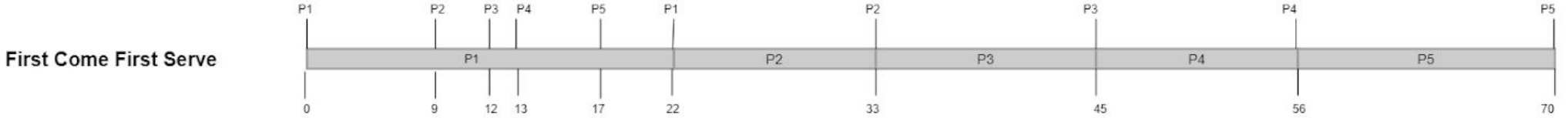
Part 1

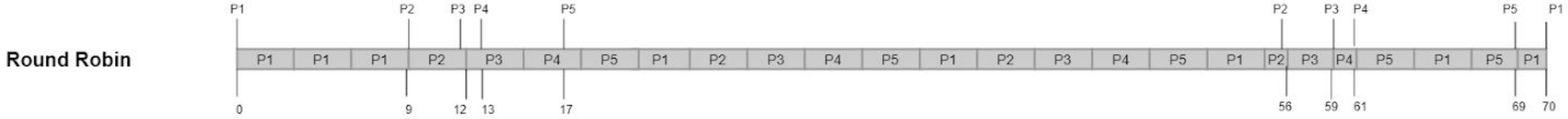
a) A possible event that can cause the process to abandon the running state is the arrival of a process with a higher priority. When the algorithm used is a preemptive priority based algorithm, if a higher priority process arrives, any lower priority process will be forced to abandon the cpu. Another event that would cause it to leave in a similar manner would be if the time slice of the round robin scheduling algorithm runs out, the current running process to also abandon the cpu. With both of the events discussed above, after being forced to abandon the processes will transition from the running state to the ready state, and will resume waiting in the ready queue. Another reason to abandon the cpu would be if the process finishes, which would result in it being terminated. Another event that would cause the process to abandon the cpu is a i/o interrupt. The process is sent to the waiting state, for the i/o event to finish and then will be back in the ready state, back in the queue.

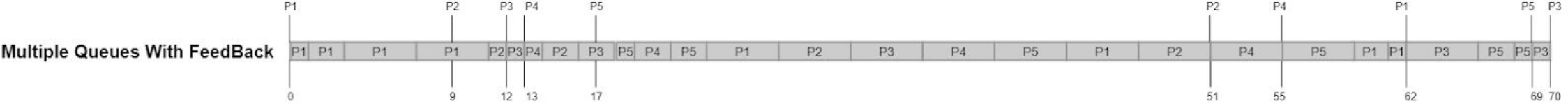
b) User threads are managed primarily by user applications through a library and do not have any kernel support as the kernel does not even know that these threads exist.On the other hand, kernel threads are directly managed by the system's Kernel. All Scheduling for user threads is done by the applications while Kernel threads are scheduled through the thread itself. User threads hold the advantage when it comes to running processes that aren’t vital to system operations, such as user applications, since thread switching occurs without any major overhead and the scheduling is application specific allowing more efficient selections of algorithm, with the only issue being that with any system call, user threads inside a blocked process will also all be blocked. Kernel Threads hold the advantage when performing processes vital to the operating system, as they can split up the threads of a single process on separate processors, with blocking only occurring on a thread basis over potentially entire processes being blocked. The thing holding back Kernel level threads is that any thread switching done must be done through the Kernel which causes a lot of slow down.

C)

First occurrence of process tag is when it is received, second is when it finishes.







Average turnaround time:
FCFS:

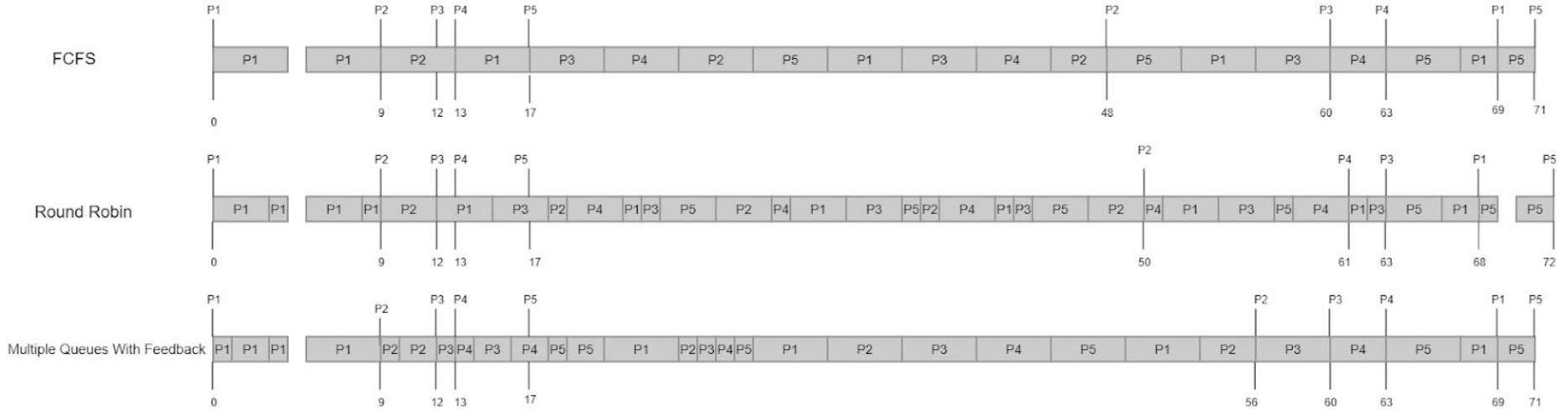
(22 + (33-9) +(45-12) +(56-13) + (70-17))/5 = 35

Round Robin:

(70 + (56-9) +(59-12)+ (61 -13) + (69 -17))/5 = 52.8

Multiple Queues With FeedBack
(62 + (51 -9) +(70 -12) + 55 -13) + (69 -17))/5

D)



Average turnaround time:
FCFS:
((69+ (48-9) + (60-12) +(63-13)+(71-17))/5 = 53.8

Round Robin:
((68+ (50-9) + (63-12) +(61-13)+(72-17))/5 = 52.6

Multiple Queues With Feedback:
((69+ (56-9) + (60-12) +(63-13)+(71-17))/5 = 53.6

E)
(i) FCFS can be considered to favor long processes as short processes are severely affected by longer processes, with long processes arriving first causing short process turnaround to inflate, while on the other hand long processes would not be severely affected by a short process going first.

(ii) Round Robin is the most balanced out of the three methods, with each process being given an equivalent time slice to run therefore none being unfairly biased towards. This can be seen in the results with all of the processes finishing around the same time.

(iii) Multiple Queues with feedback on the other hand favours shorter processes as at the beginning new processes will be higher priority, thus allowing shorter processes to finish, while longer ones get stuck in the low priority queue, forcing them to wait for the newer processes to finish.

F)

First Fit

	122		105			203		90	
102		102		102	739	203	536	90	446
205	122	83		83	0		0		0
43		43		43	0		0		0
180		180	105	75	0		0		0
70		70		70	0		0		0
125		125		125	0		0		0
91		91		91	0		0		0
150		150		150	0		0		0

Best Fit

	122		105		203		90	
102		102		102		102		102
205		205		205	203	2		2
43		43		43		43		43
180		180		180		180		180
70		70		70		70		70
125	122	3		3		3		3
91		91		91		91	90	1
150		150	105	45		45		45

Worst Fit

	122		105			203		90	
102		102		102	739	203	536	90	446

205	122	83		83	0		0		0
43		43		43	0		0		0
180		180	105	75	0		0		0
70		70		70	0		0		0
125		125		125	0		0		0
91		91		91	0		0		0
150		150		150	0		0		0