Faculty of Computer Science
University of Indonesia

MIDTERM EXAM
IKI 20505 - OPERATING SYSTEMS

November 5, 2010

There are 5 questions in this exam. You have 120 minutes to solve all of the questions. Write your answer on this exam sheet. Each question has the same weight 20% of the total mark. Using calculator is forbidden in this exam. Only stationery, One note, and an exam sheet may exist on your table. Please clearly answer the questions by using pen or pencil. One blank paper is provided for the scratch work. Left the exam sheet on the table if you have completed the exam. You may do the questions in any order as you wish. Please use your time wisely!

Name : _____

NPM : _____

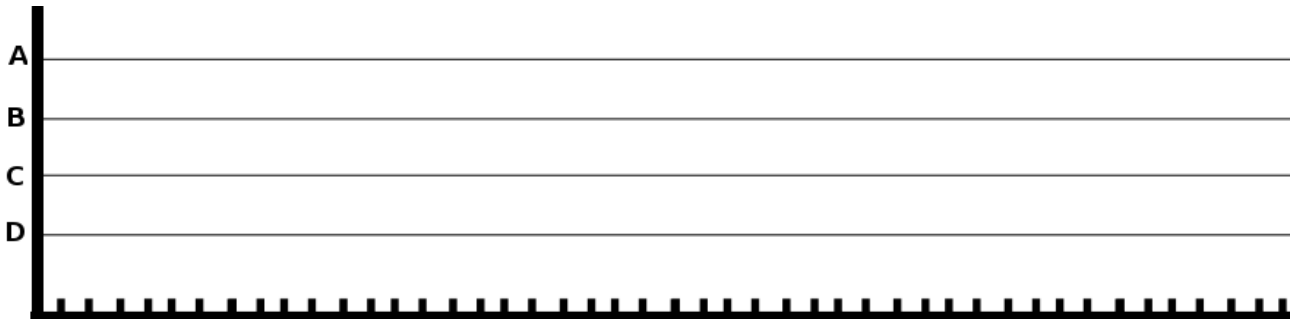
SCORE : _____

Question 1 : PROCESS STATE

There exists four processes, A(90: 150: 7), B(80: 100: 29), C(70: 50: 68), D(60: 0: 131); [where P(X:Y:Z) means P=process; X=I/O Wait (%); Y=arrival time; Z=CPU time] with this following CPU utilization table above. Please draw a process/time relation at the diagram above.

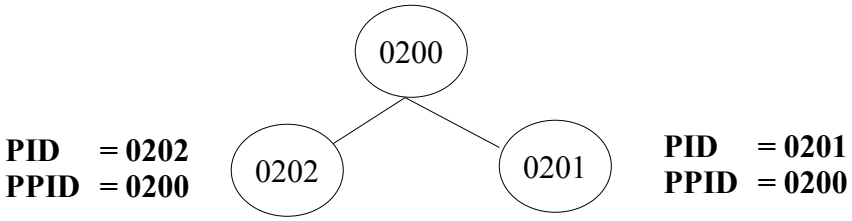
	Multiprogramming Combination (%)														
	A	B	C	D	A+B	A+C	A+D	B+C	B+D	C+D	A+B+C	A+B+D	A+C+D	B+C+D	A+B+C+D
CPU utilization per proses A	10	-	-	-	9.3	9.3	9.2	-	-	-	8.3	8.1	7.8	-	7
CPU utilization per proses B	-	20	-	-	19	-	-	18	17	-	17	16	-	15	14
CPU utilization per proses C	-	-	30	-	-	28	-	26	-	25	25	-	23	22	21
CPU utilization per proses D	-	-	-	40	-	-	37	-	35	33	-	32	31	30	28

Please draw a "processes/time relation" chart and calculate the starting time of all processes!



Question 2 : FORK

One process has one PPID. PPID stands for the Parent Process ID. Below are the relation between PID and PPID :



Look at the following C program below. If the initial PID of this program is **0401** and the PPID of **0401** is **0400**, print the **OUTPUT** of the program.

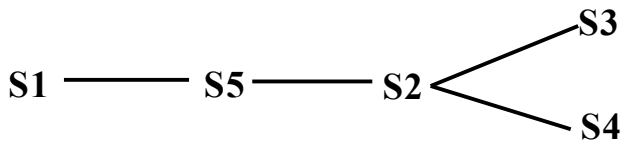
```
//Forkloop.c
//getpid()  : return the Process ID (PID)
//getppid() : return the Parent Process ID (PPID)

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#define STR "PID = %4.4d, PPID = %4.4d, Val = %4.4d\n"

int main()
{
      int count=0, loop=3, val=4;
      while (count != loop)
      {
            if(fork()>0) val--;
            wait(NULL);
            count++;
      }

      printf(STR, getpid(), getppid(), val);
}
```

Question 3 : SYNCHRONIZATION

There exist four processes P1(S1), P2(S2, S3), P3(S4), P4(S5), where Pn(Sx) means **Pn**=process number, **Sx**=Statement **x** belongs to **Pn**.

CPU will execute the process statements based on the following order :

1. CPU executes S1
2. CPU executes S5
3. CPU executes S2
4. CPU may execute S3, S4 or both at the same time



Note : when CPU executes S1 of P1, other process will wait. After executing S1, CPU executes S5 of P4 and so on

To achieve those execution order, write the solution code by using **acquire()** and **release()** method. Determine how many semaphore objects you will use. Implement your code on the empty space below :

//**initialize your semaphore objects here**			
P1	**P2**	**P3**	**P4**
`//P1 codes`	`//P2 codes`	`//P3 codes`	`//P4 codes`

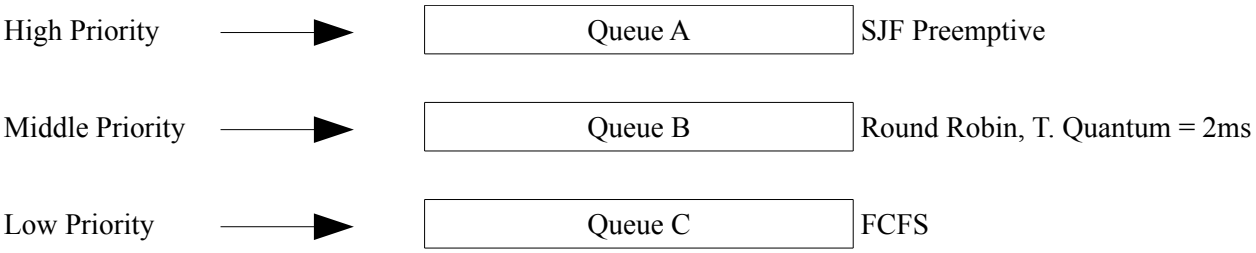
Question 4 : CPU SCHEDULING

An Operating System uses multilevel queue to schedule the processes execution. A multilevel queue scheduling consists of three queues ordered by priority level (high, middle, and low). Each queue has different scheduling algorithm.

- First Queue (Queue A) uses SJF Preemptive scheduling and eligible for process A1, A2, …, An
- Second Queue (Queue B) uses Round Robin scheduling with time quantum = 2ms and eligible for process B1, B2, …, Bn
- Third Queue (Queue C) uses First Come First Serve (FCFS) scheduling and eligible for process C1, C2, …, Cn

CPU will execute those queues under the following rules:

- CPU execute the queue based on its priority. If each queue is not empty, the queue that has high priority (Queue A) will be executed first. After that, CPU executes middle priority queue, then low priority queue.
- If high priority queue is empty, CPU executes process at other less priority queue
- If there is a process entering an empty high priority queue while CPU is executing process in other less priority queue, CPU must change its execution to high priority queue to service the process. CPU may move to other less priority queue if no process waiting in high priority queue.



Process	**A1**	**A2**	**A3**	**A4**	**B1**	**B2**	**B3**	**C1**	**C2**	**C3**
Arrival Time	0	2	8	11	2	4	9	3	5	7
Burst Time (ms)	4	2	1	4	5	4	3	7	8	2

Draw the gantt chart and determine the **total waiting time** of each queue. BE CAREFULL in writing the process starting time on each gantt chart.

Queue A (QA)

Total Waiting Time (QA) =

Queue B (QB)

Total Waiting Time (QB) =

Queue C (QC)

Total Waiting Time (QC) =

Question 5 : DEADLOCK

There exist four processes in the system. Total resources in the system are P(13), Q(19), R(15). The process sequence : <P2, P4, P5, P1, P3>

Processes	Allocation			Maximum			Available		
	P	Q	R	P	Q	R	P	Q	R
P1	2	2	1	11	16	14	2	7	3
P2	3	2	1	4	8	3			
P3	2	2	2	5	7	6			
P4	4	5	6	9	11	10			
p5	0	1	2	3	2	4			

By using deadlock avoidance, please specify :

1. The need table
2. Is the system safe or not ? Prove it!
3. What happen to the system if P3 ask one more resource R ? is it safe or not ?

Answer :