Assignment 2

4.

A variation of the round-robin scheduler is the **regressive round-robin** scheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted. (The time quantum for a process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

Ans:

Regressive round robin favors CPU bound types of a process. This is because those types of processes will be rewarded up to 5 times by scheduler. The reasoning behind this is that if a CPU bound process consumes entire slice it had it was doing computation and processor was “active” the entire time. It is possible that such CPU bound process could finish soon so it is rewarded by scheduler.

I/O bound processes are not punished with lowered priorities, but if they spend entire time waiting for I/O device they should get a smaller slice. Idea here is that such process depends on speed of I/O device which is unknown to processor cores.

Moreover, it favors CPU bound types of a process since those processes activity consume CPU time.

5. A program is decomposed into 4 sub-processes. All the sub-processes can execute in parallel. However, some processes can execute only after some other processes can finish execution. This is described by the following precedence constraints:

* Process 3 can start only after processes 1 and 2 finish execution
* Process 4 can start only after 2 finish execution.

Provide a pseudo code with semaphores to accomplish this.

Ans:

Semaphore is an integer variable which is used in mutual exclusion manner by various concurrent cooperative process to achieve synchronization. In binary semaphore the values of semaphore only can be 1 or 2. Here, process 3 can start only after processes 1 and 2 finishes. And process 4 can start only after process 2 finish execution. So, it Can be

1. P1>P2>P4>P3

It can be process P1 & P2 comes first. By chance P2 finish its execution & P1 still in critical section while executing. As P2 finishes execution P4 comes in ready queue & P3 become blocked. As there is priority which is P4 start just after P2 finish. Then after that P4 & P1 both finishes. So, the 1st condition become true that process 1 & 2 both finish its execution. And So, Process 3 start then.

1. P2>P4>P1>P3

It also can be happening if we take, P2 process first. For condition 2, P4 start executing just after P2 finish executing & P3 will become block. Then after a wait operation we can unblock P3 & then execute P1. After P1 & P2 both finishes, P3 can start executing for condition 1.

Now we can implement these by using counting or binary semaphore. Here, I am using binary semaphore for simplification. The pseudo code & algorithm are in follows:

1. For 1. P1>P2>P4>P3

I will use 3V(Signal/Up), 1P(Wait/Down), 2V(Up)

Assume, S1 & S2 are 2 binary semaphores.

Int S1=0, S2=0;

**For P1**: Up(Semaphore S1) {

If(suspended list==0) //if suspended or block list empty

{

S1.value=1; //S1= from 0 to 1 means successful operation so P1 enter the critical section

}

else{

select a process from suspend list and wakeup(); //not required for process 1 as block list is empty at first

}

} now , p1 in CS

**For P2**: Up(Semaphore S2) {

If(suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 0 to 1 means successful operation so P1 enter the critical section

}

else{

select a process from suspend list and wakeup(); //not required for process 2 as block list is empty

}

} now , p2 in CS

**For P3**: Up(Semaphore S2) {

If(suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 1 to 1 means unsuccessful operation so P3 will enter the suspended list

}

else{

select a process from suspend list and wakeup(); //not come in this

}

} now , p3 in block list

**For P4**:

Down(Semaphore S2){

If ( S2.value==1)

S2.value=0; //successful operation

}

else{

Block this process and place in suspended list;

Sleep(); }

} So, now P4 is in CS

**For P3**: Up(Semaphore S2) {

If(suspended list==0) //if suspended or block list empty

{

S2.value=1; //S2= from 0 to 1 means successful operation so P3 will enter the CS

}

else{

select a process from suspend list and wakeup(); //not come in this

}

} now , p3 in CS

So, all the processes execution successfully like that.

Moreover, if we want the 2nd option P2>P4>P1>P3 which is also correct. We have to apply 1V,1P, 1V,1P,1V which can be complex.

For P2: up(semaphore S2);

For P3 block: down(semaphore S1)

For P4: up(semaphore S1)

For P1: down(semaphore S2)

For P3: up(semaphore S1)

The codes are same. So this is how we can implement the priority based process scheduling by using semaphore.