Bilkent University
Department of Computer Engineering

CS 342
Operating Systems
Spring 2003
Final Exam

Date: May 28, 2003, Wednesday
Time: 15:30
Duration: 120 minutes

Name	
Student ID	

Grade	

Please Read Carefully

- Show your work and reasoning clearly!
- Write legibly!
- There should be total of 6 questions. Check your exam paper!
- You can use a calculator, but not a PDA or handheld computer.
- Answers that do not make sense will not get any credit!
- Try to write to the space provided under each question! You can use extra sheet(s) if that space is not enough. Do not foget to staple your extra sheet(s).

Note that
1 KB = 2^{10} bytes
1 MB = 2^{20} bytes
1 GB = 2^{30} bytes

1. Problem (5 points)

How much cylinder skew (in terms of <u>number of sectors</u>) is needed for a 7200 rpm disk with a track to track seek time of 2 msec? Assume the disk has 300 sectors in each track and each sector is 512 bytes.

X = The time that it takes until a whole sector passes under disk head = (1/7200)*60/300sec.

Cylinder skew = 0.002 sec / (x sec/sector) = **<u>72 sectors.</u>**

2. Problem (16)

A machine that is using virtual memory has 32 bit virtual addresses and 256 MB physical memory. The page size is 4 KB. There are two programs, A and B, which wil be run at different times. Program A has a size of 512MB and Program B has a size of 16 MB. Assume each page table entry requires 8 bytes of storage in memory.

a) Compute the required page table size that each program (A and B) requires, if the system is using inverted page tables. Assume each program requires a different inverted page table and assume the whole physical memory is dedicated to the program (although this may not be realistic).

Memory size = 256 MB.
Number of frames in memory = 256 MB / 4 KB = 65536 = 2^16 page frames.

1) Program A

Inverted page table size = number of entries in the table * entry size
= 2^16 * 2^3 = 2^19 = 512 KB. = 524288 bytes. = 0.5 MB

2) Program B

The same with 1). 0.5 MB

b) Compute the required page table size for each program if the system is using *two-level page tables*. Assume top-level page table index is 11 bits, and second-level page table index is 9 bits.

1) Program A

Each second level page table can address 2^9 * 2^12 = 2^21 = 2 MB address space.

For program A, wee need 512 MB / 2 MB = 256 second level page table.

Total page table size =
Size of top level page table + 256 * (size of a second level page table)

= (2^11 * 2^3) + (2^8 * 2^9 * 2^3) = 2^14 + 2^20 = **1064960 bytes.**

2) Program B

Program B requires 16 MB / 2 MB = 8 second level page tables.

Total page table size =
Size of top level page table + 8* (size of a second level page table)

= (2^11 * 2^3) + (2^3 * 2^9 * 2^3) = 2^14 + 2^15 = **49152 bytes**

3. Problem (20)

Assume requests for disk blocks that are located in the ***following cylinders*** of a disk are queued: 55,58,39,18,90,160,150,38,184. Assume the requests are queued in the order of their arrivals. Compare the peformance of the following disk scheduling algoritms using the tables (figures) provided in the next page. For each algorithm, at each step, compute the next cylinder visited, the number of cylinders that are traversed to reach to this next cylinder, and the total number of cylinders that are traversed (which is the total cost of the algoritm).

Make sure you write your final answers on the tables provided on the next page. Those tables will be graded!

a) FIFO
b) Shortest Seek First (SSF)
b) Elevator (direction up) (starting in the direction of *increasing* cylinder number)
c) Elevator (direction down) (starting in the direction of *decreasing* cylinder number)

Assume the head is intially at cylinder 100.

	a) FIFO Starting at cylinder 100	
	Next cylinder accessed	Number of cylinders traversed
1	55	45
2	58	3
3	39	19
4	18	21
5	90	72
6	160	70
7	150	10
8	38	112
9	184	146
	Total cylinders traversed	498

	b) SSF Starting at cylinder 100	
	Next cylinder accessed	Number of cylinders traversed
1	90	10
2	58	32
3	55	3
4	39	16
5	38	1
6	18	20
7	150	132
8	160	10
9	184	24
	Total cylinders traversed	248

	c) Elevator (Up) Starting at cylinder 100	
	Next cylinder accessed	Number of cylinders traversed
1	150	50
2	160	10
3	184	24
4	90	94
5	58	32
6	55	3
7	39	16
8	38	1
9	18	20
	Total cylinders traversed	250

	d) Elevator (Down) Starting at cylinder 100	
	Next cylinder accessed	Number of cylinders traversed
1	90	10
2	58	32
3	55	3
4	39	16
5	38	1
6	18	20
7	150	132
8	160	10
9	184	24
	Total cylinders traversed	248

4. Problem (10 points)

Consider the two-dimensional array A:

```
int A[][] = new int[100][100];
```

where A[0][0] is stored at location 200, in a paged memory system with pages of size 200 bytes. A small process (whose size is 100 bytes) resides in page 0 (locations 0 through 99) for manipulating the matrix A; thus every instruction fetch will be from page 0.

Assume 3 page frames are allocated for the program (page frame 1, page frame 2, page frame 3). How many page faults will be generated by the following array-initialization loops, using LRU replacement, and assuming page frame 1 has the process in it, and the other two are initially empty.

Information: *Assume a two dimensional array that is declared in a program is stored in virtual memory so that row 1 comes first, then row 2, then row 3, etc. In this case, A[0][0] is stored at virtual address 200, A[0][1] is stores at virtual address 204, A[1][0] is stored at virtual address 600, , A[2][0] is stored at virtual address 1000, etc.*

a)

```
for (int j = 0; j < 100;  j++)
      for (int i = 0; i < 100; i++)
            A[i][j] = 0;
```

The reference string will look like the follwing (consists of the following page numbers).
:

First part:

1,3,5,….199, 1,3,5,…199,…… 1,3,5,….199

(1,3,5,…199 string is repeated 50 times above. Each string contains 100 page numbers).

Then, second part:

2,4,6,…,200, 2,4,6,…,200,..... ,2,4,6,….,200.

(2,4,6,…200 string is repeated 50 times above. Each string contains 50 page numbers).

Every new cell reference will cause a page fault in this case.
So first part will have:
100 x 50 = 5000 page faults.
Second part will have the same number of page faults:
Therefore, the total number of page faults will be : **10000** (ten thousand). .

b)

```
for (int i = 0; i < 100;  i++)
     for (int j = 0; j < 100; j++)
          A[i][j] = 0;
```

Each page contains: 200 bytes / 4 = 50 integers.
Total number of pages required: 100x100 / 50 = 200 pages.

A[0][0] is stored in page 1
A[0][1] is stored in page 1
A[0][2] is stored in page 1
…
A[0][49] is stored in page 1
A[0][50] is stored in page 2
A[0][51] is stored in page 2
….
A[0][99] is stored in page 2
A[1][0] is stored in page 3
…
A[0][150] is stored in page 4
…
A[0][1] is stored in page 4

Page accesses will be like the followin:

1,1,1,1…,1,2,2,2,…,2,3,3,3,…,3,…….199,199,199,…199,200,….2000
Each page is referenced 50 times. Since each page keeps 50 integers.
Since pages are accessed successively, there will be total of **200** page faults.
Every new page access will cause a page fault.

5. *Problem (24 points)*

A process contains 8 virtual pages and is assigned a fixed allocation of 4 page frames in memory. The page frames that are assigned to that process are: page frame 1, page frame 2, and page frame3, and page frame 4. The following virtual page trace (reference string) occurs: 1,0, 2,2,1,7,6,7,0,1,2,0,3,0,4,5,1,5,2,4,5,6,7,6,7,2.

For each of the following page replacement algorithms, **mark** which references causes a page fault <u>on the row named "Answer (Page Faults)"</u> and compute the total number of page faults and write it on the box named "Answer (number of page faults)".

(You can use the cells between "Refence String" row and "Answer (Page Faults)" row in the tables below for your computation. But this is up to you. You don't have to use those cells if you don't want to use).

Important Assumption*: When there are more than one page that are candidates to become evicted (for removal), select the lower numbered page. For example if page 0 and page 3 are candidates for removal, select page 0!.*

a) FIFO algorithm

FIFO Algorithm

Reference String		1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2
keeps the oldest page.		1	1	1	1	1	1	0	0	0	2	2	7	6	6	1	0	3	3	4	4	4	5	1	1	1	1
			0	0	0	0	0	2	2	2	7	7	6	1	1	0	3	4	4	5	5	5	1	2	2	2	2
				2	2	2	2	7	7	7	6	6	1	0	0	3	4	5	5	1	1	1	2	6	6	6	6
keeps the newest page							7	6	6	6	1	1	0	3	3	4	5	1	1	2	2	2	6	7	7	7	7
Answer (Page Faults)		x	x	x			x	x			x		x	x		x	x	x		x			x	x			

Answer (number of page faults)	14

b) LRU Algorithm

LRU Algorithm

Reference String		1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2
		1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2
			1	0	0	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7
				1	1	0	2	1	1	6	7	0	1	2	2	3	0	4	4	1	5	2	4	5	5	5	6
							0	2	2	1	6	7	7	1	1	2	3	0	0	4	1	1	2	4	4	4	5
Answer (PageFaults)		x	x	x			x	x		x		x		x		x	x	x		x			x	x			x

Answer (number of page faults)	15

c) NFU Algorithm

NFU Algorithm

Reference String		1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2
		1	1	2	2	2	2	2	7	7	1	2	2	2	0	0	0	1	1	2	2	2	2	2	2	7	2
			0	1	1	1	1	1	2	2	7	1	1	1	2	2	2	0	0	1	1	1	1	1	1	2	7
				0	0	0	7	7	1	1	2	7	0	0	1	1	1	2	5	0	0	0	0	0	0	1	1
							0	6	6	0	0	0	7	3	3	4	5	5	2	5	4	5	6	7	6	0	0
Answer (Page Faults)		x	x	x			x	x		x				x		x	x				x	x	x	x	x	x	

Answer (number of page faults)	15

Frequencies

0	x	x	x	x																		
1	x	x	x	x																		
2	x	x	x	x	x																	
3	x																					
4	x	x																				
5	x	x	x																			
6	x	x	x																			
7	x	x	x	x																		

The above answer is based on the assumption that counters are never reset. I will also accept the answers that assume that counters will be reset when a page is evicted from memory. In this case, the answer will be:

NFU Algorithm

Reference String		1	0	2	2	1	7	6	7	0	1	2	0	3	0	4	5	1	5	2	4	5	6	7	6	7	2
Answer (Page Faults)		x	x	x			x	x		x				x	x	x	x				x	x	x				

Answer (number of page faults)	13

6. Problem (25 points)

The following piece of code is given. Assume there is no buffering in I/O library and OS so that everything that is printed by printf() goes immediately to the screen without any delay. In the code, **sleep(1)** just causes the process to delay execution for 1 seconds. **fflush(stdout)** causes the output of printf() to be immediately flushed (written) to the screen wihout getting buffered.

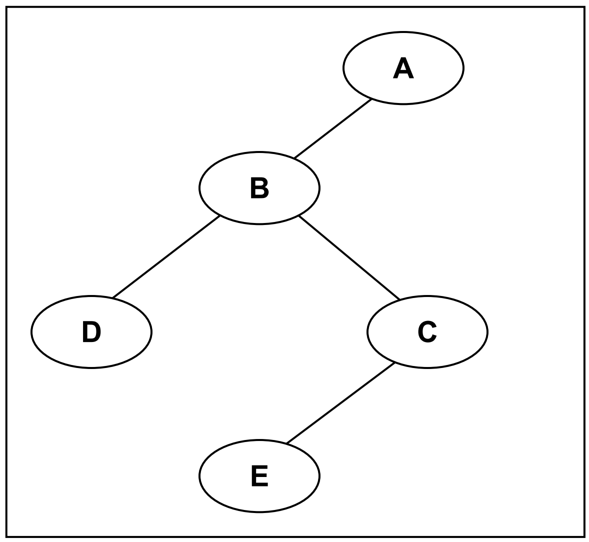
```
#include <stdio.h>

void
main()
{
      int x, y;

      x = 20;
      y = 30;

      if (fork()==0)
      {
            y = 35;
            printf("%d\n",   x); fflush(stdout);
            sleep(1);
            if (fork() == 0) {
                  x = 25;
                  printf("%d\n", x); fflush(stdout);
                  printf("%d\n", y); fflush(stdout);
                  sleep(1);
            }

            if (fork()== 0)
            {
                  printf("%d\n", x); fflush(stdout);
                  exit(0);
                  printf("%d\n", y);  fflush(stdout);
            }
            else {
                  printf("%d\n", x); fflush(stdout);
                  printf("%d\n", y);  fflush(stdout);
                  sleep(1);
            }
      }
      printf("%d\n", x);  fflush(stdout);
      printf("%d\n", y);  fflush(stdout);
      exit(0);
}
```

Answer the following questions!. Please provide clean and clear answers!. Otherwise your answers will NOT get ANY credit.

a) How many total number of processes will be created (including the first parent process that is created by typing the name of the program).

5 processes

b) Draw the process tree (a tree that shows parent-child relationships). Give a letter to each process, such as *A, B, C, D, E, F, G*, …. *A* should be given to the initial parent process. You will use this identification scheme in some later questions.



Process Tree is shown above. The order of creation of processes are: A, B, C, D, E.

c) The program prints numbers from the set {20, 25, 30, 35} to the screen. It prints one number at each line. How many numbers will be printed totally? (For example, if *20 20 25* would be printed to the screen – each number on a different line - , then the answer would be 3).

15

d) How many 20's will be printed totally?

5

e) How many 25's will be printed totally?

4

f) How many 30's will be printed totally?

1

g) How many 35's will be printed totally?

5

h) For each process that you have shown in the process tree (in question b)), which numbers are printed by that process. If a number is printed many times, show that number many times (as much as it is printed).

A - 20 30
B - 20 20 35 20 35
C - 25 35 25 35 25 35
D – 20
E – 25

i) By referencing your answer in b) and using the identification of processes in that question, which process would terminate first.

A, since it does not executes a sleep(1) call. The scheduling granularity is in the order of 10s of miliseconds. During 1 second period, scheduler may intervene and schedule new processes lots of times.

j) By referencing your answer in b) and using the identification of processes in that question, which process would terminate last.

C, since it will call *sleep(1)* twice.