In this report, you should first explain your understanding of the implemented algorithms. Then, briefly summarize all experiments you have done with your schedulers on the given CPU workload. Finally, report the optimal parameters you have found for RR and FBQ schedulers and give your reasons to justify these optimal parameters.

**Introductions/Summery:**

The three algorithms implemented in this project seems to be very easy by reading the mechanism of scheduling in the textbook, but in fact, when it comes to specific code implementation, we need to consider various conditions that might affect our top-level algorithm, which means, we need to do a lot of slight modifications during implementations.

**Algorithms:**

For the FCFS algorithm, my general idea of implementation is as we increase the clock time, which is the simulated by code, from 0 to a certain integer number, at each time spot, we need to

1. allocate CPUs to processes when CPUs is available.
2. enqueue processes to waiting queue when the CPU burst of current process running in one of our 4 processors is over.
3. enqueue processes to ready queue when I/O burst is finished.

This is the general workflow for us to implement FCFS algorithm, note that the main feature of FCFS algorithm is that when each CPU is allocated to a process, it will be released until the current CPU burst is finished, called non-preemptive. As mentioned in the textbook, this kind of algorithm will lead to convoy effect when a process with a large CPU burst, as a result, scheduling criteria in First Come First Serve algorithm has following characteristics, throughput can be low, since the CPU might hold process for a long time and there will be no context switch, turnaround time and waiting time can be high, additionally, there is no starvation.

For the Round Robin algorithm, the idea of implementation can be modified slightly from FCFS algorithm. In FCFS, the processes are move from CPUs to waiting queue only when current CPU burst is finished. However, in RR algorithm, given a fixed time quantum, if the processes at each CPU have run out of given quantum as clock time increases and the burst is not finished yet, here is where preemption needs to be performed, that is, the processes need to be push to the tail of ready queue and wait next available CPU allocation.

Scheduling criteria in RR are highly depend on the time quantum. If it is too small, then too many context switches will be generated and slowing the execution of processes accordingly. However, the time quantum cannot be too large as it will be pretty much FCFS as a result.

For multilevel feedback queue algorithm. The idea is:

1. Divide the original ready queue into three queues with different priority, CPUs allocate processes from higher priority to lower priority among three queues, the processes in the lower-priority queue will not be allocated until higher-priority queue is empty.
2. Processes coming from different queue determines the running time that they will stay in the CPUs, namely, certain time quantum for processes dequeue from first two queues. For the processes coming from the lowest-priority queue, they follow the FCFS algorithm, that is, they will release the CPUs until the rest of burst finished.
3. During the CPU bursting procedure in each CPU, if there are incoming processes from higher priority queue, then the current process running in the CPU from lower-priority queue need to be pre-empted, namely, give CPU away to the incoming processes. In my implementation, I push those processes to the head of its original queue.

By this algorithm, initially, if the processes take a long CPU burst, at least it will be allocated once, then it will be move to lower-priority queue, however, if it waits too long in the lower-priority queue, it will be move back to higher-priority queue again.

**Experiments Summery:**