Bilkent University
Computer Engineering Department

CS 342
Operating Systems

Midterm Exam
Date: April 25, 2003, Friday
Duration: 120 minutes

Name of the Student	
ID of the Student	

GRADE $\boxed{}$

- *Show your work and reasoning clearly!*
- *Write legibly!*
- *Write only to the space provided for each question!*
- *There should be total of 12 questions. Check your exam paper.*

Good Luck!

1. Problem (30 points)

Mark each of the following statements as either *true (T)* or *false (F)* . You will receive +1 point for each correct answer, and -1 point for each incorrect answer. Therefore, be careful while you are marking. There is a possibility that one may end-up with a negative total grade for this question.

	Statement	***True***	***False***
1.	Application programming would be easier without having and using operating systems.		x
2.	Clean separation between designers, builders, operators, and programmers for operating systems has been done in 3rd generation systems.		x
3.	A batch system should have a preemptive scheduler.		x
4.	Multiprogramming increases CPU utilization.	x	
5.	Multiprogramming and time-sharing are different concepts.	X	
6.	A multiprogramming system needs to be an interactive system, but a time-sharing system needs only be able to share the memory and CPU between several batch jobs.		x
7.	A trap is an other name for hardware interrupt.		x
8.	Memory mapped I/O does not require special I/O instructions and separate I/O port space does not consume from main memory address space.	X	
9.	Polling and DMA I/O methods cause waste of CPU time in busy waiting.		x
10.	PIC can enable/disable interrupts	x	
11.	A device controller can use a single IRQ number, but still can generate different types of interrupts.	x	
12.	The bus between CPU and PCI bridge is called local bus.	x	
13.	Level 2 cache is a cache that is on the same chip with CPU.		x
14.	ISA bus is faster than PCI bus.		x

15.	SCSI disks have faster transfer rates than IDE disks.	X	
16.	USB is a standard that uses dedicated and separate data lines between each USB device and the USB controller.		x
17.	Every system call requires changes from user mode to kernel mode.	x	
18.	All library routines in POSIX standard interface use one or more system calls to implement the functionality of the library routine.		x
19.	In monolytic operating systems, the kernel is just a single executable file.	x	
20.	A process has four states: running, ready, blocked, sleeping.		x
21.	A process may switch from ready to blocked state if it starts an I/O operation.		x
22.	All threads have the same address space but different set of open files.		x
23.	Every thread that a process creates will have a different PC register value but the same SP register value.		x
24.	Each thread will have separate address space.		x
25.	For user-level thread implementation, run-time system is a sub-module of kernel that is responsible from managing the threads.		x
26.	If threads are implemented inside kernel, thread creation will be more costly compared to user-level thread implementation.	x	
27.	Peterson’s solution for mutual exclusion is a solution that requires OS support.		x
28.	TSL instruction for providing mutual exclusion is a hardware solution and does not cause busy waiting.		x
29.	A mutex variable is just a binary semaphore that can be in one of two states.	x	
30.	SJF scheduling algorithm provides always optimal turnaround time only if all jobs arrive at the same time to a batch system	x	

2. Problem (9 points)

Process	Arrival Time	Required CPU Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

Assume five processes (A through E) arrive to a system at arrival times shown in the figure. The figure also gives the CPU time that each process requires. Assume all processes are %100 CPU-bound and ignore the context-switch overhead. Compute finish time and turnaround time of each process for the following scheduling algorithms:

a) (*1 points*) FCFS
b) (*2 points*) Round Robin with quantum=1
c) (*2 points*) Round Robin with quantum=4
d) (*2 points*) Shortest Job First (SJF)
e) (*2 points*) Shortest Remaining Job First

Use the figures below to provide your answers. Mark on them. We provide some part of the answer for FCFS case. You should fill the rest taking this as an example.

First Come First Served (FCFS)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19 20
A	x	x	x																	
B				x	x	x	x	x	x											
C										x	x	x	x							
D														x	x	x	x	x		
E																			x	x

Finish Time	
A	??? 3
B	??? 9
C	??? 13
D	??? 18
E	??? 20

Turnaround Time	
A	??? 3
B	??? 7
C	??? 9
D	??? 12
E	??? 12

7

Round Robin (q=1)

	0	1	2	3	4	5	6	7	8	8	10	11	12	13	14	15	16	17	18	19	20
A	x	x		x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
B			x		x		x			x				x				x	▨	▨	
C						x			x				x				x	▨	▨	▨	
D								x				x				x			x	x	
E											x				x	▨	▨	▨	▨	▨	

Finish Time	
A	4
B	18
C	17
D	20
E	15

Turnaround Time	
A	4
B	16
C	13
D	14
E	7

Round Robin (q=4)

	0	1	2	3	4	5	6	7	8	8	10	11	12	13	14	15	16	17	18	19	20
A	x	x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
B				x	x	x	x									x	x	▨	▨	▨	
C								x	x	x	▨ x	▨	▨	▨	▨	▨	▨	▨	▨	▨	
D												x	x	x	x					x	
E																		x	x	▨	

Finish Time	
A	3
B	17
C	11
D	20
E	19

Turnaround Time	
A	3
B	15
C	7
D	14
E	11

Shortest Job First (SJF)

0	1	2	3	4	5	6	7	8	8	10	11	12	13	14	15	16	17	18	19	20
A	x	x	x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
B					x	x	x	x	x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
C													x	x	x	▨	▨	▨	▨	▨
D																x	x	x	x	x
E											x	x	▨	▨	▨	▨	▨	▨	▨	▨

Finish Time	
A	3
B	9
C	15
D	20
E	11

Turnaround Time	
A	3
B	13
C	4
D	14
E	2

Shortest Remaining Job First

0	1	2	3	4	5	6	7	8	8	10	11	12	13	14	15	16	17	18	19	20
A	x	x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
B				x							x	x	x	x	x	▨	▨	▨	▨	▨
C					x	x	x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
D																x	x	x	x	x
E									x	x	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨

Finish Time	
A	3
B	15
C	8
D	20
E	10

Turnaround Time	
A	3
B	13
C	4
D	14
E	2

3. Problem (4 points)

What are the four conditions that are necessary for having deadlocks in a system?

a) (*1 point*) Mutual Exclusion

b) (*1 point*) Hold and Wait

c) (*1 point*) No preemption

d) (*1 point*) Circular Wait

4. Problem (4 points)

Assume there are 3 processes and 4 resource types in a system. The following system state is given. Is there a deadlock in the system or not? Why or why not? Explain!

$$\text{Available} = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

$$\text{Request} = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \qquad \text{Allocation} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$

No, there is no deadlock. The following execution sequence of processes can be achieved: process 3,process 2, process 1

5. Problem (6 points)

Assume we have 5 processes and 4 resources types in a system and the following system state.

Current Allocation				
	R1	R2	R3	R4
P1	0	0	1	2
P2	2	0	0	0
P3	0	0	3	4
P4	2	3	5	4
P5	0	3	3	2

Maximum Demand				
	R1	R2	R3	R4
P1	0	0	1	2
P2	2	7	5	4
P3	6	6	5	6
P4	4	3	5	6
P5	0	6	5	2

Available			
R1	R2	R3	R4
2	1	0	0

a) (*2 points*) Compute *Resources Still Needed* matrix.

Resources Still Needed				
	R1	R2	R3	R4
P1	0	0	0	0
P2	0	7	5	4
P3	6	6	2	2
P4	2	1	0	2
P5	0	3	2	0

b) (*4 points*) Is the system currently in safe or unsafe state? If safe, show the execution sequence of processes so that all of them can finish without deadlock! If not safe, show the state of the *Resources Still Needed* matrix and *Available vector* after which no progress in system state is possible.

The current state is safe. The following execution order of processes will enable every process to complete without any problem: P1, P4, P5, P2, P3.

6. Problem (9 points)

Assume we have a memory that have the following holes in increasing memory start address order: Hole1 = 20 KB, Hole2 = 8 KB, Hole3 = 40 KB, Hole4 = 36 KB, Hole5 = 14 KB, Hole6 = 18 KB, Hole7 = 24 KB, Hole8 = 30 KB. Which hole is taken for *successive* segment requests of 24 KB, 20 KB and 18 KB, if we use:

a) (*3 points*) First Fit

1) 24 KB:
   Hole3
2) 20 KB:
   Hole1
3) 18 KB:
   Hole4

b) (*3 points*) Best Fit

1) 24 KB:
   Hole7
2) 20 KB:
   Hole1
3) 18KB:
   Hole6

c) (*3 points*) Worse Fit

1) 24 KB:
   Hole3
2) 20 KB:
   Hole4
3) 18 KB:
   Hole8

7. ***Problem (3 points)***

Virtual Page Number	Valid Bit	Referenced Bit	Modified Bit	Page Frame Number
0	1	1	0	4
1	1	1	1	7
2	0	0	0	-
3	1	0	0	2
4	0	0	0	-
5	1	0	1	0

Assume the portion of the page table for a process that is currently executing in CPU is given in the figure above. All numbers are decimal and everything is numbered starting from zero, and all addresses are memory byte addresses. The page size is 2 KB.

What physical address would each of the following virtual address correspond to? Which virtual addresses would cause page faults?

a) 2104

2104 = 2048 + 56
2104 is in page# 1
page# 1 is stored in frame# 7.

7x2048 + 56 = <u>14392</u>

b) 4443

4443 = 4096 + 347
4443 is in page# 2
page# 2 does not have a corresponding frame.

This causes a page fault.

c) 10199

10199 = 8192 + 2007
2007 is in page# 4
page# 4 does not have a corresponding frame.

This causes a page fault.

8. Problem (6 points)

A hypothetical personal computer that uses virtual memory has 1 MB of RAM and generates 24 bit virtual addresses. Assume that the computer uses pages of size 1 KB.

a) (*1 points*) What is the required *page table size* (in bytes) if the computer uses a one-level page table and stores the whole table in memory? (Assume each page table entry requires 8 bytes of memory).

$1KB = 1024 = 2^{10}$

10 bits are used for offset field in virtual address

24 - 10 = 14 bits are used for page table index.

page table size $= 2^{14} x 2^{3} = 2^{17} = 131072$ bytes

b) (*2 points*) What is the maximum total memory space (in bytes) that is required to a store two-level page table. Assume that, bits 18 through 23 of a virtual address are used as index to the top level page table and bits 10 through 17 are used as index to a second level page table. Assume each page table entry requires 8 bytes of memory independent of whether it belongs to a top-level or second-level page table and independent of whether the entry is valid or not.

Entries in top level page table $= 2^{6}$

Entries in a secondary page table $= 2^{8}$

Maximum number of secondary page tables $= 2^{6}$

Maximum space required for the page table =

$2^{6} x 8 + 2^{6} x 2^{8} x 8 = 2^{9} + 2^{17} = 131584$

c) (*3 points*) Assume a program is to be run in this system which uses two-level page table with parameters given in b). The program's text segment is 512 KB, data segment is 2 MB, and stack segment is 256 KB. Hence, the total size of the program is 2816 KB. How much memory space (in bytes) is required to store the two-level page table for this program while it is in execution in CPU.

Each secondary page table can address :

$2^{8} x 2^{10} = 2^{18} = 256MB.$

For text segment we need : 2 secondary page tables (512/256)

For data segment we need : 4 secondary page tables

For stack segment we need : 1 secondary page tables

Total number of secondary page table required = 7

Size of top level page table $= 2^{6} x 2^{3} = 2^{9}$

Size of each secondary page table $= 2^{8} x 2^{3} = 2^{11}$

Total space required to store all pages $= 2^{9} + 7 * 2^{11} = 14848 bytes$

9. Problem (9 points)

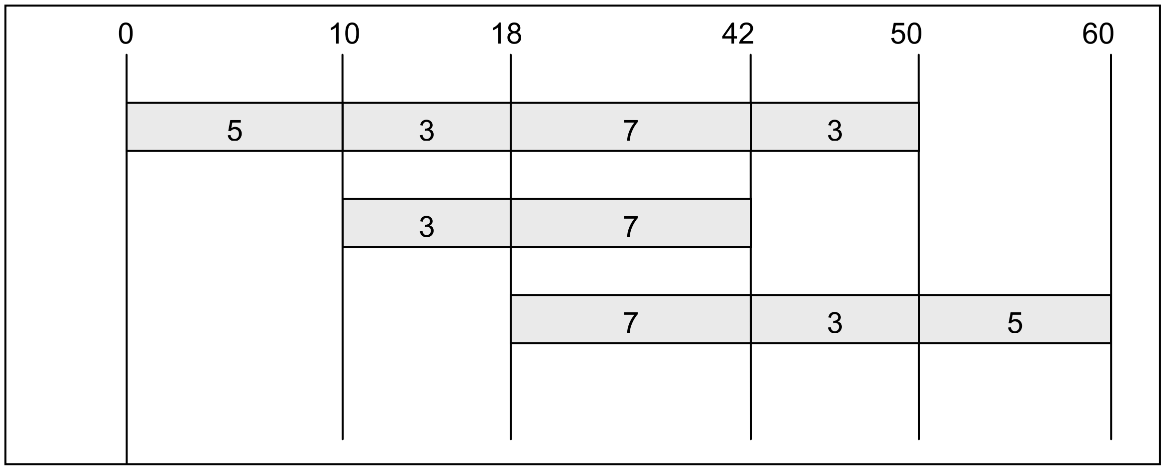
Consider a *multiprogramming* computer system (which models pure processor sharing) whose jobs average 50 percent I/O wait time. 3 jobs are submitted to the system at the arrival time given in the table below. The job A, arrives at time 0 and requires 18 minutes of CPU time. The job B arrives at time 10 and requires 10 minutes of CPU time. Job C arrives at time 18 and requires 15 minutes of CPU time. When does each job finishes? Give your final answer on the table provided below. Show your work about how you have derived your results.

	Arrival Time	CPU Time Required
A	0	18
B	10	10
C	18	15

Answer	Finish Time
A (*3 points*)	50
B (*3 points*)	42
C (*3 points*)	60

Answer:

	1 process	2 processes	3 processes
CPU idle	½	¼	1/8
CPU busy	½	¾	7/8
CPU/process	½	3/8	7/24



10. Problem (6 points)

A file needs to be shared by different processes. Each process has a unique number. The file can be simultaneously accessed 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*.

a) (*3 points*) Assume our OS supports general semaphores together with two operations up(*s*) and down(*s*), where *s* is a general semaphore. Use general semaphore(s) to coordinate access to the file.

```
/* N is global */
/* s is a global semaphore */
/* s is initialized to N */
enter(int i)
{
        int k;
        for (k=0; k<i; ++k)
                down(s);
}

leave(int i)
{
        int k;
        for (k=0; k<i; ++k)
                up(s);
}
```

```
main()
{
        /* this is program with
        number i */

        enter(i);

        /* access the file here

        leave(i);
}
```

b) (*3 points*) Assume the programming language that you are using supports monitor construct. Write a monitor to coordinate access to the file.

```
monitor AccessFile
{
        condition countSmallThanN;
        int count;
        procedure enter(int i)
        {
                while (count + i > N)
                        wait(countSmallThanN);
                count = count + i;
        }
        procedure leave(int i)
        {
                count = count – i;
                signal(countSmallThanN);
        }
}
/* the following code is executed by every process with number i */
void
main()
{
        enter(i);
        /* code to access the file comes here.  */
        leave(i);
}
```

11. Problem

An attempt to solve the critical section problem between two processes is given below. Two variables *flag* and *turn* are shared by two processes*. i* is the number of one process, which can be 0 or 1, and *j* is the number of other other, which can be 1 or 0.

```
boolean flag[2]; /* initially false */
int      turn;

do
{
        flag[i] = true;
        while (flag[j])
        {
                if (turn == j)
                {
                        flag[i] = false;
                        while (turn == j);
                        flag[i] = true;
                }
        }

        /* critical region */

        turn = j;
        flag[i]= false;

        /* remainder non-critical section */

} while(1);
```

Which of the following three requirements does this solution satisfy? Mark the correct box. If you don't want to provide an answer, do not mark! *A wrong answer will cause* ***2 points*** *to be* ***subtracted****.*

a) *(2 points) Mutual Exclusion*: No two processes can be inside their critical regions simultaneously.

Satisfies	x
Does not satisfy	

b) *(2 points) Progress*: No process running out of its critical region can block other process.

Satisfies	x
Does not satisfy	

c) *(2 points) Bounded Waiting*: No process should have to wait forever to enter its critical region.

Satisfies	x
Does not satisfy	

12. Problem

Assume an operating system has only binary semaphores, and you will enhance this operating system it support also general semaphores. The binary semaphores in this OS can be in one of two states and there are two operations on them: *down_binary*() and *up_binary*(). The semantics of these operations is the same with what we have learned in the class for mutex variables. Namely, a *down_binary* operation on a binary semaphore with value zero will suspend the process, and an *up_binary* operation on a binary semaphore with value zero will increment the semaphore value if there is no process sleeping on that semaphore, or it will wake-up one of the sleeping processes, if there are one or more processes sleeping on that semaphore. In the second case, the semaphore value is unchanged.

Now, you should implement *general semaphores* using binary semaphores available in this operating system and you should provide two operations on these general semaphores, down(*s*) and up(*s*), whose semantics is the same with what we have seen in the class for general semaphores. Here, *s* is a general semaphore.

Hint: Use two binary semaphores: *mutex* and *delay*. Binary semaphore *mutex*, which is initialized to 1, will be used to provide mutual exclusion for updating general semaphore's value. Binary semaphore *delay*, which is initialized to 0, will be used to suspend a process when necessary.

The following solution can be found in the following reference:

(*) "A correct implementation of general semaphores", by Hemmendinger, Operating Systems Review, July 1988.

```
binary_semaphore mutex = 1;
binary_semaphore delay  = 0;
void down(semaphore s)
{
        down(mutex);
        s = s – 1;
        if (s < 0)
        {
                up(mutex);
                down(delay);
        }
        up(mutex);
}
```

```
void down(semaphore s)
{
        down(mutex);
        s = s + 1;
        if (s <=0 )
                up(delay);
        else
                up(mutex);
}
```