**Completed and incomplete task:**

1. **Finished tasks :**

* Implemented ThreadCB modules by following steps in chapter 4 of OSP2.
* Collected output system performance (throughput, response time) to the log.
* Implemented FCFS and Round Robin algorithms.
* Tested the FCFS and Round Robin Algorithm and collect their performance data with different types of threads (long and short).
* Attached Readme file with Report.
* Prepared three minutes short presentation.
* Compared all algorithms implemented based on our experiment in your readme file/report and presentation.

1. **Incomplete task :**

* Not implemented any version of feedback algorithm.
* Not implemented the priority-driven preemptive scheduling.

**Summary of scheduling algorithms:**

* **Summary of FCFS scheduling algorithms:**

It is a [non preemptive](http://www.moreprocess.com/process-2/non-preemptive-preemptive-scheduling-algorithm-purpose-of-a-scheduling-algorithm" \t "_blank) scheduling algorithm. Here processes are kept in a queue and are executed one by one. Whenever a new process arrives it is kept at the end of the queue. So, this is the last process in the queue. Now, when a new process arrives after this process, the new process becomes the last process in the queue. When a process which is running is blocked the next process in the queue is run and the blocked process (when it is ready) is placed at the end of the queue. FCFS is usually used in batch systems.

**Advantage:**

The advantage of FCFS algorithm is that it is very easy to implement. The disadvantage of FCFS algorithm is that since it is non preemptive, CPU usage can be wasted in some situations.

**CODE**:

This is handled in method public static int do\_dispatch(). This method first check if readyQueue is empty or not. If it is empt , it cannot Dispatch Thread because readyQueue is Empty. It then check MMU.getPTBR() is null or not . We put new threads to head of the queue and scheduled it from tail of the queue. If context switch takes place then we move running thread to tail of the queue and pick it from there only. So we are not preemptive the running thread.

thread2 = (ThreadCB)readyQueue.removeTail();

thread2.setStatus(ThreadRunning);

* **Summary of Round Robin scheduling algorithms**

It is a preemptive scheduling algorithm. In this algorithm, every process is considered as equal. A time interval to run known as quantum is assigned to every process. A queue is maintained with quantum for each process. After a process has finished its quantum, it is placed at the end of the queue and a new process is run from the queue. Round Robin scheduling algorithm is used for personal computers and servers.

**Advantage:**

Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks.

**CODE:**

This is handled in method public static int do\_dispatch(). This method first check if readyQueue is empty or not. If it is empt, it cannot Dispatch Thread because readyQueue is Empty. It then check MMU.getPTBR() is null or not . We put new threads to head of the queue and scheduled it from tail of the queue. If context switch takes place then we move running thread to head of the queue and pick it from tail of the queue. Therefore, we are preemptive the running thread. This is different from FCFS as we use to put it to tail of the queue in that case

thread2 = (ThreadCB)readyQueue.removeHead();

thread2.setStatus(ThreadRunning);

**Compare of scheduling algorithms:**

* The CPU Utilization is very high for FCFS rather than Round Robin. It is because of the fact that in Round robin due to cintext switching the CPU is idle for sometime in between this context switching.
* The throughput of the Round Robin Algorithm is high as compared to FCFS as there is no starvation in Round Robing Algo because of the context switching.
* We will use round robin if it is desirable to allow long running processes to execute while not interfering with shorter ones, with the side effect that order of completion is not guaranteed. Round Robin can suffer if there are many processes in the system, since it will take longer for each process to complete since the round trip is longer.
* If you do need a guaranteed order of completion, FCFS is a better choice but long running processes can stall the system. However, each process is given the full attention of the system and can complete in the fastest possible time, so that can be a benefit.
* In the end it really does come down to not necessarily design but need. Do I need semi-synchronous execution or do I need in-order execution? Is it to my benefit for processes to take longer but compute in sync or will I be better off if everything executes as fast as possible? The needs of the system dictate the model to use.