Unified Solution for all OS Spring 2024 Sections to grade Midterm # 2 exam

Q1. [6 marks]

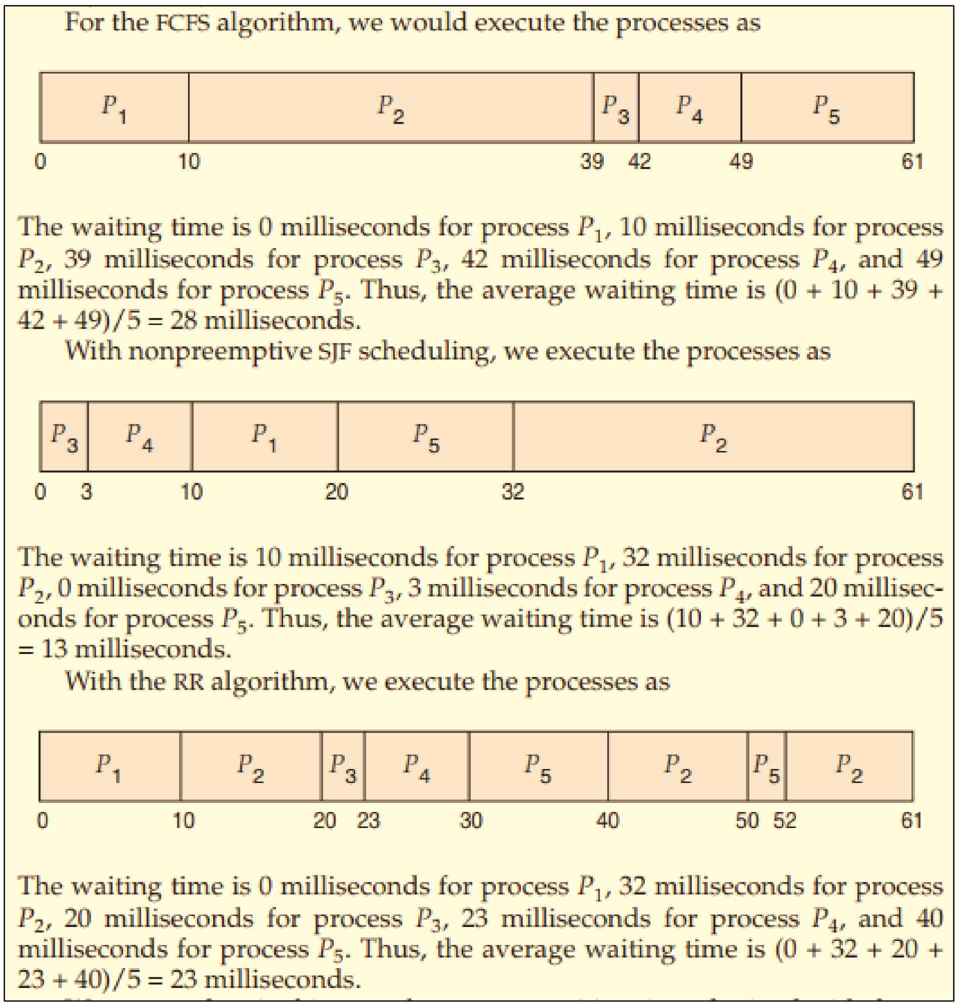
a) Consider the following five processes. All arrive at time zero, in the order listed, with the length of the CPU burst given in milliseconds:

```
Process:    p1, p2, p3, p4, p5
Burst Time: 10, 29, 3, 7, 12
```

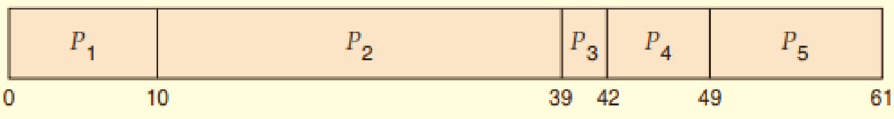
Consider the First Come First Serve, non-preemptive Shortest Job First, and Round Robin with time quantum ten milliseconds scheduling algorithms for this set of processes. *Note: Perform all calculations as per the textbook.*

i. Workout the average waiting times for all three algorithms along with their respective Gantt charts. [2]

See figure for solution.

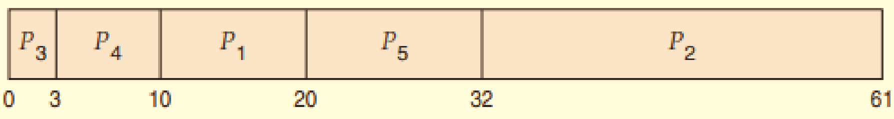


For the FCFS algorithm, we would execute the processes as



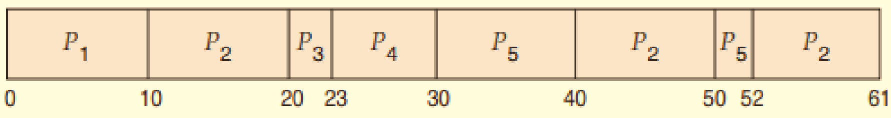
The waiting time is 0 milliseconds for process P_1, 10 milliseconds for process P_2, 39 milliseconds for process P_3, 42 milliseconds for process P_4, and 49 milliseconds for process P_5. Thus, the average waiting time is $(0 + 10 + 39 + 42 + 49)/5 = 28$ milliseconds.

With nonpreemptive SJF scheduling, we execute the processes as



The waiting time is 10 milliseconds for process P_1, 32 milliseconds for process P_2, 0 milliseconds for process P_3, 3 milliseconds for process P_4, and 20 milliseconds for process P_5. Thus, the average waiting time is $(10 + 32 + 0 + 3 + 20)/5 = 13$ milliseconds.

With the RR algorithm, we execute the processes as



The waiting time is 0 milliseconds for process P_1, 32 milliseconds for process P_2, 20 milliseconds for process P_3, 23 milliseconds for process P_4, and 40 milliseconds for process P_5. Thus, the average waiting time is $(0 + 32 + 20 + 23 + 40)/5 = 23$ milliseconds.

ii. Name the algorithm that will perform best based on your calculations. Explain reasons for your selection. [1]

SJF [0.5]. When there are large number of processes, a process will wait long for its turn in both FCFS and RR scheduling. While SJF quickly executes processes having smaller CPU burst times giving way to others [1.0].

b) How a multilevel feedback queue (MLFQ) scheduling algorithm separate processes in different priority queues? [1]

A new process is assigned in the highest priority queue. Processes exhausting their quantum in the top queue are moved to the lower queue. However, processes stuck in low priority queues might be reassigned to higher queues to prevent starvation.

c) Recall the threading concepts covered from the textbook. Now, **compare** the following:

i. Concurrent execution vs Parallel execution [1]

Concurrent execution refers to the execution of multiple tasks in different time periods on a single CPU core [0.5]. In parallel execution, multiple tasks run at a time due to the availability of multiple CPU cores [0.5].

ii. Task parallelism vs Data Parallelism [1]

Task parallelism involves dividing a task into smaller independent subtasks that perform different operations on same/different data [0.5]. In data parallelism, many independent subtasks perform the same operations on different subsets of data [0.5].

Unified Solution for all OS Spring 2024 Sections to grade Midterm # 2 exam

Q2. [9 marks]

a) Suppose you have written a multithreaded C program using Pthread library that creates ten threads to solve a problem. You also determine that 20% of execution remains serial. Now answer the following: [1.5 + 1.5]

i. **How much speedup** do we get if we run the program on a computing unit having ten cores?

$$S = \frac{1}{0.2 + \frac{0.8}{10}} = 3.75$$

S is speedup, 0.2 is serial portion, 1-0.2 is parallel portions and N =10 (cores) **[0.75]**

ii. **Explain** what happens if the same program runs on a single CPU, single core system.

They run by taking turns as per CPU schedular policy **[1.0]**. Execution time increases as each waits its turn and the added overhead of multiple context switches **[0.5]**.

b) Write a **C program** using the Pthread library that performs a parallel assignment of random numbers to an integer array of three hundred elements using three threads. *Note: There will be no partial award if incorrect syntax or logic.* [1.5 (syntax) + 1.5 (logic) = 3]

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define ARRAY_SIZE 300
#define NUM_THREADS 3

int array[ARRAY_SIZE];

void *assignRandomNumbers(void *thread_id) {
    long tid = (long)thread_id;
    int start_index = tid * (ARRAY_SIZE / NUM_THREADS);
    int end_index = start_index + (ARRAY_SIZE / NUM_THREADS);
    for (int i = start_index; i < end_index; i++) { array[i] = rand() % 100; }
    pthread_exit(NULL);
}
int main() {
    pthread_t threads[NUM_THREADS]; int rc; long t;
    srand(time(NULL)); // Seed for random number generator
    for (t = 0; t < NUM_THREADS; t++) {
        rc = pthread_create(&threads[t], NULL, assignRandomNumbers, (void *)t);
        if (rc) {
            printf("ERROR: return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }
    // Wait for all threads to finish
    for (t = 0; t < NUM_THREADS; t++) { pthread_join(threads[t], NULL); }
    exit(0);
}
```

c) Consider the situation illustrated in Figure 1 where two processes, running in parallel, try to execute the fork () system call at the same time. During execution, both fork () system calls request pid, which involves the identifier **next_available_pid**. Now, explain the synchronization problem shown in this diagram by answering the following questions: *Note: No coding required.* [1.5 + 1.5]

i. How do they get the same value? Explain.

It happens due to race condition on global variable **next_available_pid [0.5]**. While running in parallel on two cores, each reads the previous values 2614, increments it to 2615 and uses and writes it to next_available_pid **[1.0]**

ii. Discuss a solution using which they get a unique value?

They need mutual exclusion during their execution **[0.5]**. This means one needs to busy wait while the other updates the global variable next_availalbe_pid. However, the waiting process should be allowed to update immediately. **[1.0]**.

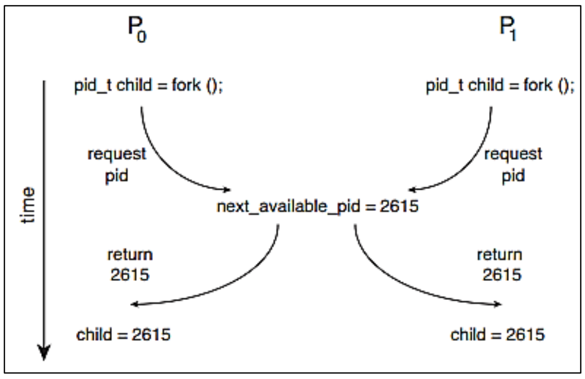


Figure 1: Q2 part (c)