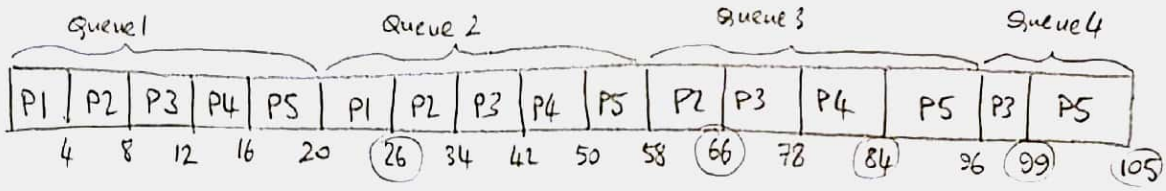
Özgür Taylan ÖZCAN
54284

02.04.2018

COMP 304 ASSIGNMENT #2

1) a)



$$\text{Avg. Turnaround} = \frac{26+66+84+99+105}{5} = 76$$

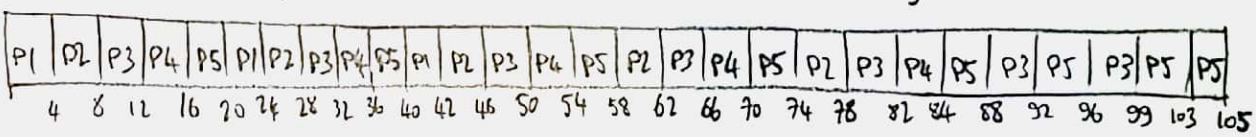
b)

$$T_{P1} = 26 + 5 = 31$$
$$T_{P2} = 66 + 10 = 76$$
$$T_{P3} = 99 + 14 = 113$$
$$T_{P4} = 84 + 12 = 96$$
$$T_{P5} = 105 + 15 = 120$$

$$\text{Avg. } T = \frac{31+76+113+96+120}{5} = 87.2$$

c)

Gaunt chart for round robin without counting context switch overheads:



Multi level method had 15 context switches. This method has 27 context switches.

So, round robin scheduling is worse in terms of context switch overhead.

2) a) count is the shared variable. So, without synchronization, it would involve in the race condition.

b) race condition may occur in two cases:

1) if one process increases the count while the other decreases.
(line 17) (line 11)

2) if one process checks if available.resources<count while the other increas
(line 8) (line 17

c) Fix of race condition using semaphore:

```
typedef struct{
    int available_rec;
    Struct process xlist;
} SEMAPHORE;
SEMAPHORE license;
wait(license){
    license.available_rec--;
    if (license.available_rec <0){
        license.list.add(current_process);
        block();
    }
}
signal(license){
    license.available_rec++;
    if(license.available_rec <=0){
        process p = license.list.removelast();
        wakeup(p);
    }
}
```

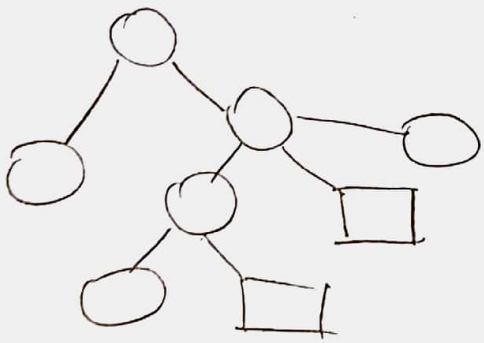
3)

```
Monitor Dentist{
    int empty_chairs = N;
    Condition wake_up, treatment_full, treatment_not_full;

    get_dental_treatment(){
        if (empty_chairs==0)
            leave;
        else if (dctor is sleeping)
            signal(wake_up);
        empty_chairs--;
        wait(treatment_not_full);
        empty_chairs++;
    }

    get_next_patient(){
        if (empty_chairs == N)
            wait(wake_up);
    }
    finish_treatment(){
        signal(treatment_not_full);
    }
```

4) a)
```
pthread_t pthreadt;
pid_t pid;

pid = fork();
if (pid == 0) {
    fork();
    :
    pthread_create(&pthread, NULL, funct, NULL);
    :
}
fork();
```

- num of processes created = 5 (without the initial process)
- num of threads created = 5+2=7 (without the initial process/thread)

b)



c)
```
pid = clone();
if (pid==0) {
    clone();
    :
    clone();
    :
}
clone();
```

5)

	Max	Available	
P0	2	4	✓
P1	2	4	✓
P2	2	4	✓

This system is deadlock free. Because there are 3 processes and 4 recourses, which means a process is able to get 2 recources, and it will return its recources when done.