Pseudo Code for all important methods/classes etc in your Simulator

simulation main:

Create a list of servers, with their IDs, initialize all with state idle

and reqInService = NULL.

initialize simulation {

set time=0,

schedule initial events

(create Event objects, timestamp them,

Then throw the event into the event list)

}

while (eventlist not empty) {

remove the nearest event (pop from priority queue, key being event timestamp)

advance simulation clock to this event time

handle event based on event type (i.e. call event handler).

(This will usually be a switch that picks the event handler based on the event

type)

}

arrival_event_handler {

We first create a request object from the event information

Then we check if buffer is not full.

If buffer is full,

request is dropped and no change in system state occurs.

If not buffer is not full, push the request into buffer (note that buffer is our only way of entry).

Now we scan through the server list to look for an idle server.

If idle server found,

dequeue from the buffer and put it into service at this idle server.

Else, no dequeuing is done and there is no change in system state except the queuing of a request .

Updation of metrics also occurs at appropriate times.

Also we schedule the arrival of next request in arrival event handler by creating an Event object and push into EventList

}

departure_event_handler {

On a departure event, only one of the following two cases can happen:

Either there is no request waiting in the buffer.

In this case we set the server state (where departure occurred) to “idle” and (pointer to)request in service is set to NULL

Or there is a request waiting in the buffer.

In this case, we dequeue the buffer on a FIFO basis. And set the request dequeued is set as the request in service at the server (where departure occurred). So the server state remains “busy”.

At this moment we also schedule the departure of this request, at time = Simulation clock time + service time, and push this event into event list.

}

probe_event_handler {

At a probe, whenever job queue length (waiting requests only) is seen above threshold JHigh,

speed is increased by S (upto max). Whenever job queue length is seen below threshold Jlow,

speed is decreased by S (upto min).

New departure time is computed. And departure Events are rescheduled with the help of indexedHeap.

We also schedule the next probe event at time simTime+P. But only till the time when we do not have any arrival or departure event into the eventList.

}

indexedHeap:

const Comparable & insert(const Comparable & x) {

Create ‘hole” at position beyond last element. percolateUp the hole created and insert the element at appropriate position.
Position[ ] array will also be updated accordingly.

}

```
void deleteMin() {
copy the key of array[currentsize] to array[1],
decrease the current size by one and percolateDown(1)
update position[ ] accordingly
}
changeKey(idType id, newKey){
this will change the key value of the object whose ID
in the heap was “id” to newKey
with the help of position[id] we get index of required key in heap
vector.
Now change its key value to newKey and call buildHeap.
}
```