Individual Assignment - 2

Topic : Operating System

YOUR NAME
ROLL NO

YOUR DEPARTMENT

YOUR INSTITUTE NAME

06^{th} ***December,*** 2020

Contents

Chapter 1

Task-1

1.1 Comparison of FreeBSD ULE and Linux CFS Schedulers

- **Does ULE support threads, or does it support processes only? How about CFS?**
  - ULE as well as CFS both support threads. The paper suggests to provide the outcomes of the both the schedulers when run on a particular suite of applications.

- **How does CFS select the next task to be executed?**
  - CFS implements a weighted fair queueing algorithm. It evenly divides CPU cycles between threads weighted by their priority. CFS also uses heuristics to improve cache usage.

- **What is a cgroup and how is it used by CFS? Does ULE support cgroups?**
  - Threads of the same application are grouped into a structure called a cgroup. A cgroup has a vruntime that corresponds to the sum of the vruntimes of all of its threads. CFS uses this to apply in its algorithm, the thread with the lowest vruntime is executed first.
    No, ULE doesnt support cgroups. It treats threads as independent entitities.

- **How many queues in total does ULE use? What is the purpose of each queue?**
  - ULE uses two queues to schedule threads: one queue contains interactive threads, and the other contains batch threads.

- **How does ULE compute priority for various tasks?**

  - ULE first computes a score defined as interactivity penalty + niceness. A negative nice value (high priority) makes it easier for a thread to be considered interactive.

- **Do CFS and ULE support task preemption? Are there any limitations?**
  - In ULE, full preemption is disabled, meaning that only kernel threads can preempt others. CFS uses heuristics to improve cache usage. However it has to sacrifice latency in order to avoid frequent preemption, which may hamper cache usage.

- **Did Bouron et al. discover large differences in per-core scheduling performance between CFS and ULE? Which definition of "performance" did they use in their benchmark, and why?**
  - The main difference between CFS and ULE in per-core scheduling is in the handling of batch threads: CFS tries to be fair to all threads, while ULE gives priority to interactive threads.
    ULE batch applications can starve for an unbounded amount of time, leading to unfairness.
    "Performance" is defined as : for database workloads and NAS applications, we compare the number of operations per second, and for the other applications we compare "1/execution time". The higher the "performance", the better a scheduler performs

- **What is the difference between the multi-core load balancing strategies used by CFS and ULE? Is any of them faster? Does any of them typically reach perfect load balancing?**
  - CFS relies on a rather complex load metric. It uses a hierarchical load balancing strategy that runs every 4ms. ULE only tries to even out the number of threads on the cores. Load balancing happens less often (the period varies between 0.5s and 1.5s) and ignores the hardware topology.
    CFS balances the load much faster than ULE. ULE reaches a balanced state however CFS being faster than ULE does not achieve a perfect load balancing.

Chapter 2

Task - 2

2.1 Developing a Scheduling Algorithm in Java

- The scheduling algorithm is able to work with both premptive as well as non-premptive processes.
- Usage
  - `ScheduleProcess` - class (id, burstTime, arrivalTime, waitingTime, cpuAllocationTime)
  - `processQueue` - Linked List of Process
- Creating the processes and adding them into `processQueue`

```
      // creating process and adding them into processQueue
      int pId = 1;
    int arTime = 0;
    for(int i=0 ; ; i+=1 ){
      int burstTime = this.rng.nextInt(maxBurstTime - minBurstTime) +
          minBurstTime;
      int id = pId;
      // getArrivalTime - it gets the arrival time for the next expected arrival
          time with a probability of 0.75
      int arrivalTime = getArrivalTime(arTime,probArrival);

      // if arrivalTime is -1 -> the process didnt arrive (got probability 0)
      if( arrivalTime == -1 )
      {
        arTime+=1;
        continue;
      }
      ScheduledProcess p = new ScheduledProcess( id, burstTime, arrivalTime, 0,
          0 );
      processQueue.add( p );
      if( processQueue.size() == numProcesses )
        break;
      pId += 1;
      arTime+=1;

    }
```

- following is the non-premptive scheduler

```
if( !isPreemptive )
{
   // start the simulation - NonPreemptive Version
   int time = 0;
   int sumTime = 0;
   while(true)
   {
      // randomly choose a process to give it CPU
      int randomProcess = this.rng.nextInt( processQueue.size() );
      ScheduledProcess p = processQueue.get(randomProcess);
      if( time>=p.arrivalMoment )
      {
         // if currentTime is >= arrival then only execute this process
         p.totalWaitingTime += time;
         sumTime += time;
         time += p.burstTime;
         p.allocatedCpuTime = p.burstTime;
         p.burstTime = 0;
         // execute the process completely
         this.result.add( p );
         processQueue.remove(randomProcess);
         if( processQueue.isEmpty() )
            break;
      } else {
         // if currentTime is < arrival, then dont sit idle choose a process
             randomly
         time+=1;
         continue;
      }
   }
   this.avgWaitingTime = sumTime / numProcesses;
   this.completeExecTime = time;
}
```

- following is the premptive scheduler

```
if( isPreemptive )
{
   // each chosen process should run only for timeQuantum time unit
   int time = 0;
   int sumTime = 0;
   while(true)
   {
      // randomly choose a process to give it CPU
      int randomProcess = this.rng.nextInt( processQueue.size() );
      ScheduledProcess p = processQueue.get(randomProcess);
      if( time>=p.arrivalMoment )
      {
         // if currentTime is >= arrival then only execute this process
         p.totalWaitingTime += time;
         sumTime += time;
         time += timeQuantum;
         p.allocatedCpuTime += timeQuantum;
         p.burstTime -= timeQuantum;
         if( p.burstTime <= 0 )
         {
```

```
                    // process is completely finished
                    p.burstTime = 0;
                    this.result.add( p );
                    processQueue.remove(randomProcess);
                } else {
                    processQueue.remove(randomProcess);
                    processQueue.add( p );
                }
                if( processQueue.isEmpty() )
                    break;
            } else {
                // if currentTime is < arrival, then dont sit idle choose a process
                    randomly
                time+=1;
                continue;
            }
        }
        this.avgWaitingTime = sumTime / numProcesses;
        this.completeExecTime = time;
    }
```

- Results - Screenshots

```
********************************************************************************
                    Running non-preemptive simulation #3
********************************************************************************
Process Information
PID        ArrivalTime     BurstTime     TotalWaitingTime        TotalAllocatedCpuTime
----------------------------------------------------------------------------------
#1              1               6               N.A.                    N.A.
----------------------------------------------------------------------------------
#2              2               4               N.A.                    N.A.
----------------------------------------------------------------------------------
#3              3               7               N.A.                    N.A.
----------------------------------------------------------------------------------
#4              4               9               N.A.                    N.A.
----------------------------------------------------------------------------------
#5              5               8               N.A.                    N.A.
----------------------------------------------------------------------------------
#6              6               6               N.A.                    N.A.
----------------------------------------------------------------------------------
#7              7               2               N.A.                    N.A.
----------------------------------------------------------------------------------
#8              10              4               N.A.                    N.A.
----------------------------------------------------------------------------------
#9              11              4               N.A.                    N.A.
----------------------------------------------------------------------------------
#10             12              6               N.A.                    N.A.
```

(a) Process Information

```
Simulation results:
PID        ArrivalTime     BurstTime     TotalWaitingTime        TotalAllocatedCpuTime
----------------------------------------------------------------------------------
#1              1               0               5                       6
----------------------------------------------------------------------------------
#6              6               0               11                      6
----------------------------------------------------------------------------------
#8              10              0               17                      4
----------------------------------------------------------------------------------
#5              5               0               21                      8
----------------------------------------------------------------------------------
#3              3               0               29                      7
----------------------------------------------------------------------------------
#10             12              0               36                      6
----------------------------------------------------------------------------------
#7              7               0               42                      2
----------------------------------------------------------------------------------
#2              2               0               44                      4
----------------------------------------------------------------------------------
#9              11              0               48                      4
----------------------------------------------------------------------------------
#4              4               0               52                      9
Total Execution Time : 61
Average Waiting Time : 30
```

(b) After Simulation

Figure 2.1: Non-Premptive Scheduler

```
********************************************************************************
                    Running preemptive simulation #1
********************************************************************************
Process Information
PID        ArrivalTime     BurstTime     TotalWaitingTime        TotalAllocatedCpuTime
----------------------------------------------------------------------------------
#1              1               6               N.A.                    N.A.
----------------------------------------------------------------------------------
#2              2               3               N.A.                    N.A.
----------------------------------------------------------------------------------
#3              3               5               N.A.                    N.A.
----------------------------------------------------------------------------------
#4              4               5               N.A.                    N.A.
----------------------------------------------------------------------------------
#5              7               7               N.A.                    N.A.
----------------------------------------------------------------------------------
#6              9               6               N.A.                    N.A.
----------------------------------------------------------------------------------
#7              10              9               N.A.                    N.A.
----------------------------------------------------------------------------------
#8              11              9               N.A.                    N.A.
----------------------------------------------------------------------------------
#9              12              7               N.A.                    N.A.
----------------------------------------------------------------------------------
#10             13              6               N.A.                    N.A.
```

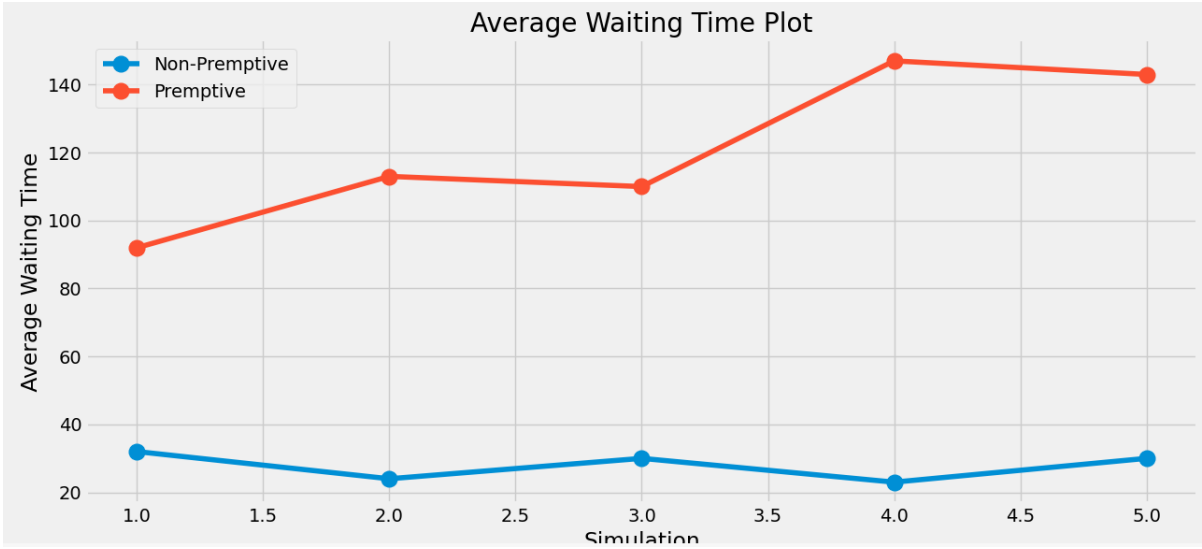
(a) Process Information

```
Simulation results:
PID        ArrivalTime     BurstTime     TotalWaitingTime        TotalAllocatedCpuTime
----------------------------------------------------------------------------------
#1              1               0               21                      6
----------------------------------------------------------------------------------
#9              12              0               120                     8
----------------------------------------------------------------------------------
#6              9               0               106                     6
----------------------------------------------------------------------------------
#4              4               0               118                     6
----------------------------------------------------------------------------------
#3              3               0               132                     6
----------------------------------------------------------------------------------
#7              10              0               188                     10
----------------------------------------------------------------------------------
#8              11              0               290                     10
----------------------------------------------------------------------------------
#5              7               0               178                     8
----------------------------------------------------------------------------------
#10             13              0               150                     6
----------------------------------------------------------------------------------
#2              2               0               92                      4
Total Execution Time : 76
Average Waiting Time : 139
```

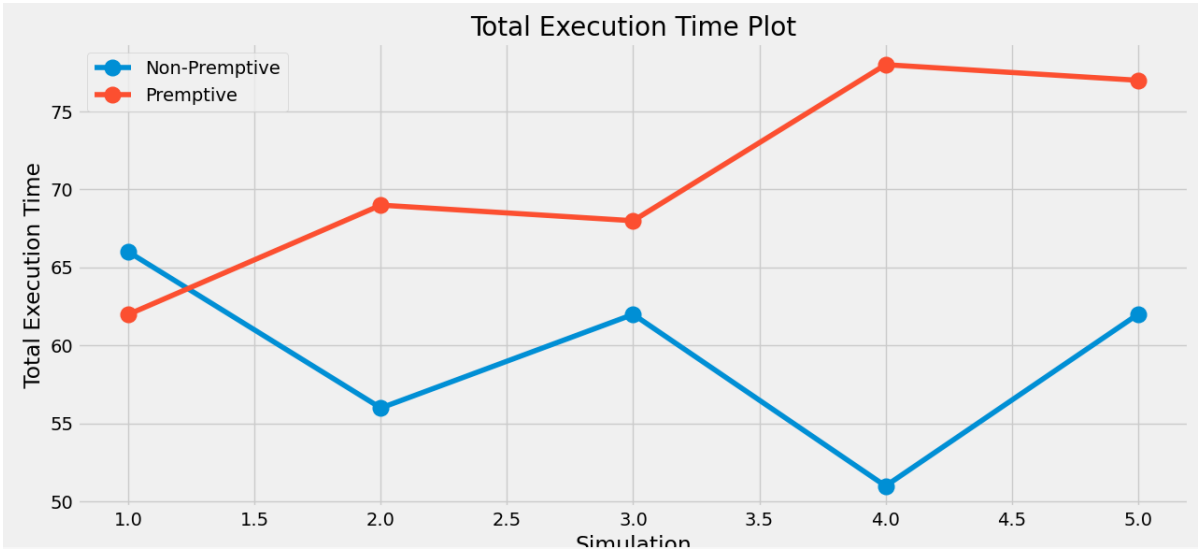
(b) After Simulation

Figure 2.2: Premptive Scheduler

- Results - Graphs



(a) Total Average Waiting time

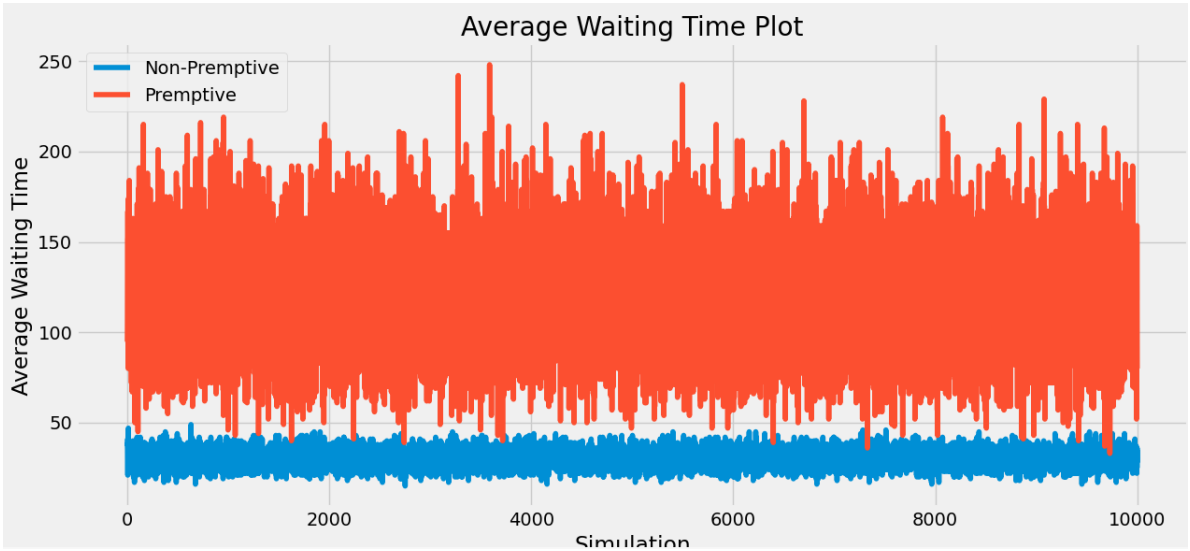


(b) Total Execution Time

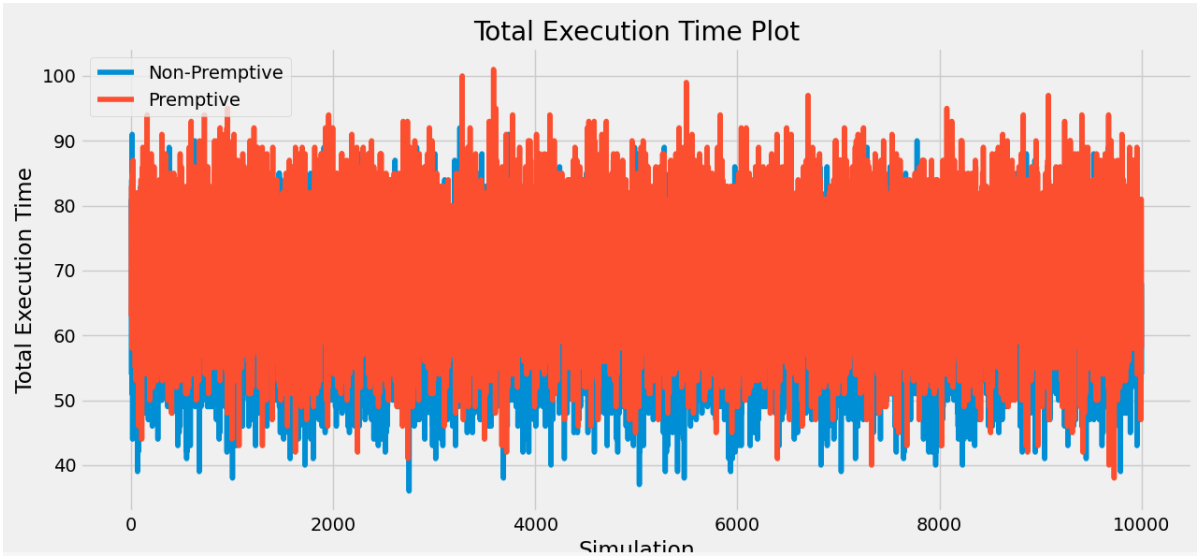
Figure 2.3: Comparison - Waiting Time Vs Execution Time

- **Observable pattern** - From the above graph we can say that both average waiting time and Total Execution time for preemptive scheduler increases and for non-preemptive they are neither increasing nor decreasing. However, one can't be sure as the number of simulations are very less.

- **10,000 iterations** - We ran the random scheduler for 10,000 iterations. We can see a clear pattern than non-preemptive scheduler is always better in Average Waiting Time. However we still can't determine a pattern in total execution time. Refer below comparison-graphs.



(a) Total Average Waiting time



(b) Total Execution Time

Figure 2.4: Comparison - Waiting Time Vs Execution Time (10,000 iteration)

- We dont see any advantage of the random scheduler over FCFS. However, if the random function somehow selects the smallest burst time process or near to it then it can outperform the FCFS algorithm. But this case is near to impossible.