**Comparison of Scheduling Algorithms**

Thomas Anderson & Jonathan Lamoureux, Stetson University

CSCI301, Operating Systems

Hala ElAarag

March 27, 2022

**Round Robin**

Round Robin Scheduling is one of the original scheduling algorithms and is still used to this day (for good reason). It is known as one of the fairest algorithms as every process has a chance at their time in the CPU, meaning it prevents starvation of processes. The program that we wrote starts off by generating a random quantum number. This number determines how long a process will run before moving on to the next process.

Performance Measures:

* Average waiting time
* Average turnaround time
* Quantum value

Round Robin is one of the oldest scheduling, and it is still around due to its preemptive nature and avoidance to starvation. We found that it is one of the more efficient scheduling techniques of these four that we tested.

**Lottery**

Lottery was a pretty concept to understand. The program that we wrote has the user delve out tickets to 3 different processes which then run and increment when each one was called. The program will then spit out the amount of time the process was called compared to how many tickets the process had using a random number generator. Tickets would be based on how many the user wanted to give, and the program would then run 100 times to get a general sense of how many times each process would run.

Performance Measures:

* Time taken for:
  + 100 processes: .003s
  + 10000 processes: .004s
  + 100000 processes: .013s
  + 1000000 processes: .101s
  + 10000000 processes: .534s

Overall, Lottery Scheduling seems to be an efficient way to schedule. A big component of its success is the ability to prevent starvation as all processes have a chance of being picked, even if it’s only with one ticket.

**Fair Share**

Fair Share scheduling uses three randomized parameters for the implementation; a random number of users from 3 to 7, a random number of processes from 1 to 4, and a randomized burst time from 1 to 20. The arrival times are all initialized to 0, and the initial priority is decided by the user and process IDs for simplicity. The average wait time is higher if one users has significantly more processes than the others, but if the amount of processes are evenly dispersed, then the wait times are fairly low since fair share allocates the same amount of CPU time to each user up until that user’s processes are completed.

Performance Measures:

* Average Wait Time
* Average Turnaround Time

**Multilevel Feedback**

MLF (Multilevel Feedback) is the most complicated of the four scheduling algorithm, so uses the least amount of randomly inputted variables in our implementation. The scheduler uses four different queues, starting with 3 round robins with quantum times of 2, 4, and 6, and then finishing any remaining processes with a FCFS queue. The implementation stores a randomized number of processes from 5 to 10 inside of a struct, along with random burst times from 1 to 20 and arrival times from 0 to the total number of processes.

As seen in figure 1.0, each process will execute through one queue at a time, and carry over to the next queue if the process is not completed in the current one. The order is RR1 (Round Robin with quantum = 2), RR2 (Round Robin with quantum = 4), RR3 (Round Robin with quantum = 6), and FCFS (First Come First Served).

Performance Measures:

* Average Wait Time
* Average Turnaround Time

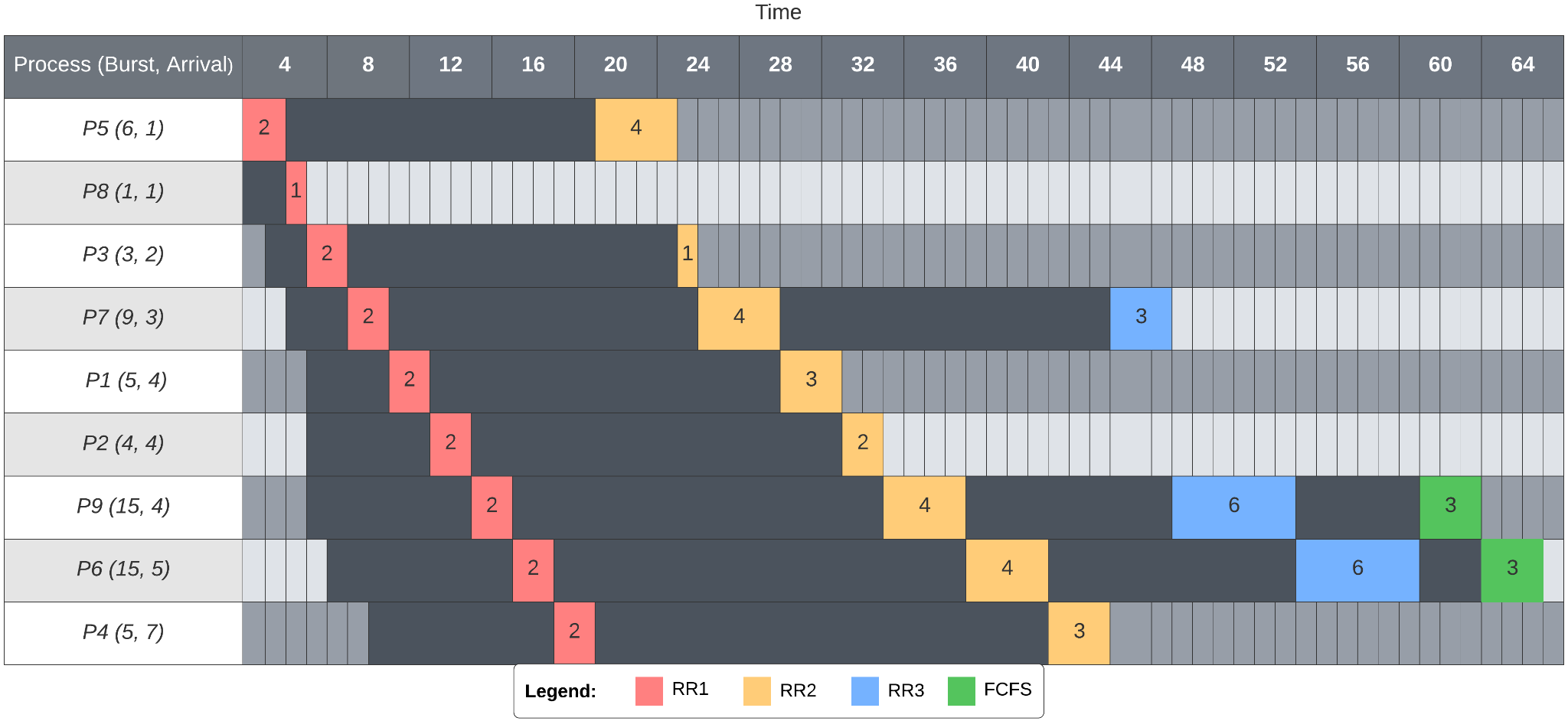


Figure 1.0: Gantt chart of MLF scheduler with 9 processes

**Comparison of Schedulers**

For the comparison, each scheduler is given a static number of processes. The burst times for each process is still a random number between 1 and 20 for each scheduler, so the average wait time can vary significantly, yet the average wait time when comparing the three schedulers still ended up having similar wait times for each batch size as seen in Figure 2.0. It can be gathered from this that a far more specific method of comparison would be required to get an accurate comparison of which scheduler is more efficient than the other, that cannot be feasibly done in such a short period of time.

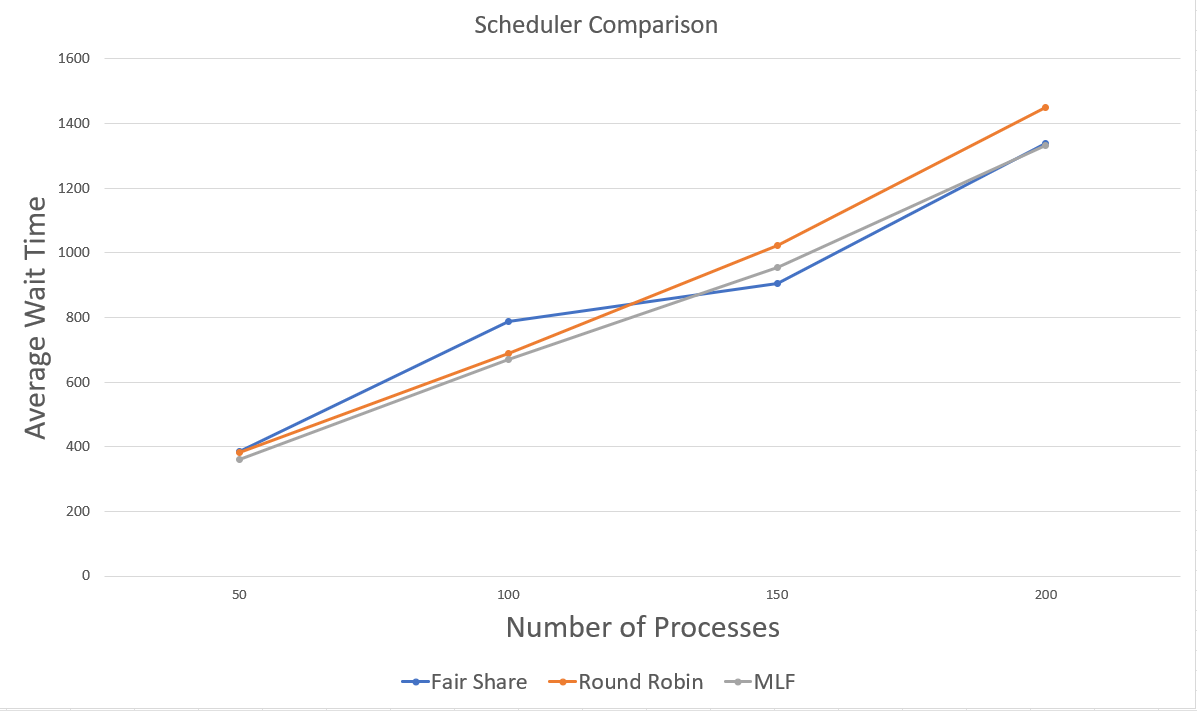


Figure 2.0: Scheduler Comparison