Professor: Ehab ELMALLAH

Question	Mark
1	
2	
3	
4	
5	
Total	

Surname: ....

Given name: ....

Id number: ....

CMPUT 379 - Final Examination (38%) A1

Time: 120 minutes
Total Marks: 68
Questions: 5

December 19, 2000
Closed Book
Calculators Allowed

- ***No questions during exam time.***
- ***If you are unsure, write down your assumptions.***
- ***Answer each question in the space given on this form.***
- ***No additional sheets are allowed.***

Question 1 [20 marks]

1. True or false: the password encryption scheme in UNIX is limited to map each password text into a unique ciphertext? Explain.

2. A *cache* is a relatively small, relatively fast memory that is used to hold a partial copy of the contents of a larger, slower memory. Give two examples of caching that are directly related to virtual memory management. For each example, indicate what is the small fast memory, and what is the large slow memory.

3. Consider a UNIX i-node with 10 direct pointers, one single-indirect pointer, and one double-indirect pointer. Assume that the block size is 8 Kbytes, and that the size of a pointer is 4 bytes. How large a file can be indexed using such an i-node? Explain.

4. A small virtual memory system has only two frames of physical memory, and uses demand paging. Both frames are initially empty. Consider the following page reference string:

1 2 3 2 3 1 3 1 2 1

Give a lower bound on the number of page faults that will result from this reference string.

Question 2 [10 marks]

Recall that the semantics of the P and V primitives for a *counting* semaphore C are as follows:

P(C): decrement by 1; if the new value is negative then add the process to an associated waiting queue and block the process;

V(C): increment by 1; if the associated queue is nonempty then wakeup one of the waiting processes;

The following is intended to be an implementation of the above primitives using two ***binary*** semaphores S1 and S2 (each semaphore has an associated waiting queue), and an integer C, and nothing else. Fill in each of the ***seven*** empty boxes below with an appropriate statement.

1. Initialize: S1= 1, and S2= [ ]

2. Implement the P(C) operation as:

```
[ ] ;
C= C - 1;
if ( C < 0 ) { [ ] ; [ ] ; }
else V(S1);
```

3. Implement the V(C) operation as:

```
[ ] ;
C= C + 1;
if ( [ ] ) [ ] ;
V(S1);
```

Question 3 [15 marks]

1. Suppose we have 4 processes requesting units of a particular non-shareable resource with N units. The following table gives the current allocation situation.

	Maximum Demand (in units)	*Current Allocation* (in units)
$P1$	5	4
$P2$	6	1
$P3$	6	3
$P4$	4	2

(a) Find the smallest N for which the above state is *safe* with respect to the banker's algorithm? Explain.

(b) Suppose that $N = 12$, and that process $P1$ were to complete and 3 of its units were allocated to $P2$. If $P3$ and $P4$ were then to issue "simultaneous" requests for an additional unit, which request(s) does the banker's algorithm grant?

(a) $P3$ only (b) $P4$ only (c) both
(d) either one, but not both (e) neither one

2. Imagine that you have a pool of identical resource units of a non-preemptable resource type. Each unit is sharable by up to 5 processes. Any request for multiple units, however, should be allocated distinct units. Which one of the following statements is true?

   (a) such a resource can never be involved in a deadlock scenario because the mutual exclusion condition is denied

   (b) if there is a deadlock scenario involving this resource, the number of deadlocked processes is at least 25

   (c) the banker's algorithm can be used to prevent deadlock with this resource, if each unit is represented as five separate units of that particular resource type

   (d) the banker's algorithm requires modifications to prevent deadlock with this resource

   (e) there is no way to prevent deadlock with this resource because you never know if a unit is available or not

   Explain your answer.

Question 4 `[10 marks]`
In a paged virtual memory system, two processes are active. The system has 10 physical page frames numbered from 0 to 9. Each physical frame has 1024 bytes. The page tables of the two processes are shown below, where each row specifies: the physical frame number, the *referenced* (R), the *modified* (M), and the *valid* (V) bits. An entry with $V = 1$ is a valid entry.

Process 1 Page Table

	Frame	R	M	V
0	6	0	1	1
1	4	1	1	1
2	5	1	0	1
3	0	0	0	0
4	1	1	0	1
5	9	1	0	1

Process 2 Page Table

	Frame	R	M	V
0	0	0	0	1
1	7	1	0	1
2	8	0	0	1
3	2	0	1	1
4	0	0	0	0
5	3	1	1	1

The system uses the global Clock Algorithm (the 2*nd* Chance Algorithm). Replacement decisions are made periodically, rather than on demand. At the end of each period, the Clock Algorithm scans **half** of the frames. Any frames selected for replacement during the scan are placed on a free list. When page faults occur, the system chooses pages from the free list for replacement. When modified frames are added to the free list, their pages are not copied back to secondary storage until they are actually replaced by new pages.

1. Given that the above page tables exist at the end of a period, and that the clock hand points at frame zero. (This means that the algorithm will consider frames zero, one, two, three, and four.) Show the page tables of the two processes after the algorithm runs. Indicate which frames will be added to the free list.

2. Immediately after the replacement algorithm runs, process 2 runs and attempts to read logical address 4100, resulting in a page fault. Show the contents of process 2's page table after the page fault handler has run and the virtual address has been translated successfully by the MMU. Assume that the free list contains those frames that were placed there when the replacement algorithm was run (in the above part of the question). Indicate which pages if any, will be copied from memory to disk, and which pages, if any will be copied from disk to memory.

Question 5 [13 marks]

Recall the following rules in UNIX:

- For directories, the *execute* permission lets us pass through the directory when it is a component of a pathname that we are trying to access, whereas the *read* permission lets us obtain a list of all the filenames in the directory.
- To open any file for reading, a process must have *execute* permission in each directory mentioned in the pathname, and appropriate permissions for the file itself.
- We cannot create a new file in a directory unless we have *write* and *execute* permissions in the directory.

The listing shown below is for an unprotected UNIX directory. The directory contains five subdirectories: `dirA` through `dirE`. Each subdirectory contains files with `"-rw-rw-rw-"` permissions whose names are known to all users. In addition, the listing shows three standard UNIX programs: `cat`, `edit`, and `ls`. Each program has either the *set-user-identifier* bit, or the *set-group-identifier* bit turned on, as indicated below. All user groups are disjoint.

```
drwx------   2 admin    admin        1024 Dec 17 01:27 dirA
drw-------   2 admin    admin        1024 Dec 17 01:27 dirB

drwxrwx---   2 lu       labadmin     1024 Dec 17 02:04 dirC
drwxrw----   2 carol    labadmin     1024 Dec 17 01:28 dirD

drwxr-xr-x   2 janice   users        1024 Dec 17 02:07 dirE

-rwxr-sr-x   1 lu       labadmin     9328 Dec 17 00:59 cat
-rwxr-sr-x   1 carol    labadmin   204188 Dec 17 01:19 edit

-rwsr-xr-x   1 admin    admin       50148 Dec 17 00:59 ls
```

Determine the access rights for any arbitrary user in the system by completing the following table. Use one of the two codes to fill in each entry: "$\surd$" for a permissible operation, or " – " for a forbidden operation.

	list directory files	read a known file in the directory	edit a known file in the directory	create a new file in the directory
dirA				
dirB				
dirC				
dirD				
dirE				