At level 3, the requestor is allowed to include alternative sets of requests for resources of each type.

4.2 Passive Resource Manager Conceptual Models
4.2.1 Level One

(single resource type, single user request)

A customer requiring passive resources arrives at the passive resource manager. The requestor enters the manager's queue, and is serviced according to the manager's allocation discipline: fcfs, priority or skipping. On acquiring the required number of resources, the customer leaves the manager. The customer may subsequently return to the manager and release resources back to the manager without entering the manager's queue.

Passive resource center attributes:

* initial pool of resources integer >= 0 infinite static
* queue type {fcfs,priority} fcfs static
* user request type integer variable dynamic

4.2.2 Level Two

(multiple resource types, multiple-type user request)

A customer requiring passive resources arrives at the passive resource manager. The requestor enters the manager's queue, and is serviced according to the manager's al-
location discipline: fcfs, priority or skipping. The passive resource manager has more than one type of resource available, and a customer indicates in its request how many resources of each type is needed. If enough resources of each type are available in the resource pools, then the customer is allocated the requested number of resources of each type. On acquiring the required number of resources, the customer leaves the manager. The customer may subsequently return to the manager and release resources back to the manager without having to enter the manager's queue. When a customer releases resources back to the resource manager, the number of resources that are being returned of each type are specified.

Passive resource center attributes:

* initial pool of resources
  - vector of integer \(\geq 0\)
  - infinite
  - static

* queue type
  - \{fcfs,priority, skipping\}
  - fcfs
  - static

* number of resource types
  - integer \(\geq 1\)
  - 1
  - static

* user request type
  - vector of integer \(\geq 0\)
  - dynamic

4.2.3 Level Three

(multiple resource types, alternative multiple-type user requests)

A customer requiring passive resources arrives at the passive resource manager. The requestor enters the manager’s queue, and is serviced according to the manager’s allocation discipline: fcfs, priority or skipping. The passive resource manager has more than one type of resource available, and a customer needing resources of these types presents the resource manager with a request that contains alternative sets of requests.
Each set contains a single request value for each resource type available. If enough resources of each type are available in the resource pools to satisfy one of the sets of requests, then the customer is allocated the requested number of resources of each type. On acquiring the required number of resources, the customer leaves the manager. The customer may subsequently return to the manager and release resources back to the manager without having to re-enter the manager's queue. When a customer releases resources back to the resource manager, the number of resources that are being returned of each type are specified.

Passive resource center attributes:

* initial pool of resources
  - vector of integer >= 0
  - infinite
  - static

* queue type
  - \{fcfs, priority, skipping\}
  - fcfs
  - static

* number of resource types
  - integer >= 1
  - 1
  - static

* user request type
  - matrix of alternative vectors of integer >= 0
  - dynamic

5. Shared Memory Manager Heirarchy

5.1 Description of Memory Manager Attributes

The memory manager manages access to blocks of simulation memory. Two types of blocks are managed: 1) Free blocks which are available for data storage and 2) data blocks which contain previously stored data. The memory manager receives requests to allocate free blocks, allocate data blocks, or release a data block.

At the simplest level of the heirarchy(level 1), the only attribute that is specifiable
by the user is memory size.

At level 2, the user is given the additional capability of specifying the service discipline for the manager. The service discipline refers to the order in which requests for data and free blocks are serviced. Although the requests for free blocks and data blocks are kept in separate queues, the discipline specified applies to both.

In addition to the attributes at the first two levels, at level 3, the user may specify the memory allocation strategy. This attribute determines the method by which free blocks of simulation memory is allocated. The methods provided are first-fit and best-fit. For the first-fit strategy, the first block available which satisfies the request is returned. For the best-fit strategy, the request is filled by the smallest block large enough to satisfy the request. This implies that the list of free blocks is ordered on size.

5.2 Shared Memory Heirarchy Conceptual Models:

5.2.1 Level One

request \( \xrightarrow{\text{Free}} \) \( \square \) \( \xrightarrow{\text{release Data}} \) \( \square \) \( \xrightarrow{\text{free Data}} \) \( \square \)

A customer requesting either a Free block or Data block immediately joins a queue and is serviced (or terminated) in finite time. The customer leaves the manager upon receiving the desired amount of memory.

Memory Manager Attributes:

* memory size \( \text{integer} \geq 1 \) static

5.2.2 Level Two
A customer request for Data or Free blocks is serviced according to the specified queueing discipline - either fcfs or priority.

Memory Manager Attributes:

* memory size                   integer $\geq 1$         static
* service discipline             \{fcfs,priority\}       fcfs   static

5.2.3 Level Three

The strategy for the allocation of simulation memory may also be initially specified.

Memory Manager Attributes:

* memory size                   integer $\geq 1$         static
* service discipline             \{fcfs,priority\}       fcfs   static
* allocation strategy           \{bestfit,firstfit\}     bestfit static
6. Code For The Examples

6.1 Notes on the Example Code

6.1.1 Functions used in the example simulation programs

The following functions are used by the example programs:

\textit{expon(mean)}: given a \textit{mean} value of type integer, generates a random number according to the exponential probability distribution. The value returned is of type real.

\textit{prbgen(array,size)}: given an array of cumulative probabilities and the size of the array, returns an index to one of the cumulative probability values. \textit{Array} should be of type real and \textit{size} of type integer. The returned index is of type integer. The intrinsic function \text{rand}() is used.

\textit{cstore(cvar1,cvar2)}: copies the value of \textit{cvar2} into \textit{cvar1}. Both variables should be of type clocktype.

\textit{csub(cvar1,cvar2)}: given two variables of clocktype, returns the value of \textit{cvar1} minus \textit{cvar2}. The return value is of type integer.

6.1.2 Entity family reserved words used in the examples

The following reserved words are used by the example simulation programs. They are located in \texttt{common.sim}. For a description of the MAY reserved words, please consult the MAY User Manual.

\textit{fcfs} : first-come-first-served service discipline flag.
\textit{priiry} : priority service discipline flag.
\textit{prepri} : pre-emptive priority service discipline flag.
6.2 Code and Results
6.2.1 Code for Example One
**ENTITY SOURCE**

FUNCTION:
This is the source entity which creates transactions according to a specified interarrival rate.
Transactions are created until SOURCE is terminated.

PARAMETERS:
- CPU: cpu active server entity id.
- PRNMGR: printer manager entity id.
- STAT1: statistics entity id.

MESSAGES:
- TRMNT: Message to terminate the entity-instance.

```c
entity source(cpu,prnmgr,stat1)
local integer jobcnt,intarr
integer trans
real exponent
message trmnt

set interarrival rate
intarr = 3000
jobcnt = 0

create transactions until terminated by main
continue
let trans be job(cpu,prnmgr,stat1)
jobcnt = jobcnt + 1
wait int(exponent(intarr))
if (msg .ne. trmnt) go to 10

write(6,9000)jobcnt
9000 format(""Source Termination.",
+ 
+ ,"Number of transactions created by source = ",i6)
+ 

ende
```

**ENTITY JOB**

FUNCTION:
This is the transaction entity.

PARAMETERS:
- CPU: cpu active server entity id.
- PRNMGR: printer manager entity id.
- STAT1: statistics entity id.

MESSAGES:
- RQCOMP: Request for service completed.
- HAVTOR: Have been allocated resource requested.

```c
entity job(cpu,prnmgr,stat1)
integer cpumnp,prmmn,numres,csub
```
mean service times for cpu & printer, number of resources needed
parameter (cpum = 50, prmn = 4000, numres = 1)
real expon
clocktype begclk

message rqcomp
message havtok

trace 3 when .true.
record starting time token for response stat
call cstore(begclk,clock)

request service from computer
invoke cpu with request (myid, int(expon(cpum)))

wait for service complete message
wait maxint for (msg .eq. rqcomp)

request printer
invoke prnmgr with regtok(myid,numres)
wait to receive resource
wait maxint for (msg .eq. havtok)
simulate printer usage
wait int(expon(prmn))
return printer to manager
invoke prnmgr with reltok(myid,numres)

collect service stat
invoke stat1 with insert(csub(clock,begclk))
999 continue
ende

*********************************************************************
ENTITY MAIN
*********************************************************************

This is entity main which creates the active servers, passive
resource manager, statistics entity, and source entity.

entity main

local integer cpu, prnmgr, stat1
integer svrte, initrs, title(26), titlen
service rate 1 sim unit = 1 msec, initial resources = 2
parameter (svrte = 1, initrs = 2, titlen = 26)
include 'common.sim'

message

trace 3 when .true.
write(6, 9000)
9000 format (/,"Start of Simulation."/)
set izstim to simulation run length
simulation period 500000
izstim = 500000

set up statistics entity for job response time
let stat1 be mstat(title, title)
create cpu activer server and printer resource manager
let cpu be izzs12(fcfs, svrte)
let prnmgr be izzf1(initrs)
create source
let src be source(cpu,prnmgr,stat1)
run for simulation period
wait izstimm

dump statistics
dump(stat1)
dump(cpu)
dump(prnmgr)

terminate cpu,prnmgr,source,stat1
terminate(cpu)
terminate(prnmgr)

terminate(stat1)

terminate(trmnt)
terminate(stat1)

terminate(trmnt)
write(6,9001)
format(6,"Simulation has finished.")
endc
append informs MAY compiler of needed entities
append izas12,iztmf1,mnstat
6.2.1 Results for Example One
start of simulation.

simulation has finished.

statistical dump at simulation time 500000.

transaction response times
the total of all values is 838267.
the total no. of values is 158
the average value is 5305.49

statistical dump at simulation time : 500000

System statistics for active server

Server type = fcfs

Service rate = 1
Number of servers = 1

response time
min  max  mean  number
1    236  53.019  160

service time
min  max  mean  number
1    236  52.763  160

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean  max  current
0.017  2    0

System statistics :

utilization  thruput  jobs completed  jobs pre-empted
0.017  0.000  160    0

statistical dump at simulation time : 500000

System response time stats for token requests:

Initial number of tokens: 2

minimum response time : 0
maximum response time : 11496
number of responses : 160
mean response time : 1484

System throughput : 0.00032
mean queue length : 0.47514
mean available tokens (all types) : 0.80533
jobs still in queue: 0

source termination.
number of transactions create by source = 160
Simulation terminated normally
Simulation period specified: 500000.
6.2.1 Code for Example Two
FUNCTION:
This is the source entity which creates transactions according to a specified interarrival rate.
Transactions are created until SOURCE is terminated.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
STAT1: statistics entity id.

MESSAGES:
TRMNT: Message to terminate the entity-instance.

entity source(cpu,prnmgr,stat1)
local integer jobcnt,intarr
integer trans
real expn
message trmnt
set interarrival rate
intarr = 3000
jobcnt = 0

create transactions from job entity
continue
let trans be job(cpu,prnmgr,stat1)
jobcnt = jobcnt + 1
wait int(expon(intarr))
if (msg .ne. trmnt) go to 10

write(6,9000) jobcnt
format(,,"Source Termination.",
+ ,,"Number of transactions created by source = ",i6)
ende

FUNCTION:
This is the transaction entity.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
STAT1: statistics entity id.

MESSAGES:
REQUEST: Request for service completed.
HAVETOK: Have been allocated resource requested.

entity job(cpu,prnmgr,stat1)
real x,lasprb,expon
integer cpmn,prnmn,linepr,lasprn,prbgen,csub
mean service times for cpu & printers, resource array indices,
laser printer probability
parameter(cpumn=50, prnml=4000, linepr=1, lasprn=2, lasprb=0.20)
local integer prntyp(2)
clocktype begclck

message rqcomp
message havtok

trace statement
trace 3 when .true.

record starting time token for response stat
call cstore(begclck,clock)

request service from computer
invoke cpu with reqest(myid,int(expon(cpumn)))
wait for service complete message
wait maxint for (msg .eq. rqcomp)

request printer
decide upon printer to use according to probability of laser
set resource array slots to number of each type of resource needed
prntyp(linepr) = 0
prntyp(lasprn) = int(rand(x)+lasprb)
if (prntyp(lasprn) .eq. 0 ) prntyp(linepr) = 1
invoke prnmgm with reqtok(myid,prntyp[2])
wait to acquire resource
wait maxint for (msg .eq. havtok)
simulate printer usage
wait int(expon(prnml))
return printer to manager
invoke prnmgm with reitok(myid,prntyp[2])
collect service stat
invoke stt1 with insert(csub(clock,begclck))

999 continue
ende

******************************************************************************

ENTITY MAIN

This is entity main which creates the active servers, passive
resource manager, statistics entity, and source entity.

******************************************************************************

entity main

integer initrs,nmres,rors(2),srvrte,title(26),titen
set initial resource, service rate, number of resources
parameter(initrs = 2,srvrte = 1,nmres = 2,titen=26)
local integer cpu,prnmgm,statl,src
clocktype begclck
include 'common.slim'
data title/"T',"R',"A',"N',"S',"A',"C',"T',"I',"O',"N',"'",
"R',"B',"S',"P',"O',"N',"S',"E',"'","T',"I',"M',"E',"S'/

message
trace 3 when .true.
write(6,9000)
9000 format(/,"Start of simulation.",/)
set simulation'period and izstim
simulation period 500000
izstim = 500000

* initialize initial-resources array for resource manager
do 10, i=1,nmres
   resors(i) = initres
10 continue

* set up statistics entity for response time collection
let stat1 be mnstat(titlen,title)

* create cpu active server and printer resource manager
let cpu be izasl2(fcfs,srvrte)
let prnmg be iztmf2(nmres,resors)

* create jobs
let src be source(cpu,prnmg,stat1)

* run for simulation period
wait izstim

* dump statistics
invoke stat1 with dump(6)
invoke cpu with dump(6)
invoke prnmg with dump(6)

* terminate cpu,prnmg,stat1,src
invoke cpu with trmnt
invoke prnmg with trmnt
invoke src with trmnt
invoke stat1 with trmnt

write(6,9001)
format('/',"Simulation has finished.")
ende

* append informs MAY compiler of needed entities
append izasl2,iztmf2,mmnstat
6.2.1 Results for Example Two
start of simulation.

simulation has finished.

statistical dump at simulation time 500000.

transaction response times
the total of all values is 1019225.
the total no. of values is 175
the average value is 5824.14

statistical dump at simulation time : 500000

System statistics for active server

Server type = fcfs

Service rate = 1
Number of servers = 1

<table>
<thead>
<tr>
<th>response time</th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>260</td>
<td>48.497</td>
<td>177</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>service time</th>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>260</td>
<td>48.497</td>
<td>177</td>
</tr>
</tbody>
</table>

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean max current

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>max</th>
<th>current</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.017</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

System statistics :

utilization thruput jobs completed jobs pre-empted

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.017</td>
<td>0.000</td>
<td>177</td>
<td>0</td>
</tr>
</tbody>
</table>

statistical dump at simulation time : 500000

System response time stats for token requests:

Token type: 1 initial tokens: 2

Token type: 2 initial tokens: 2

minimum response time : 0
maximum response time : 9226
number of responses : 177
mean response time : 1514

Token type: 1 mean available tokens: 0
Token type: 2 mean available tokens: 1

System throughput: 0.00035
mean queue length: 0.53605
mean available tokens (all types): 2.51000

jobs still in queue: 0

source termination.
number of transactions created by source = 177
Simulation terminated normally
Simulation period specified: 500000.
6.2.1 Code for Example Three
ENTITY SOURCE

FUNCTION:
This is the source entity which creates transactions according to a specified interarrival rate.
Transactions are created until SOURCE is terminated.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
TTY: tty active server entity id.
STAT1: statistics entity id.

MESSAGES:
TRMNT: Message to terminate the entity-instance.

entity source(cpu,prnmgr,tty,stat1)
local integer jobcnt,intarr
integer trans
real expn

message trmnt

c* trace 3 when .true.
c* set interarrival rate
intarr = 4000
jobcnt = 0

create transactions until terminated by main
10 continue
let trans be job(cpu,prnmgr,tty,stat1)
jobcnt = jobcnt + 1
wait int(expn(intarr))
if (msg .ne. trmnt) go to 10

9000 write(6,9000)jobcnt
format(//,"Source Termination.",
+/"Number of jobs create by source = ",i6)
ende

ENTITY JOB

FUNCTION:
This is the transaction entity.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
STAT1: statistics entity id.

MESSAGES:
REQCOM: Request for service completed.
HAVTOK: Have been allocated resource requested.
entity job(cpu,prnmgr,tty,stat1)

real x,prob,lasprb,ttyprb,priarr(3),expon
integer cpumn,prnmn,linepr,lasprn,thnkmn,arrsze,csub,prbgen
printer indices for resource request array, laser probability,
size of job priority array
parameter (linepr = 1, lasprn = 2, lasprb = 0.20, arrsze = 3)
means for cpu, printer service times, tty probability and think mean
parameter(cpumn = 50, prnmn = 4000, ttyprb = 0.80, thnkmn = 5000)
local integer prntyp(2), priort
clocktype begclk

c* cumulative priority probabilities for prof, grad, undergrad
data priarr/0.20, 0.50, 1.0/

message rqcomp
message havtok

c* trace 3 when .true.
record starting time token for response stat
call cstore(begclk,clock)

c* set priority of transaction - ungrad = 3, grad = 2, prof = 1
priort = prbgen(priarr, arrsze)

continue

terminal think time
invoke tty with reqest(myid,int(expon(thnkmn)))
wait maxint for (msg .eq. rqcomp)

continue

typrb service from computer
invoke cpu with reqest(myid, priort,int(expon(cpumn)))

wait for service complete message
wait maxint for (msg .eq. rqcomp)

branch back to tty or go on to printer
if (rand(x) .le. ttyprb) go to 10

request printer
decide upon printer to use according to probability of laser
prntyp(linepr) = 0
prntyp(lasprn) = int(rand(x)+lasprb)
if(prntyp(lasprn) .eq. 0) prntyp(linepr) = 1
invoke prnmgr with reqst(myid, priort, prntyp[2])
wait maxint for (msg .eq. havtok)

simulate printer usage
wait int(expon(prnmn))
return printer to manager
invoke prnmgr with reitok(myid, prntyp[2])

collect service stat
invoke stat1 with insert(csub(clock,begclk))

continue
ende

-----------------------------------------------

ENTITY MAIN

This is entity main which creates the active servers, passive
resource manager, statistics entity, and source entity.

-----------------------------------------------
entity main

local integer cpu, tty, prnmgr, statl, src
integer initrs, srvrte, numres, resors(2), title(26), titlen
set initial resources, service rate, number of resources
parameter(initrs = 2, srvrte = 1, numres = 2, titlen=26)
include 'common.sim'
message

trace 3 when .true.
write(6, 9000)
9000 format(1, 'Start of Simulation.
')
c* set length of simulation run
simulation period 3000000
izstim = 1000000

initialize passive manager - printer - resources
do 10, i=1, numres
   resors(i) = initrs
10 continue

set up statistics entity for job response times
let statl be mnstat(title, title)
c* create tty
let tty be izasdy(srvrte)
c* create cpu active server and printer resource manager
let cpu be izas12(priyty, srvrte)
let prnmgr be iztml2(numres, resors, priyty)
c* create jobs
let src be source(cpu, prnmgr, tty, statl)
c* run for simulation period
wait izstim
c* dump statistics
invoke statl with dump(6)
invoke cpu with dump(6)
invoke prnmgr with dump(6)
invoke tty with dump(6)
c* terminate
invoke cpu with trmnt
invoke prnmgr with trmnt
invoke tty with trmnt
invoke src with trmnt
invoke statl with trmnt
write(6, 9001)
9001 format(1, 'Simulation has finished.
')
ende

c* append informs MAY compiler of needed entities
append izasdy, izas12, iztml2, mnstat
6.2.1 Results for Example Three
start of simulation.

simulation has finished.

statistical dump at simulation time 1000001.

transaction response times
the total of all values is 7072408.
the total no. of values is 264
the average value is 26789.42

statistical dump at simulation time : 1000001

System statistics for active server

Server type = priority
Service rate = 1
Number of servers = 1

response time

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>492</td>
<td>52.863</td>
<td>1259</td>
</tr>
</tbody>
</table>

service time

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>347</td>
<td>49.718</td>
<td>1259</td>
</tr>
</tbody>
</table>

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting

mean max current
0.067 3 0

System statistics :

utilization thruput jobs completed jobs pre-empted
0.063 0.001 1259 0

statistical dump at simulation time : 1000001

System response time stats for token requests:

Token type: 1 initial tokens: 2

Token type: 2 initial tokens: 2

minimum response time : 0
maximum response time : 6309
number of responses : 264
mean response time : 322

Token type: 1 mean available tokens: 1
Token type: 2 mean available tokens: 1
System throughput: 0.00026
mean queue length: 0.08523
mean available tokens (all types): 3.04110
jobs still in queue: 0
statistical dump at simulation time: 1000001

System statistics for active server

Server type = delay
service rate = 1
maximum number of servers = 20

response time
min | max | mean | number
4   | 38944 | 4933.894 | 1269

service time
min | max | mean | number
4   | 38944 | 4933.894 | 1269

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean | max | current
6.207 | 13 | 10

System statistics:
utilization | thruput | jobs completed | jobs pre-empted
0.050 | 0.001 | 1269 | 0

source termination.
number of jobs create by source = 274
Simulation terminated normally
Simulation period specified: 3000000.
6.2.1 Code for Example Four
FUNCTION:
This is the source entity which creates transactions according to a specified interarrival rate. Transactions are created until SOURCE is terminated.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
TTY: tty active server entity id.
STAT1: statistics entity id.

MESSAGES:
TRMNT: Message to terminate the entity-instance.

entity source(cpu,prnmgr,tty,stat1)
local integer jobcnt,intarr
integer trans
real expos
message trmnt

set interarrival rate
intarr = 3500
jobcnt = 0

create transactions
10 continue
let trans be job(cpu,prnmgr,tty,stat1)
jobcnt = jobcnt + 1
wait int(expon(intarr))
if (msg .ne. trmnt) go to 10

write(6,9000) jobcnt
format(//,"Source Terminated.",
+ 
/,"Number of transactions created by source = ",i6)
ende

FUNCTION:
This is the job entity.

PARAMETERS:
CPU: cpu active server entity id.
PRNMGR: printer manager entity id.
TTY: tty active server entity id.
STAT1: statistics entity id.

MESSAGES:
RQCOMP: Request for service completed.
QLCLOS: Queue closed, cannot request service.
HAVTOK: Have been allocated resource requested.
entity job(cpu, prnmg, tty, stat1)

real x, prob, lasprb, ttyprb, priarr(3), expon
integer cpu, prnm, linepr, laspr, thnkmn, arrsze, prbgen, csub
resource request array indices, laser probability
parameter (linepr = 1, laspr = 2, lasprb = 0.20, arrsze = 3)
means for cpu, printers, think time at tty, tty probability
parameter (cpu = 50, prnm = 4000, ttyprb = 0.80, thnkmn = 5000)
local integer prntyp(2), priort
clocktype begclk
c* priority probability array
data priarr/0.20,0.50,1.0/

message rqcomp
message qclose
message havtok

c* trace 3 when .true.
c* record starting time token for response stat
call cstore(begclk,clock)
c* set priority of transaction - ungrad = 3, grad = 2, prof = 1
c* priort = prbgen(priarr,arrsze)

10 continue
c* terminal think time
invoke tty with reqest(myid,int(expon(thnkmn)))
wait maxint for (msg .eq. rqcomp)

20 continue
c* request service from computer
invoke cpu with pregst(myid,priort,int(expon(cpu)))
c* wait for service complete message
wait maxint for ((msg .eq. rqcomp).or. (msg .eq. qclose))
if (msg .eq. qclose) go to 10

branch back to tty or go on to printer

if (rand(x) .le. ttyprb) go to 10
request printer
c* decide upon printer to use according to probability of laser
prntyp(linepr) = 0
prntyp(laspr) = int(rand(x)+lasprb)
if(prntyp(laspr) .eq. 0) prntyp(linepr) = 1
invoke prnmg with pregtk(myid,priort,prntyp[2])

wait maxint for (msg .eq. havtok)
c* simulate printer usage
wait (expon(prnmm))
c* return printer to manager
invoke prnmg with reltok(myid,prntyp[2])
c* collect service stat
invoke stat1 with insert(csub(clock,begclk))

999 continue
ende

***************************************************************************

c* ENTITY MAIN

This is entity main which creates the active servers, passive resource manager, statistics entity, and source entity.

***************************************************************************
entity main

local integer cpu, tty, prnmg, stat1, src
integer initrs, srvrte, numres, resors(2), title(26), titlen, nmsrvs, queuecap
initial resources, service rate, number of resources, number of servers
queue capacity
parameter(initrs = 2, srvrte = 1, numres = 2, titlen=26, nmsrvs=1, queuecap=5)
include 'common.sim'
message

c* trace 3 when .true.
write(6, 9000)
format(1, "Start of simulation.")
set maximum simulation period, and izstim
simulation period 3000000
izstim = 1000010

c* initial number of passive resource (printer) resources
do 10, i=1,numres
   resors(i) = initrs
10 continue

c* create response time statistics entity
let stat1 be mnstat(titlen, title)

c* create tty
service rate 1 sim unit = 1 msec
let tty be izasdy(srvrte)

c* create cpu activer server and printer resource manager
service rate 1 sim unit = 1 msec
let cpu be izas13(preprei, srvrte, nmsrvs, queuecap)
let prnmg be iztml2(numres, resors, prltry)

c* create jobs
let src be source(cpu, prnmg, tty, stat1)

c* run for simulation period
wait izstim

c* dump statistics
invoke stat1 with dump(6)
invoke cpu with dump(6)
invoke prnmg with dump(6)
invoke tty with dump(6)

c* terminate
invoke cpu with trmnt
invoke prnmg with trmnt
invoke tty with trmnt
invoke src with trmnt
invoke stat1 with trmnt

c* write(6, 9001)
format(1, "Simulation has finished.")
ende

c* append informs MAY compiler of needed entities
append izasdy, izas13, iztml2, mnstat
6.2.1 Results for Example Four
start of simulation.
simulation has finished.

statistical dump at simulation time 1000010.

transaction response times
the total of all values is 8032277.
the total no. of values is 273
the average value is 29422.26

statistical dump at simulation time : 1000010

System statistics for active server
Server type = preemptive priority
Service rate = 1
Number of servers = 1
waiting queue capacity = 5

response time
min  max  mean  number
  0  420  53.160  1409

service time
min  max  mean  number
  0  347  49.772  1409

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean  max  current
  0.075  3   0

System statistics :
utilization  thruput  jobs completed  jobs pre-empted
  0.070  0.001   1409   38

statistical dump at simulation time : 1000010

System response time stats for token requests:
Token type: 1 initial tokens: 2
Token type: 2 initial tokens: 2
minimum response time : 0
maximum response time : 12086
number of responses : 273
mean response time : 491
Token type: 1 mean available tokens: 1
Token type: 2 mean available tokens: 1

System throughput : 0.00027
mean queue length : 0.13431
mean available tokens (all types): 2.93317
jobs still in queue : 0
statistical dump at simulation time : 1000010

System statistics for active server

Server type = delay

service rate = 1
maximum number of servers = 20

response time

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38944</td>
<td>4931.275</td>
<td>1416</td>
</tr>
</tbody>
</table>

service time

<table>
<thead>
<tr>
<th>min</th>
<th>max</th>
<th>mean</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>38944</td>
<td>4931.275</td>
<td>1416</td>
</tr>
</tbody>
</table>

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting

<table>
<thead>
<tr>
<th>mean</th>
<th>max</th>
<th>current</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.949</td>
<td>15</td>
<td>7</td>
</tr>
</tbody>
</table>

System statistics:

<table>
<thead>
<tr>
<th>utilization</th>
<th>thruput</th>
<th>jobs completed</th>
<th>jobs pre-empted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.050</td>
<td>0.001</td>
<td>1416</td>
<td>0</td>
</tr>
</tbody>
</table>

source terminated.
number of transactions created by source = 280
Simulation terminated normally
Simulation period specified: 3000000.
6.2.1 Code for Example Five
entity job(cpu, prnmg, tty, disk1, disk2, stat1)

integer cpumn, prnmm, thnkmn, dskmn, pbasze, pbsize
means for cpu, printers, think time, disks
parameter (cpumn = 50, prnmm = 4000, thnkmn = 5000, dskmn = 20)
real laspr, dsk1lp, dsk2pb, prnprb, x, exponent, prbarr(4), priarr(3)
laser probability, probability array sizes
parameter (laspr = .20, pbsize = 4, pbsize = 3)
type linepr, lasprn, ungrad, grad, prof, branch, prbgen, csub
resource request array indices, priority assignments
parameter (linepr = 1, lasprn = 2, ungrad = 3, grad = 2, prof = 1)
local integer prntyp(2), prior

clocktype begclk
cumulative probability array for branching after cpu
data prbarr/0.20, 0.30, 0.40, 1.00/
cumulative probability array for priority assignment
data priarr/0.20, 0.50, 1.00/
message rqcomp
message havtok
message qclosed

trace 3 when .true.

record starting time token for response stat
call cstore(begclk, clock)

set priority of transaction
prior = prbgen(priarr, pbsize)

continue

terminal think time
invoke tty with request(myid, int(expon(thnkmn)))

wait maxint for (msg .eq. rqcomp)

continue

request service from computer
invoke cpu with request(myid, prior, int(expon(cpumn)))
wait for service complete message
wait maxint for (msg .eq. rqcomp). or. (msg .eq. qclosed)

branch after cpu
printer = 1, disk1 = 2, disk2 = 3, tty = 4
branch = prbgen(prbarr, pbsize)
go to (30, 40, 50, 10) branch

continue

request printer
decide upon printer to use according to probability of lazer
prntyp(linepr) = 0
prntyp(lasprn) = int(rand(x) + laspr)
if (prntyp(lasprn) .eq. 0 ) prntyp(linepr) = 1
invoke prnmg with request(myid, prior, prntyp(2))
wait maxint for (msg .eq. havtok)

simulate printer usage
wait int(expon(prnm))
invoke prnmg with reitok(myid, prntyp(2))
go to 998
invoke disk1 with regest(myid, int(expon(dskmn)))
wait maxint for (msg .eq. rqcomp)
c*
return to cpu
go to 20

continue
invoke disk2 with regest(myid, int(expon(dskmn)))
wait maxint for (msg .eq. rqcomp)
c*
return to cpu
go to 20

continue
c*
collect response time stats
invoke stat1 with insert(csub(clock,begclk))

continue
ende

**********************
ENTITY MAIN
**********************

This is entity main which creates the active servers, passive
resource manager, statistics entity, and source entity.

entity main

integer nmres,resors(2),nmsrvs,quecap,titen
local integer cpu,prnmgr,tty,disk1,disk2,stat1,src
integer srvrte,initsr,titen(26)

service rate, initial resources, number of resources, number of
servers, queue capacity
parameter(srvrte = 1,initsr = 2,nmres = 2,nmsrvs = 2,quecap = 5)
parameter(titen = 26)
include 'common.sim'
data title/'T','R','A','N','S','A','T','I','O','N',
    'R','E','S','P','O','N','S','E','T','I','M','E',
message
c*
write(3,when .true.

9000
format("Start of simulation.")

set maximum simulation period
simulation period 2500000
izstim = 850001

c*
initial number of passive resource (printer) resources
do 10, i=1,nmres
    resors(i) = initsr
10 continue

c*
set up statistics entity for transaction response times
stat1 will be for simple recording of the mean response time
let stat1 be mnstat(titen,title)

create tty
let tty be izasdy(srvrte)
c*
create cpu activer server and printer resource manager
let cpu be izas13(prepri,srvrte,nmsrvs,quecap)
let prnmgr be iztuml2(nmres,resors,prity)
c*
create disks
let disk1 be izeal2(fcfs,srvrte)
let disk2 be izeal2(fcfs,srvrte)
create jobs
let src be source(cpu,prnmgr,tty,disk1,disk2,stat1)

wait izstim

dump statistics
invoke stat1 with dump(6)
invoke cpu with dump(6)
invoke prnmgr with dump(6)
invoke tty with dump(6)
invoke disk1 with dump(6)
invoke disk2 with dump(6)

terminate
invoke cpu with trmnt
invoke prnmgr with trmnt
invoke tty with trmnt
invoke disk1 with trmnt
invoke disk2 with trmnt
invoke src with trmnt
invoke stat1 with trmnt

write(6,9001)
9001 format(//."Simulation has finished")
ende

c* append informs MAY compiler of entities needed
append izeal1,izeasdy,izeal2,izeal3,mnstat
start of simulation.
simulation has finished

statistical dump at simulation time 850001.
the total of all values is 791502.
the total no. of values is 33
the average value is 23984.91

statistical dump at simulation time : 850001

System statistics for active server
Server type = preemptive priority
Service rate = 1
Number of servers = 2
waiting queue capacity = 5

response time
min max mean number
0 199 45.744 172

service time
min max mean number
0 199 45.744 172

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean max current
0.009 2 0

System statistics :
utilization thruput jobs completed jobs pre-empted
0.005 0.000 172 0

statistical dump at simulation time : 850001

System response time stats for token requests:
Token type: 1 initial tokens: 2
Token type: 2 initial tokens: 2
minimum response time : 0
maximum response time : 0
number of responses : 33
mean response time : 0
Token type: 1 mean available tokens: 1
6.2.1 Results for Example Five
FUNCTION:
  This is the source entity which creates transactions according to a specified interarrival rate. Transactions are created until SOURCE is terminated.

PARAMETERS:
  CPU: cpu active server entity id.
  PRNMG: printer manager entity id.
  TTY: tty active server entity id.
  DISK1: disk1 active server entity id.
  DISK2: disk2 active server entity id.
  STAT1: statistics entity id.

MESSAGES:
  TRMNT: Message to terminate the entity-instance.

entity source(cpu, prnmg, tty, disk1, disk2, stat1)
local integer jobcnt, intarr
integer trans
real expon
message trmnt

trace 3 when .true.

set interarrival rate
intarr = 25000
jobcnt = 0

create transactions
continue
let trans be job(cpu, prnmg, tty, disk1, disk2, stat1)
jobcnt = jobcnt + 1
wait int(expon(intarr))
if (msg .ne. trmnt) go to 10

write(6, 9000) jobcnt
9000 format(11, "Source Termination.",
          + "," Number of transactions created by source = ", i6)

end

FUNCTION:
  This is the job entity.

PARAMETERS:
  CPU: cpu active server entity id.
  PRNMG: printer manager entity id.
  TTY: tty active server entity id.
  DISK1: disk1 active server entity id.
  DISK2: disk2 active server entity id.
  STAT1: statistics entity id.

MESSAGES:
Token type: 2 mean available tokens: 1
System throughput: 0.00004
mean queue length: 0.
mean available tokens (all types): 3.82239
jobs still in queue: 0

statistical dump at simulation time: 850001

System statistics for active server
Server type = delay
service rate = 1
maximum number of servers = 20
response time
  min  max  mean  number
  53  23053  4612.314  137
service time
  min  max  mean  number
  53  23053  4612.314  137

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean max current
  0.743  4  0

System statistics :
  utilization thruput jobs completed jobs pre-empted
  0.026  0.000  137  0

statistical dump at simulation time: 850001

System statistics for active server
Server type = fcfs
Service rate = 1
Number of servers = 1
response time
  min  max  mean  number
  0    77  24.375  16
service time
  min  max  mean  number
  0    77  24.375  16
Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean max current
0.000 1 0

System statistics :
utilization thruput jobs completed jobs pre-empted
0.000 0.000 16 0

statistical dump at simulation time : 850001

System statistics for active server
Server type = fcfs
Service rate = 1
Number of servers = 1

response time
min max mean number
2 62 20.632 19

service time
min max mean number
2 62 20.632 19

Queue length statistics:
total jobs at server = jobs being serviced + jobs waiting
mean max current
0.000 1 0

System statistics :
utilization thruput jobs completed jobs pre-empted
0.000 0.000 16 0

source termination.
number of transactions created by source = 33
No active entities present.
Simulation terminated at time: 850001.