

THE AUSTRALIAN NATIONAL UNIVERSITY
First Semester Examinations 2006

COMP3300
(OPERATING SYSTEMS IMPLEMENTATION)
Final Exam

Writing Period: 3 hours duration
Study Period: 15 minutes duration
Permitted Materials: None
Answer all questions.

Name

Student Number

PLEASE READ THE FOLLOWING INSTRUCTIONS CAREFULLY.

- This paper will be marked out of 100 and consists of 10 questions. Questions are of unequal value. The value of each question is shown in square brackets. Questions that are partitioned into parts show the number of marks given to each part within square brackets. Students should attempt all questions.
- Answer all questions using either a black or blue pen. Use the space provided. Marks may be lost for giving information that is irrelevant. There is additional space at the end of this booklet in case the space provided is insufficient (Clearly indicate, within the question concerned, that you have used this extra space at the end of the booklet.).
- Students are asked to check that this examination paper contains all 18 pages. No pages are to be torn from this examination paper.
- This examination paper is **CONFIDENTIAL** and is not to be taken from the examination room.

Official use only:

1	2	3	4	5	6	7	8	9	10	Total
---	---	---	---	---	---	---	---	---	----	-------

Question 1. [6 Marks] Give a definition of what an operating system is. What are some characteristics of a good operating system?

Question 1.

Question 2. [7 Marks] In the view of the history of operating systems, describe the purpose and function of the resident monitor. What approach was used before the resident monitor? Why was this replaced by the resident monitor?

Question 2.

Question 3. [5 Marks] Given the following code:

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>

int main() {
    char com[100];
    pid_t p;
    int status;

    printf("> ");
    while((scanf("%s", com) == 1) &&
        !(strcmp(com, "bye") == 0)) {
        printf("About to fork : %s\n", com);
        p = fork();
        printf("p : %d\n", p);
        if (p == 0) {
            execve(com, NULL, NULL);
            printf("Wrong!! %s \n", com);
            exit(EXIT_FAILURE);
        }
        wait(&status);
        printf("> ");
    }
    printf("bye.\n");
    return 1;
}
```

If you ran the above program and typed in the following lines `/bin/date`, `asdf`, and `bye` what would you expect to see on the terminal (assume the `date` command is in the `/bin` directory)? What is the relationship between *fork*, *execve*, *exit*, and *wait*? This is a simple form of what sort of program?

Question 3.

Question 3.

Question 4. [10 Marks] Given the below 15 True/False questions, circle either **T** or **F** (but not both). Each question that is correctly answered gains you 1 mark, each question answered incorrectly loses you 1 mark, a question left unanswered neither loses nor gains marks. The final mark for this question is calculated by bounding the sum of marks between 0 and 10. For example, if you answered all questions correctly you would gain 10 (not 15) for this question. If you answer 8 correctly, 2 incorrectly and leave the remaining 5 unanswered you would gain 6/10 for this question.

If you wish to change your answer then cross off both **T** and **F** and clearly state your answer in words. (e.g. "I would like to leave this question unanswered." or "My answer is True.")

The following True/False questions relate to synchronization:

T F : One solution to the multi-process critical section problem is the bankers algorithm.

T F : Hardware support is introduced to make synchronization easier.

T F : A semaphore is a message that passes from one processor to another.

T F : The readers and writers problem involves synchronization of processes that either read or write to a shared data structure.

The following True/False questions relate to virtual memory:

T F : FIFO (which is a stacking page replacement algorithm) will never suffer from Belady's anomaly.

T F : One machine instruction will cause at most one page fault.

T F : Generally, increasing the number of frames reduces the number of page faults.

The following True/False questions relate to memory management:

T F : A smaller page size leads to smaller page tables.

T F : A smaller page size leads to less internal fragmentation.

The following True/False questions relate to UNIX:

T F : *execv* replaces the current process image with a new process image.

T F : *signal* is a system call that notifies a process when a segmentation fault occurs.

T F : In older versions of UNIX */etc/passwd* contained an unencrypted copy of users passwords.

T F : *fork* creates a child process that differs from the parent process only in its PID, PPID, and the return value of the fork call.

T F : In Unix protection domains are associated with files.

T F : In Unix protection domains are associated with users.

Question 5. [10 Marks] Suppose you have an array of internal kernel structures. The kernel performs operations that involve two of the elements of this array at the same time. Originally a lock on the entire array was used to ensure data integrity, however this scaled poorly. Why does it scale poorly? How would you change this synchronization approach to address the scaling problem?

Question 5.

(Continue question 5.)

Question 6. [20 Marks]

a) [4/20] The `mmap()` system call provides a way for a user process to ask the kernel to map some pages from a file or device into the memory space of the process. The return value from `mmap()` is a pointer that points to an area of memory that, when accessed, will cause a page fault and a load of the corresponding page of data from the device or file. In what way is this useful? Provide two examples of uses of `mmap`.

Question 6.

b) [4/20] The `mmap()` call takes the following parameters:

`void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset)`

- The *start* parameter (is usually NULL) indicates the preferred address that the process would like the kernel to return.
- The *length* parameter specifies how much memory to map.
- The *prot* parameter specifies the required protections on the pages of memory. These can be any combination of `PROT_READ`, `PROT_WRITE` and `PROT_EXEC`.
- The *flags* parameter specifies whether the pages should be either `MAP_PRIVATE` or `MAP_SHARED`.
- The *fd* parameter specifies a file descriptor open on the file you want to map.
- The *offset* parameter specifies the location in the file you want to map. You would set this to zero to start the mapping at the start of the file.

Explain the difference between `MAP_PRIVATE` and `MAP_SHARED` ?

Question 6.

c) [16/20] Suppose you have just built a new hardware device which you wish to be able to control from Linux. This device consists of a single LED (Light Emitting Diode), it will sit on top of your monitor and may be used for a variety of purposes (e.g. It could go on when you have unread email in your inbox.) It is given that the device's hardware control and data addresses are memory mapped and at locations 0xfb000000 and 0xfb000001 respectively. Using mmap, write two user space routines "LED_on()" and "LED_off()" that enables you to turn the LED on and off. To turn the LED on you need to write 1 in the device's data register and then 24 (say this is the set command) in the control register. To turn the LED off you need to write 0 in the device's data register and then 24 in the control register. Note that /dev/mem enables you to access all of memory.

Just to jog your memory, below are some randomly selected lines of code from a lab.

```
int sfile, dfile;
sfile = open("mysource", O_RDONLY);
dfile = open("mydestination", O_RDWR|O_CREAT|O_TRUNC);
sdata = mmap(NULL, filesize, PROT_READ, MAP_PRIVATE, sfile, 0);
if (sdata == MAP_FAILED)
ddata = mmap(NULL, filesize, PROT_READ|PROT_WRITE, MAP_SHARED, dfile, 0);
memcpy(ddata, sdata, filesize);
close(sfile);
close(dfile);
```

Question 6.

(Continue question 6.)

Question 7. [8 Marks] Consider the following sequence of memory references from a 5000-byte program:

92, 153, 1027, 1669, 783, 3454, 1852, 2459, 2461, 4340, 4580, 3642.

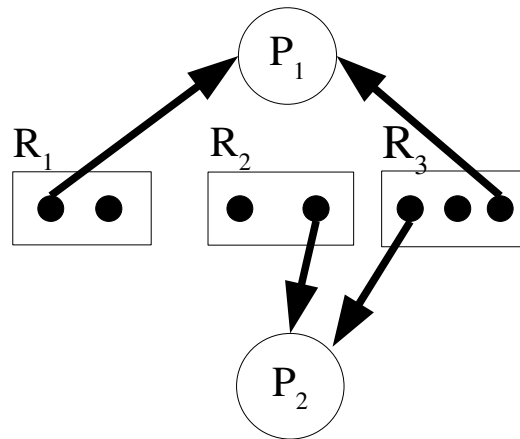
a) [2/8] Suppose the page size is 1000 bytes. What is unusual about this size? What is the reference string?

b) [2/8] Suppose the page size is 1000 bytes. Calculate the page fault rate given there is 2000 bytes of primary memory and that a LRU page replacement algorithm is used. (You may assume that all the primary memory can be used for this program.)

c) [2/8] Suppose that page size is again 1000 bytes. Calculate the page fault rate given there is 2000 bytes of primary memory and FIFO page replacement is used.

d) [2/8] Suppose the page size is 500 bytes. What is the reference string? Calculate the page fault rate given there is 2000 bytes of primary memory and the optimal replacement algorithm is used.

Question 8. [6 Marks] Given the following resource allocation graph:



Fill in the below tables:

	<i>Max</i>			<i>Allocation</i>			<i>Need</i>			<i>Available</i>		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P ₁	2	2	2									
P ₂	1	2	1									

If process P₂ requests one instance of resource R₁ then should this request be granted? Show your working by providing a trace of the Bankers Algorithm. Also list all the requests that may be safely granted.

Question 8.

Question 9. [20 Marks]

Suppose a file system uses a contiguous approach for storing data within files. The file system also uses a bit vector to determine which blocks are free. So the blocks on the disk would be divided up into: blocks for the free map bit vector (starting at block 0), and blocks for the data (starting after the free map bit vector). Please ignore the existence of any super blocks.

a) [4/20] What are some advantages and disadvantages of using a contiguous approach for a file system?

Question 9.

b) [4/20] Given blocks are b bytes long and there are m blocks for this file system on the disk drive. Write an expression that will determine how many blocks need to be allocated to the free map bit vector.

Question 9.

c) [12/20] Assume that the blocks are 512 bytes. Write a routine called, **block_pointer first_fit(block_pointer start_data_blocks, int num_data_blocks, int num_block_required)**

which finds the first contiguous set of free blocks which are sufficiently large for the request, where **start_data_blocks** is the index of the first block in the set of the data blocks, **num_data_blocks** is the total number of data blocks (also indicates the size of the bit vector as each bit maps to exactly one data block), **num_blocks_required** is the number of contiguous blocks required for the routine to find.

If there is a sufficiently large hole to fulfill the request then the index to the first block of the hole is returned, otherwise return -1. You may make your own assumption with regards to the ordering of bits. Also, you may assume the existence of

int read_block(block_pointer b, char *buf)

that reads block **b** and places the result in buffer **buf**. The routine will return the number of bytes successfully read, or -1 if an error has occurred (note that **block_pointer** is just an unsigned long).

Question 9.

(Continue question 9.)

Question 10. [8 Marks] a) [3/8] What is authentication? Authentication is normally based on one or more of three general 'items'. State and illustrate these three 'items'.

Question 10.

b) [5/8] Linux generally uses passwords for authentication. Describe the password authentication mechanism within Linux. What part does the kernel play in this authentication process.

Question 10.

Additional answers to question: ____

Additional answers to question: ____

Additional answers to question: ____