EE 367 Machine Problem 1: Server Design. 20 points

1 Introduction

Suppose there is a company called, CheapPictures.com, that has an Internet server. The server responds to requests from clients as shown in Figure 1. Examples of client requests are downloading files (mpg, jpg, etc) or running applications on the server.

When a client request arrives at the server, it is put into a "request queue" at the server. Whenever the server is "idle" (i.e., not processing a request, such as when it has completed processing a request), it immediately "accepts" another request by removing it from the head of the queue. Then it immediately begins processing.

The server is manufactured by the Manoa Warrior Communication company. It is a third generation machine called Warrior 3. This machine has only one processor, and can only service one request at a time. To ensure that clients experience reasonably low delay, the size of the queue is limited to 4. If a client's request finds the queue full with 4 requests then it is "blocked" (i.e., it is dropped and never processed).

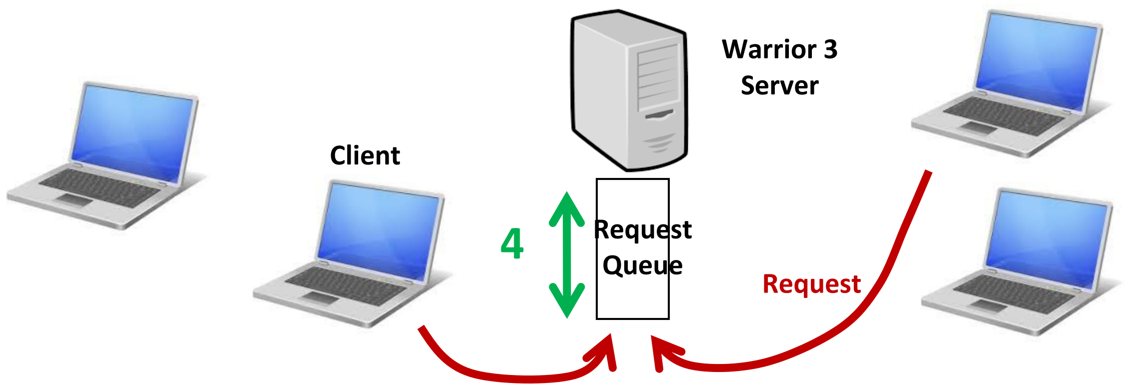

Figure 1. Client server scenario.

The chief engineer of CheapPictures.com wants to determine if it is better to upgrade the server. To study the economics, she considers the cost of an upgrade versus lost revenue. The lost revenue is due to blocked requests. She estimates that each blocked requests costs the company $10 because

- (i) the revenue from the blocked request is lost and
- (ii) future revenue may be lost because the client may become discouraged by poor service and switch to another company.

She considers two upgrades.

Upgrade 1. Double Warrior 3: This upgrade is to purchase another Warrior 3 server and a simple switch as shown in Figure 2. The switch receives all requests and then immediately forwards them to the servers, where the forwarding is assumed to take negligible time. It does this in a round robin fashion, and in particular, all even number requests go to server 0 and all odd numbered requests go to server 1. Thus, request 0 goes to server 0, request 1 goes to server 1, request 2 goes to server 0, request 3 goes to server 1, and so on. Note that since the switch forwards requests independent of the state of the servers, an even numbered request goes to server 0, even though server 0 is full and server 1 is empty; and similarly, an odd numbered request goes to server 1 no matter what the occupancy is of server 0.

A Warrior 3 server costs $11,000 while the switch cost is negligible. ■

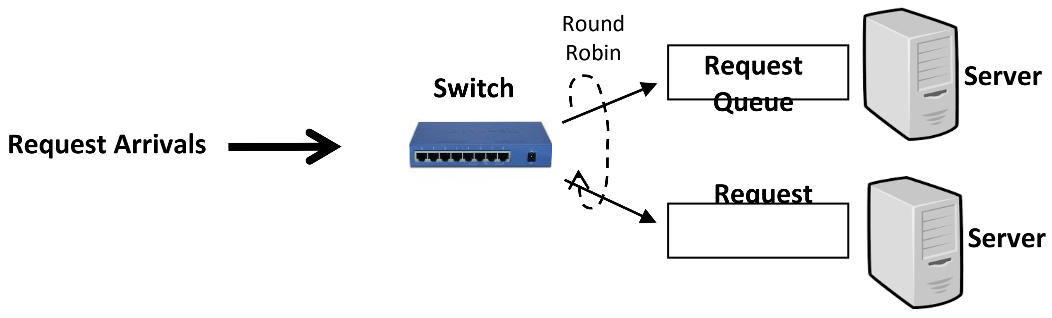

Figure 2. Double Warrior 3.

Upgrade 2. Warrior 4: This upgrade is to replace the current Warrior 3 server with the newest server, Warrior 4. The new server has two processors, so it can service two requests at the same time. The processors are the same as in Warrior 3. When a processor becomes idle (e.g., completing service of a request), it will remove a request from the request queue and begin servicing it. Figure 3 shows Warrior 4 and Warrior 3.

The request queue has maximum size of 8. This ensures that the maximum delay of a request is about the same as Warrior 3. Warrior 4 can have queue size twice as long as Warrior 3 because it processes requests twice as fast.

Warrior 4 costs $13,000. ■

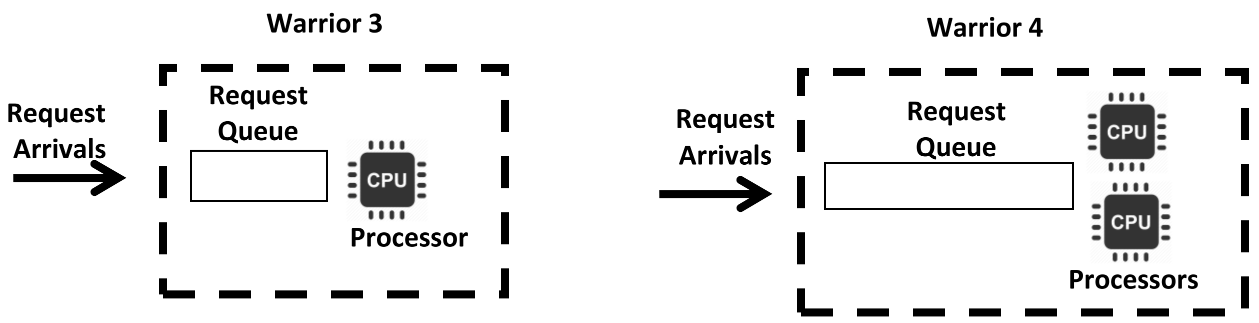

Figure 3. Warrior 3 and Warrior 4.

The chief engineer has three options:

(1) make no changes,
(2) buy another Warrior 3 and switch, and
(3) buy a Warrior 4.

She decides that her decision should be based on the operation of the server over a one year period. She will compare the options using the following formula:

$$F = \$10 \times S - \text{Cost of the upgrade}$$

where

- $S = B - B^*$
- B = the number of blocked requests over the year with the current server, Warrior 3
- B^* = the number of blocked requests over the year using the upgraded server system

Thus, S is the number requests that are saved over the year because of the upgraded server system. Note that F is the marginal improvement in overall revenue over a year. If $F > 0$ then it is better to make the upgrade.

2 Modeling the Server Systems

To determine B for the current Warrior 3 server, the chief engineer wrote a simulator for it using the following model:

Time

- The simulation is over a time line of 8 hours (830 AM to 430 PM). This is the peak request period, and outside this period there are no blocking of requests.
- The time line is slotted, where each slot is exactly one second. There are 8 hours x 60 min x 60 sec = 28800 time slots. Time slots are numbered 0, 1, ..., 28799.

Arriving Requests

- Requests are numbered 0, 1, 2,....
- At most one request arrives in a time slot.
- Requests arrive at the beginning of time slots.
- If a request arrives when the request queue is full then it is blocked and considered no further; otherwise, it is immediately put into the request queue.

Servicing Requests

- Each request k has a service time $S(k)$, which is the number of time slots it requires from a processor to service its request. For example, if $S(k) = 220$ then request k takes 220 seconds to process with a single processor.
- Immediately after requests arrive, an idle processor will remove a request from the request queue and begin service. Note that this occurs at the beginning of time slots.
- Requests complete service at the end of time slots. For example, consider a request with service time 3 that begins service at a processor in time slot 10. Then it is serviced in time slots 10 and 11, and completes service at the end of time slot 12.
- When a processor completes service of a request, it immediately removes another request from the request queue. It begins service of the new request at the beginning of the next time slot. For example, suppose a processor completes service of a request at the end of time slot 10 then it removes another request from the queue which has service time 3. Then it begins service at the beginning of time slot 11 and completes service at the end of time slot 13.

The simulator will simulate the system over time. It keeps track of the "current" time slot with a variable `ctime`, which is initially `ctime` = 0. Then it simulates time-slot by time-slot. This is known as "discrete time" simulation.

The chief engineer has collected data about request arrivals over a typical day. It is stored in a file called "request.dat". It is assumed that this data is true for every day in the year. Thus, the blocking results from "request.dat" are assumed to be true for each of the 365 days.

All the numbers in this file are integers. The file's first line the number of requests that arrived during the time period. The subsequent lines are the information per request. Each line has a request's arrival time slot and service time. For example,

```
3
104   20
110   14
572   29
```

means that three requests arrived during the period. Request 0 arrived at the beginning of time slot 104 and has service time of 20 time. Since it arrived to an empty system, the idle processor began servicing it immediately at the beginning of time of 104. The request uses 20 time slots of service, so it leaves at the end of time slot 123. Next request 1 arrives at time slot 110 and has service time 14, and request 2 arrives at time slot 72 and has service time 29.

The chief engineer has written a simulator called queue.c. At the beginning of a run, it will ask the user for the file with the request data. Then it simulates the server system, and keeps track of the number of arriving requests that are blocked. The number blocked is displayed at the end of a simulation run.

Note that at the end of a simulation run there may be requests left in the request queue. The simulator does not consider these since they have been accepted and therefore do not contribute to the blocked requests. The simulator algorithm is given in Appendix A.

4 Assignment

Modify the chief engineer's program so that it also simulates the upgrades Double Warrior 3 and Warrior 4. For these two upgrades display

- Daily number of blocked requests
- Annual number of blocked requests

The current program `queue.c` has functions `warrior3( )`, `dblwarrior( )`, and `warrior4( )` that returns the number of request blockings for Warrior 3, Double Warrior 3, and Warrior 4, respectively. Only `warrior3( )` has been implemented but for a request queue of size one rather than four.

Attached to this assignment are files from the chief engineer: `queue.c` and `request.dat`, which is the file of request data. Modify queue.c so that it simulates the three server systems correctly. Be sure that queue.c can compile and run on wiliki.

Upload into laulima,

1. Your queue.c
2. A README file that includes:
   - Your name
   - Date
   - Explanation of what your program can and can't do. For example, it compiles but doesn't work for any of the upgrades, or works for Double Warrior 3 but not Warrior 4.
   - Brief explanation of each source code file.
   - Instructions on how to compile in wiliki
   - Instructions on how to run your program.
   - Which of the three options should the chief engineer choose: (1) Do no upgrades, (2) Double Warrior 3 upgrade, and (3) Warrior 4 upgrade.
   - For each working upgrade, describe what its value for *F*, the marginal improvement in overall revenue over a year.
   - Explain which upgrade you would choose if any.

If your program

- Simulates only Warrior 3 (with queue size 4) then you will receive 12 points
- Simulates Warrior 3 and one of the two upgrades, Double Warrior 3 or Warrior 4, then you will receive 14 points.
- Completely works then you will receive the full credit of 20 points.

Appendix A. The Simulator Algorithm

The following is an algorithm of the simulator that the chief engineer wrote.

```
Main
Input data as a sequence of requests in a linked list, which will be referred to as the arrival list.
Call subroutine warrior3
Display results
Call subroutine dblwarrior
Display results
Call subroutine warrior4
Display results

warrior3
Initialize t = 0  Comment:  The current time
Create and initialize a request queue
Create and initialize a processor
numblocked = 0;

for    t = 0, 1, ..., until no more arriving requests   do

        Comment:  The following happens at the beginning of the time slot “time”
        if the next arriving request is at time t  then
                if the queue is full then numblocked = numblocked + 1
                else add the request to the request queue
                Update the next arriving request to the next request in the arrival list.
        while a processor is empty do
                Remove a request from the request queue and put it in the processor
                In the processor, record the time slot when the request will complete service
                    Comment:  This will be t + s – 1, where s is the service time of the request
                    Comment:  As a sanity check, suppose s = 1.  Then the request arrives and will complete service
                        in the same time slot t.  This is consistent with our model.

        Comment:  The following happens at the end of the time slot t
        for each busy processor do
                if time to complete service = t then
                    Remove the request from the processor since its service has been completed
        Comment:  At this point, some processor may have become idle.
        for each idle processor do
                Remove a request from the request queue and put it in the processor
                In the processor, record the time slot when the request will complete service.
                     Comment:  This will be t + s, where s is the service time of the request.
                     Comment: As a sanity check, suppose the service time is 1.  Then the request will be
                         serviced by time t+1.  This is consistent with our model since the request is
                         assigned to the processor at the end of time t, and finishes service at the end
                         of time t+1.

return numblocked
```