Simulation Program Class Design

CS 681

Chaitanya(203050026)
Mahendra(203050078)

Assumptions

- All Clients have same think time.
- Request Buffer size is limited.
- Requests will be dropped when buffer is full.
- Each CPU core has separate Job queue.
- Number of threads are limited.
- There is no listener thread,which will accept requests and send them to request buffer
  - If a request arrives the system,it will directly go to request buffer.
- If a thread is free it will pick,a pending requests from the buffer queue if any.
- Request time out is variable for each request.

Class Design

```
Class Request

{
  int clientid;
  float Arrivaltime;
  string location;
  float remaining_service_time;
  float time_out;
}


Class CPU_Core
{
     queue<Request> job_queue
     //Indicates Core is occupied or not
     string status
}
```

Class Design

```
Class Server

{

           int no_of_cpu_cores;
           int no_of_threads;
          CPU_Core[] CPU_CORES;
          Thread[] SERVER_THREADS;
           Queue<Request> Request_buffer;

           def  initialize()
           {
             Initializes cpu cores and server thread objects
              CPU_CORES=CPU_Core[no_of_cpu_cores]
              SERVER_THREADS=Thread[no_of_threads]
             Request_buffer=new Queue<Request>()
           }

  }
```

```
def get_idle_thread()
{
 //returns the idle thread if any

}
```

Class Design

```
Class **RandGenerator**
{
                def  unform_rand(mean)
               {
                    //Generates uniform random number with given mean


                }

               def expon_rand(mean)
               {

                  //Generates exponential random number  with given mean
               }



}
```

Class Design

```
Class Clients
{
       int no_of_clients
       int think_time
 }


Class Client_Server_System
{

           server=new Server();

           server.initialize();

          clients=new Clients();

      map<int,int> clent_thread_mapping;
      map<int,Request> client_request_mapping;
}
```

Class Design

```
Class Event
{
        float timestamp;
        string eventtype;
        int clientid;
}

enum EvenType
{
 Arrival,
 ContextSwitch,
 Departure
}
```

```
Class Metrics
{
        int no_of_departures;
       int  no_of_packets_dropped;
       int  badput_counter;
      float total_resposne_time;
      float cpu_utilisation;
}
```

Class Design

```
Class Simulator
{

       PriorityQueue<Event> prority_queue
      metrics=new Metrics()
      system=new Client_Server_System()


       //Initializes priority queue  with initial time stamps for
the events
      def Initialization();




}
```

```
//This will call event handlers based on event type
def Run_Simulation()
{
        while(end_of_simulation)
        {
                event_to_be_processed=prority_queue.first();
                event_type=event_to_be_processed.eventtype
                switch(event_type)
                {

                Case "Arrival":
                        Arrival(event_to_be_processed)
                        break;
                 Case "Departure":
                        Departure(event_to_be_processed)
                        break;
                 Case "Context_Switch":
Context_Switch(event_to_be_processed)
                        break;
                }
        }
}
```

Event Handlers(Arrival)

```
def  Arrival(arrival_event)
{
            Check if request buffer is full
             If No,
                  1.    Push the request to request_buffer queue.
                  2.    Find a thread which is idle.
                  3.    If an idle thread is found,assign the request to the thread.
                  4.    Push the request to Corresponding CPU job queue
                  5.    Check if corresponding CPU core is occupied or not
                          a.    If CPU is not occupied,
                                 i.    if (remaining_service_time-time_quantum)>0 schedule its context switch event
                                ii.    else  schedule departure for this request
                  6.    Set Arrival Event timestamp of this request to inf, so that it won't get scheduled next

            If Yes,
                  1.    Increment drop counter
                  2.    Schedule new arrival by adding Think time

}
```

Event Handlers(Departure)

```
def Departure(departure_event)
{
        1.  Remove the request from corresponding CPU job queue
        2.  Check if there are other pending requests in the job queue
            a.  If yes,
                i.   if (remaining_service_time-time_quantum)>0 for that request schedule its context switch event
                ii.  else  schedule departure for that request
            b.  If no, indicate that CPU is free now
        3.  Find the response time for this request and add to total response time
        4.  If response time>request.time_out:
            a.  Increment bad put counter
        5.  Else:
            a.  Increment Departures Counter
        6.  Schedule Arrival event for new request of this client by adding think time to current timestamp
        7.  Set the time_out of this new request equal to timeout_min_value+some variable number.
        8.  Set Departure event’s timestamp for new request of this client  to inf, so that it won't get scheduled next
        9.  A request from request_buffer is popped and assigned to this thread for execution.
}
```

Event Handlers(Context Switch)

```
def Context_Switch(context_switch_event)
{

        1.  Remove the request from the cpu job queue.
        2.  Set the remaining_service_time of the request as req.remaining_service_time-=time_quantum.
        3.  Check if there is next request in the queue
              i.  If yes,
                    1.  Find the next request to be serviced
                    2.  if (remaining_service_time-time_quantum)>0 schedule its context switch event
                    3.  else  schedule departure for this request
                    4.  Set current request’s context_switch timestamp as inf
             ii.  If no,
                    1.  if (remaining_service_time-time_quantum)>0 Set current request’s context_switch timestamp by
                        adding time quantum to current timestamp
                    2.  else  schedule departure for this request
        4. Push request again into the cpu job queue.
}
```