RAN

Faculty of Arts and Sciences *Department of Computer Science*

CMPS 272-Operating System Concepts

Fall 2004 – 2005, Mid-term (November 20, 2004)
Duration: 100 min

Profs. Leila Ismail and Walid Keyrouz

(1)

Student name Ruby Najjar

Student ID 200200417

Signature Ruby Najjar

Questions	***Points***	***Grade***
1. General (7 questions)	30	4
2. CPU scheduling (2 problems)	20	4
3. Synchronization (3 problems)	40	2.5
4. Deadlocks (1 problem)	14	3.5
Total	100	14

Please read the instructions below carefully before you start with the exam.

RAN

IMPORTANT INSTRUCTIONS

1. You MUST fill in your name, student ID, and your signature on the first page, and your initials on all the pages, including the first one.
2. You MUST explain your answers clearly to get points. Your handwriting MUST also be readible to be graded.
3. You MUST NOT look at your neighbor's sheet. This will be considered as a cheating attempt.
4. You MUST NOT talk during the exam. Any communication will be considered as a cheating attempt.
5. Any cheating will result in a zero on the test and possibly in failing the course and may also lead to disciplinary actions against you.
6. Use your time carefully. If you feel you are stuck, skip to the next question.
7. Use the back side of a page if the space provided for a question is not enough.
8. There is a total of 21 pages.

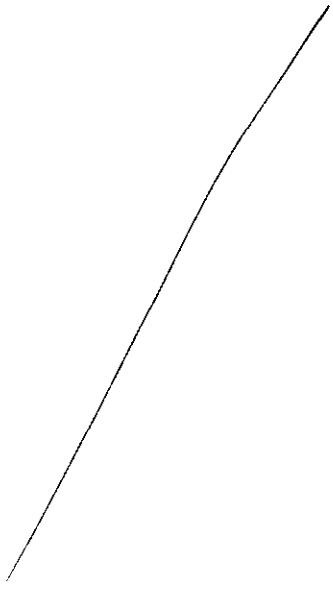
RAN

1.General

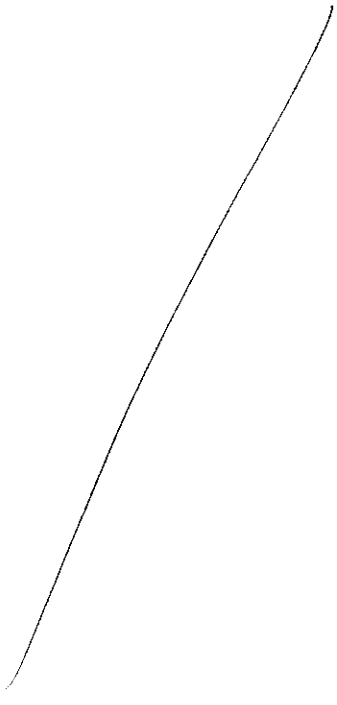
There are 7 general questions to solve.

1. Given two machines, M1 and M2: M1 uses DMA and interrupt-driven I/O; M2 uses polling-based I/O. Compare the I/O performance from a user's perspective in the following scenarios:

   a) The two machines are single-user single-tasking machines.



   b) The two machines are multi-user multitasking machines.



RAN

2. What is the purpose of a System Call? Give 2 examples.

A System call is used to ~~———————————~~

System calls are explicit requests to a the kernel made via software interrupts. System calls are interface given to the user mode process to access kernel ~~mode~~ or device specific data.

3. What are the advantages of multithreaded programming when compared to multiprogramming without mulithreading?

Multiprogramming without multithreading ~~usually needs to create a new process to deal with every~~ the parent process needs to create a child process to deal w/ every new processing that ~~comes~~ it has to compute. This new child process (not true!) has no shared resources with its parent and so creates an overhead that is not found when it comes to multithreading. 1

~~A th~~ Multiple threads, in multithreading, all share the same code, id, --- and have only a few resources particular to them. In addition, threads may be created prior to their need and put in a thread pool and picked out whenever needed.

RAN

4. What is the difference between a user-level thread and a kernel-level thread? Why does a process block when one of its user level threads issues a system call, whereas the process does not block when one of its kernel-level threads does the same?

A user-level thread is a thread created by a user program.

A kernel-level thread is a thread created by the operating system program.

The process blocks when one of its user level threads issues a system call but not in the case of a kernel thread for protection so that the user won't be able to take over.

10

5. What does the shell do? Can a user delete the shell and what happens to the operating system if a user were to delete the shell?

The shell takes commands from the user and executes them if possible (i.e.

? → there are restrictions on the commands a user can ask for).

X No, a user cannot delete the shell.
(this is one type of a command restriction)

If a user were to delete the shell, the operating system would be in control of the user since there would be no more restrictions as to what he could or couldn't do.

0.5

RAN

6. **What is the difference between the calls fork(), exec() and system()?**

fork() → creates a child process

exec() → executes ~~the~~ a certain program in the ~~child~~ process

system() →

7. **Briefly define a CPU-burst and explain its relevance to process scheduling.**

A CPU-burst is the time a certain process needs to use the CPU-scheduler, in order to finish its execution.

Its relevance to process scheduling is that, no matter what the algorithm for scheduling used, the scheduler needs to know the CPU-burst of each process in order to be able to schedule in the fashion stated by the algorithm, and hence, get the jobs done

RAN

2. CPU Scheduling

There are 2 problems to solve in this section.

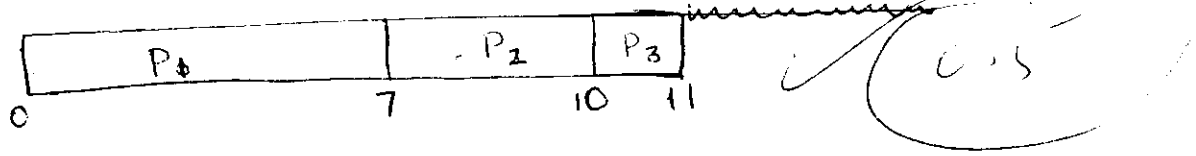
Problem 1

Suppose that the following processes arrive for execution at the times indicated in the table below and that each process will run for the given duration. In answering the questions, use nonpreemptive scheduling and base all decisions on the information you have at the time the decision must be made.

Process	Arrival	Burst Time
1	0	7
2	0.4	3
3	1	1

a) What is the average turnaround time for these processes with the FCFS and SJF scheduling algorithms? Draw the corresponding Gantt chart for each algorithm.

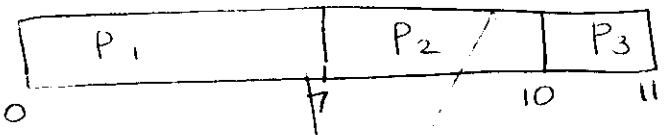
a.1) FCFS



Av. turnaround time

RAN

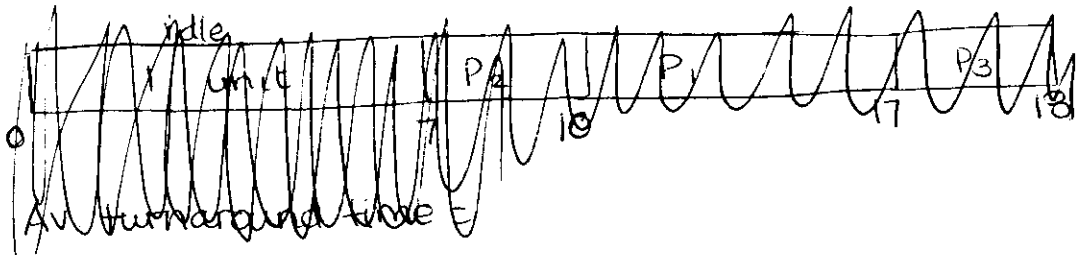
a.2) SJF



Av. turaround time =

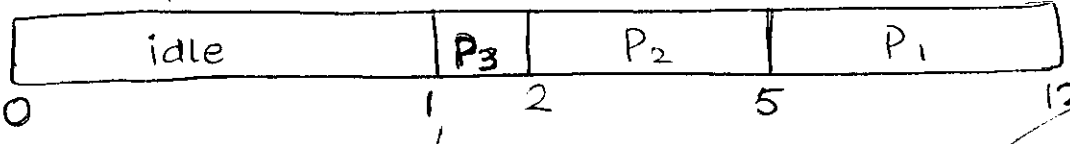
0

b) The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive soon. Compute the average turnaround time if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Draw the corresponding Gantt chart.



Av. turnaround time =

Assumption idle time is considered as waiting time.



Av. turnaround time =

1.5

RAN

Problem 2

Consider the following set of processes, with their arrival times, CPU-burst lengths, and priorities.

Process	Arrival	Burst Time	Priority
1	0	10	5
2	2	3	2
3	2	5	1
4	7	6	3
5	8	5	4

Compute the average waiting times for all the scheduling policies below and count the number of context switches. For scheduling policies (d) and (e), draw the corresponding Gantt chart.

(a) FCFS without preemption

Av waiting time = (0+10+13+18+24)/5

= 13

4 context switches

(b) SJF without preemption

Av. waiting time = (0 + 10 + 13 + 18 + 23) /5

= 12.8

4 context switches

RAN

(c) SJF with preemption

Av. waiting time = (19 + 2 ~~+5 + 10 + 15~~)/5

= 10.2

5 context switches

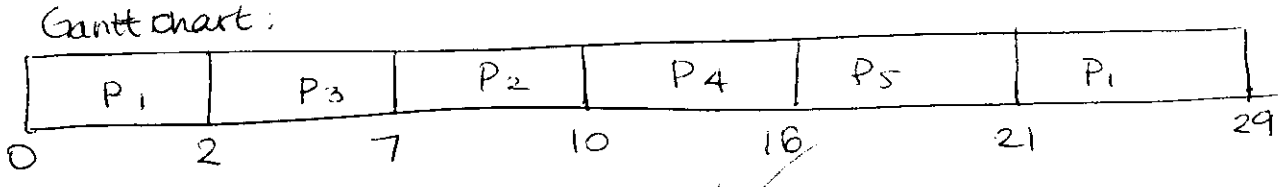
(d) Priority with preemption (highest priority is lowest number i.e. 1 here)

Av. waiting time = (0+2 + 7 + 10 + 16 + 21) / 5

= 11.2

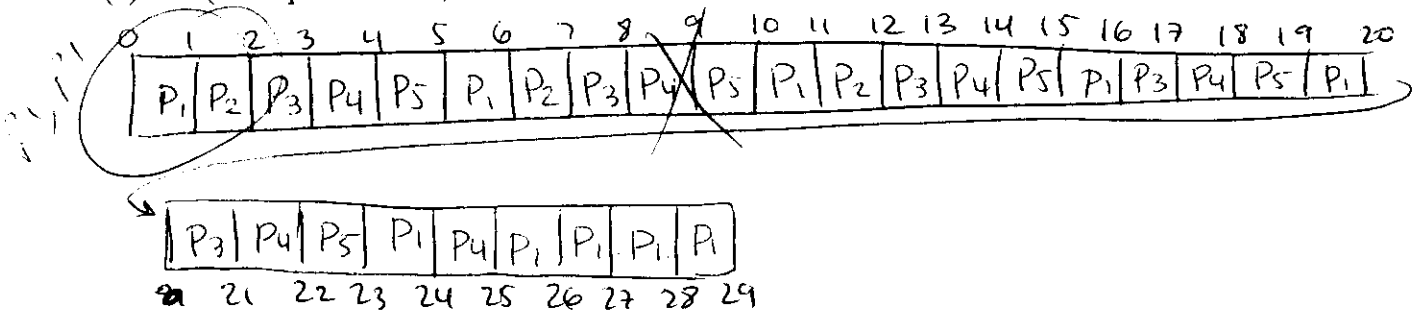
5 context switches.

Gantt chart:



RAN

(e) RR (time quantum is 1)



Av waiting time = (20 + 9 + 16 + 17 + 18)/5

= 16

28 context switches (1)

(f) How long should the context switching code be if it is to take up less than 5% of CPU time when Round Robin is used?

The context switching code should be

(6)

RAN

3.Synchronization

There are 3 problems to solve in this section.

Problem 1

Consider the Bakery algorithm, which is used to coordinate the access of multiple processes to their critical sections.

The shared data structures among processes are:

```
Boolean choosing[i];
int number[i];
```

initially:

```
choosing[i] = false;
number[i] = 0;
```

Remember that a bakery algorithm serves the process with the lowest number received by the algorithm. If two processes have the same number, then the process with the lowest id is served first.

```
do {
     choosing[i] = true;
     number[i] = max(number[0], ..., number[n-1]) + 1;
     choosing[i] = false;
     for (j=0; j<n; j++) {
          while ( choosing[j] )
               ;
          while ( (number[j] != 0) &&
                  (number[j], j) < (number[i],i) )
               ;
     }
          critical section
     number [i] = 0;
          remainder section
} while (true);
```

RAN

a) Prove that processes implementing the Bakery algorithm are mutually exclusive.

Let us take two processes P_0 & P_1 to explain this property.

Process p_0 will choose the max number and so will process p_1.

After the choosing is done, it is set to false again.

In the 2nd while condition, we have the 2nd stmt (number[j], j < number[i], i) which will never be true for both processes. It will be true for one and false for the other logically since if $a < b \not\Rightarrow b < a$.

where it holds false, the process will enter its critical section whereas the other process will be waiting.

When the 1st process exits its critical section & changes the number to 0 again, this will release the 2nd process, allowing it go into its critical section. Since they never go into the critical section together $\Rightarrow$ mutual exclusivity holds.

RAN

b) Initially, the shared data `choosing[i]` is set to `false` for all processes P_i. Explain why each process P_i needs to set `choosing[i]` to `true` and then to `false` again. Note that only one statement executes between the time P_i sets `choosing[i]` to `true` and the time it sets it to `false`. Illustrate your argument with at least one example.

This is done for processes that arrive at a [redacted] time when choosing of another process is set to true and so it will have to wait in the first while stmt [redacted], for that process to finish choosing its max-number before checking for the second while statement which will not have a value if we don't wait on the 1st while. Why ???

(1)

RAN

c) Explain why, using the algorithm, the processes are served on a first-come, first-served basis.

Because the first process to come will get the maximum number ~~and~~ (lowest id) and so, the 2nd part of 2nd while stmt will ~~not hold true~~ begin by holding false for it and so it will directly enter its critical section.

d) Verify that the Bakery algorithm satisfies progress and bounded-waiting.

It satisfies progress since after ~~finish~~ exiting its critical section, it returns the value of number to 0 making the stmt number[j] != 0 false ~~in the corresponding~~ for another process and hence allowing that process to enter its critical section.

Bounded-waiting satisfied since the number[i] is only changed in the exit section allowing a certain process to enter its critical section only once before giving the turn to another process.

RAN

Problem 2

Consider the following code sequences for processes A and B.

Process A:

```
region V1 do
begin
     { Do some stuff.}
end; /*line 4 */

region V2 do
begin
     {Do more stuff.}
end;
```

Process B:

```
region V1 do
begin
     { Do other stuff.}
end;
```

a) Assuming that processes A and B become ready at the same time, give all possible execution timelines for both processes.

RAN

b) Explain what happens if process B were to arrive after process A has executed the statement on the line labeled "`line 4`"?

RAN

Problem 3

Consider the first readers-writers problem. The solution presented in class favors readers and can lead to the starvation of writers. Propose an algorithm in which a writer would not starve, but will still allow a reader to read provided a writer is not already waiting. Give the structure of both the reader and the writer processes.

The algorithm can assume that semaphores have FCFS queues.

Writer:

```
wait (mutex),
wait (~~write~~);
    //write
signal (read);
```

Reader:

```
~~readcount~~
wait (mutex);
read count++;
~~wait (read);~~
if (read count == 1)
    //read
```

```
writer:
writecount ++,
wait (wrt),
    //write
signal (wrt),
```

```
Reader
if (writecount <= 0)
wait (mutex),
readcount ++,
~~if (readcount == 1)~~
if (readcount == 1)
~~signal~~ wait (wrt),
signal (mutex),

    //Read, --
wait (mutex);
readcount --,
if (readcount == 0)
    signal (wrt),
signal (mutex),
```

RAN

4. Deadlocks

Problem 1

Consider the following 2 resource allocation states for a single resource type A. The resource type A has 10 resource instances.

t_0	*Used*	*Max*
P_0	0	6
P_1	0	5
P_2	0	4
P_3	0	7

Available: 10

(a)

t_1	*Used*	*Max*
P_0	1	6
P_1	1	5
P_2	2	4
P_3	4	7

Available: 2

(b)

(a) At time t_0, the system has the resource allocation state given by Table (a) above. Is this resource allocation state safe? Explain.

Yes, this resource allocation state is safe since ~~the need is always the maximum in this case but the maximum is always less than what is available and so each process can use its required resources and then give turn to the next process.~~ no. resources instances of A are used by any processes.

2.5

RAN

(b)At time t_1, the system has the resource allocation state given by Table (b) above. Is this resource allocation state safe? Explain.

This resource allocation state is ~~not~~ safe since ~~only~~ ~~process P_2 can run with the available resources.~~ ~~All the other processes have need > available~~ ~~> deadlock situation. unsafe state~~ processes are using resources instances of A are there are still some available.

0

(c)At time t_2, what would happen if P_1 requests one additional resource. Should the request be granted? What would the new resource allocation state be?

Yes, the request should be granted since we still have 2 available instances of resource A.

t_2	Used	max
P_0	1	6
P_1	2	5
P_2	2	4
P_3	4	7

Available : 1

0

RAN

(d)Write the algorithm you used to derive all the above answers.

~~Finish(i) = false for all i~~

~~Used~~ work = Available

IF Request$_i$ < ~~Available~~ Max$_i$ (else produce an error)

IF Request$_i$ < work (or what is available)

else ~~also produce~~ this becomes unsafe

Then pretend to do the following:

Available = Available - Request$_i$

Used = Used + Request$_i$