Homework3

Name Yu Yang   Student ID 892449550

5.3

Busy waiting: While a process is in its critical section, any other process that tries to enter its critical section must loop continuously.

Other kind of waiting: a process could block itself, be put in a wait queue and be awakened in the future.

Busy waiting can be avoided, but it requires overhead of putting a process in into a wait queue and waking it up again.

5.4

In a single-processor system, a single cpu is shared among many processes. Busy waiting of spinlock wastes CPU cycles that some other process might be able to use productively.

In a multiprocessor system, one thread can “spin” on one processor while another thread performs its CS on another processor.

5.10

If a user-level could disable interrupts, it can block all other processes if there is only one processor.

5.11

Disabling Interrupts on the processor could not influence processes executed on other processors. So it can not guarantee mutex.

On the other hand, disabling interrupts on every processor can be a difficult task and seriously diminish performance.

5.23

```
Int gate = 0;// global shared

Typedef struct{
        Int value; //available num
        Struct process *waiting_queue;
} semaphore

Wait(semaphore *s){
        While(test_and_set(&gate));//only the first process getting here,
                                             //or when other process get a semaphore,
                                             // a process could enter
        If(s->value <=0){
                Add this process to s->wating queue;
                Block();
        }else{
                S->value --;
                gate = 0;//this process leaves wait, let others getting through the gete
        }
}

Signal(){
        While(test_and_set(&gate));// same reason as in wait()
        If(s->value <= 0 && not_empty(s->waiting_queue)){
                Wakeup(S->waiting_queue.remove());
        }
        Else{
                s->value ++;
                gate = 0;
        }
}
```

5.28

Throughput is increased by allowing multiple reading processes and only one writing process: A process wishing to modify the shared data must request the lock in write mode. Multiple processes are permitted to concurrently acquire a reader–writer lock in read mode, but only one process may acquire the lock for writing, as exclusive access is required for writers.

This approach could cause writer starving.

Solution: 1. Avoid keep letting new readers go into CS: when a new reader comes when several readers are in CS and a writer is waiting, block the new reader until waiting writer finishes.

2.keeping track of the waiting time of every process.

When several readers finish or a writer finish, give access to the process has waited for the longest time.

5.29

The signal() operation in monitor resumes exactly one suspended process. If no process is suspended, then the signal() operation has no effect; that is, the state of x is the same as if the operation had never been executed. Contrast this operation with the signal() operation associated with semaphores, which always affects the state of the semaphore(++ operation of the semaphore value).

5.32

A file is to be shared among different processes, each of which has a unique number. The file can be accessed simultaneously by several processes, subject to the following constraint: the sum of all unique numbers associated with all the processes currently accessing the file must be less than n. Write a monitor to coordinate access to the file.

```
Monitor fileAllocator{
    Int sum;
    Static int limit;
    Condition x;
    Void accessFile(int process_num){
        While(sum + process_num >= limit)
            x.wait();
        sum += process_num;
    }
    Void release(int process_num){
        Sum -= process_num;
        c.signal();
    }
    Initialization(){
        Sum = 0;
        Limit = n;
    }
}
```

5.37

a. available_resources
b. available_resource -= count and available_resources += count(because -= and += are not atomic op)
c. use semaphore with initialized value of MAX_RESOURCES. All Ops on -= or += are modified to use semaphore.

6.2

Under nonpreemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.
Otherwise, if scheduling also takes place on:

1. When a process switches from the running state to the ready state (for example, when an interrupt occurs)

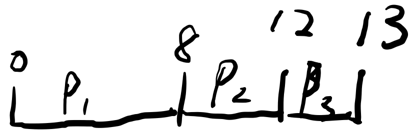
or

2. When a process switches from the waiting state to the ready state (for example, at completion of I/O)

It is preemptive.

6.3

A:



Average turnaround time = $\dfrac{(8-0)+(12-0.4)+(13-1.0)}{3} = 10.53$

B: nonpreemptive SJF

$$\frac{(8-0)+(9-1)+(13-0.4)}{3} = 9.53$$

C:

$$\frac{(2-1)+(6-0.4)+(14-0)}{3} = 6.87$$

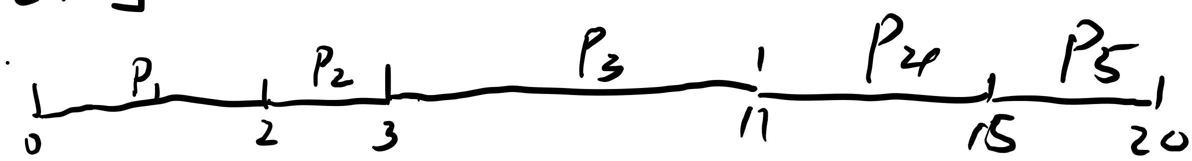
6.11

a. More context switch could decrease response time(like preemptive SJF, RR) but at the same time will require overhead, thus CPU utilization is decreased.
b. average turnaround could be decreased by using using preemptive SJF with the risk of make long burst process waiting forever in a heady-loaded system.

6.16

A:

FCFS:



SJF (assume nonpreemptive)

nonpreemptive priority

RR:

| P1 | P2 | P3 | P4 | P5 | P3 | P4 | P5 | P3 | P5 | P3 |

0 2 3 5 7 9 11 13 15 17 18 20

B:

	P1	P2	P3	P4	P5
FCFS	2	3	11	15	20
SJF	3	1	20	7	12
NonPrio	15	20	8	19	13
RR	2	3	20	13	18

C:

	P1	P2	P3	P4	P5
FCFS	0	2	3	11	15
SJF	1	0	12	3	7
NonPrio	13	19	0	15	8
RR	0	2	3+4+4+1 = 12	5+4=9	7+4+2=13

D:

SJF has the minimum one.

6.24

FCFS:

First come first serve means if short jobs arrive after long jobs, they will have long waiting time.

RR:

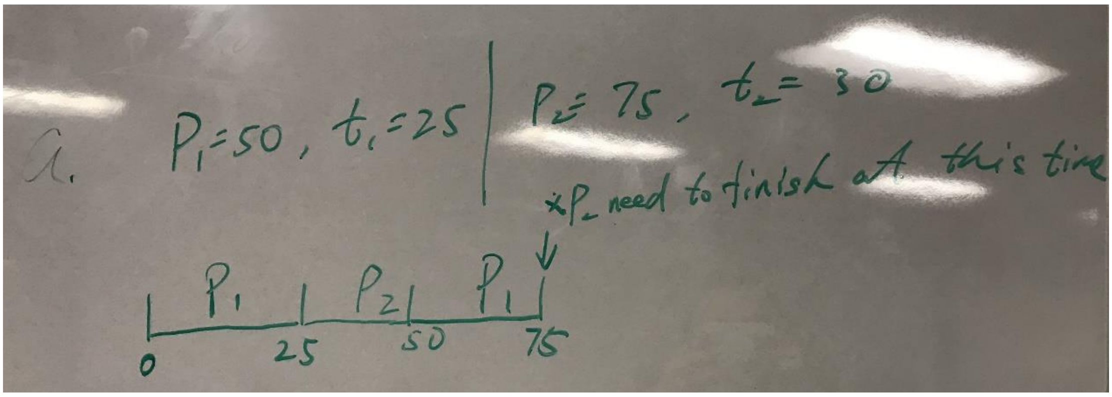
Every job has the same quantum time. So short jobs finish first with the assumption that the quantum is not to large.

Multilevel:

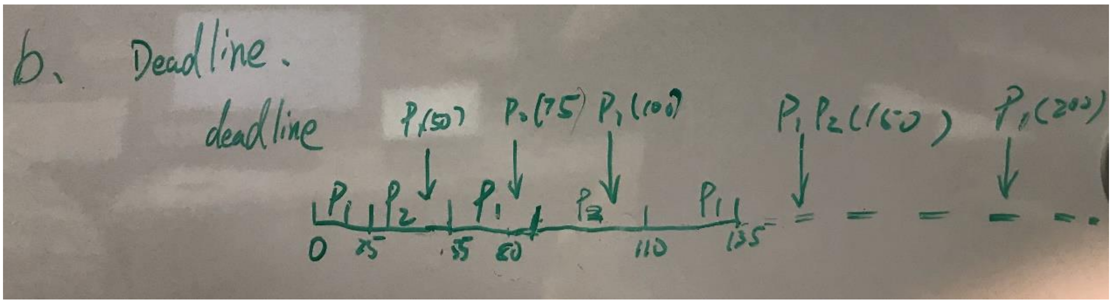
Similar to RR, put different jobs to different queue, but has mechanisms of promoting or degrading jobs

6.31

a. These two processes can't be scheduled using rate-monotonic. As it is shown in the graph, when p1 finishes at 75, p2 passes its deadline.



b.
As it is shown in the graph, the deadlines are p1(50),p2(75),p1(100),p1&p2(150),p1(200)······
This is also the order they are executed.



Virtual Box Guest Additions