# Lab 3: CPU Scheduling

In this lab you should use the CPU simulation program provided to gather data to complete copies of the table below. Read the “Instructions for using the CPU scheduling simulator” on Canvas before you begin.

Ask the CPU scheduling simulator to generate random data and use the snipping tool to copy the output to show the random data generated. **You can re-run with new data at any stage, just make sure to show the new data at the relevant stage. Alternatively you can enter your own data as needed.**

1. **For each** algorithm simulation run:
   1. Use the snipping tool to copy and paste the output into your document.
   2. Indicate whether the algorithm is pre-emptive or non-pre-emptive.
   3. Identify each time unit that a process switch occurs and fully explain why a switch happened at that time (e.g. because the previous process’s burst was complete or because the previous process was pre-empted – **fully explain why**).
   4. For each process switch that occurs, fully **explain why** the relevant scheduling algorithm made the choice it did. (Identify the processes that were in the competition (i.e. in the ‘ready’ pool) at that time. Fully explain why the winner was chosen.)
   5. Fill in a copy of the table provided below. Look at the example table to see how.
2. SPN and SRT both allow starvation but the others do not. To investigate this load the default data (option 2 in the simulator menu) then run the SPN or SRT algorithms as normal and note when process p1 gets its first use of the CPU.   
   Then re-run the same algorithm using the facility to ‘add data on the fly’. At time 3 when asked if you want to add data say ‘y’ and add 10 new processes all arriving now (i.e. time 0) and looking for 1 unit of service.  
   Now note again when process p1 gets its first use of the CPU.   
   Are the new processes always able to jump the queue? Why is that?   
   What would happen if you continued to add new short processes at times 5,6,7, etc.?   
   Try to add new processes at about the same time during the run of the other algorithms (FCFS [at time 3], RR [at time 4], HRRN [at time 3]) to see if there is a difference. Record your observations. Are the new processes always able to jump the queue? Why is that? What would happen if you continued to add new short processes at times 5,6,7, etc.?

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** |  |  |  |  |  |
| **Process** | **Arrival Time** | **Service burst** | **Wait time** | **Turn-around time** | **NTT ratio (show fractions)** |
| **p0** |  |  |  |  |  |
| **p1** |  |  |  |  |  |
| **p2** |  |  |  |  |  |
| **p3** |  |  |  |  |  |
| **p4** |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EXAMPLE** |  |  |  |  |  |
| **Process** | **Arrival Time** | **Service burst** | **Wait time** | **Turn-around time** | **NTT ratio (show fractions)** |
| **p0** | 0 | 2 | 0 | 0+2=2 | 2/2 |
| **p1** | 1 | 6 | 1 | 1+6=7 | 7/6 |
| **p2** | 3 | 3 | 5 | 5+3=8 | 8/3 |
| **p3** | 4 | 1 | 7 | 7+1=8 | 8/1 |
| **p4** | 6 | 7 | 6 | 6+7=13 | 13/7 |