Surname: .................................................. Forenames: ..................................................

ID: ......................................................................

# THE UNIVERSITY OF AUCKLAND

**SECOND SEMESTER, 2006**
**Campus: City**

**COMPUTER SCIENCE & SOFTWARE ENGINEERING**

**Operating Systems**

**(Time allowed: TWO hours)**

**NOTE:**

Attempt ALL questions.

Answer the questions in the spaces provided.

Marks for each question are shown and total **100**.

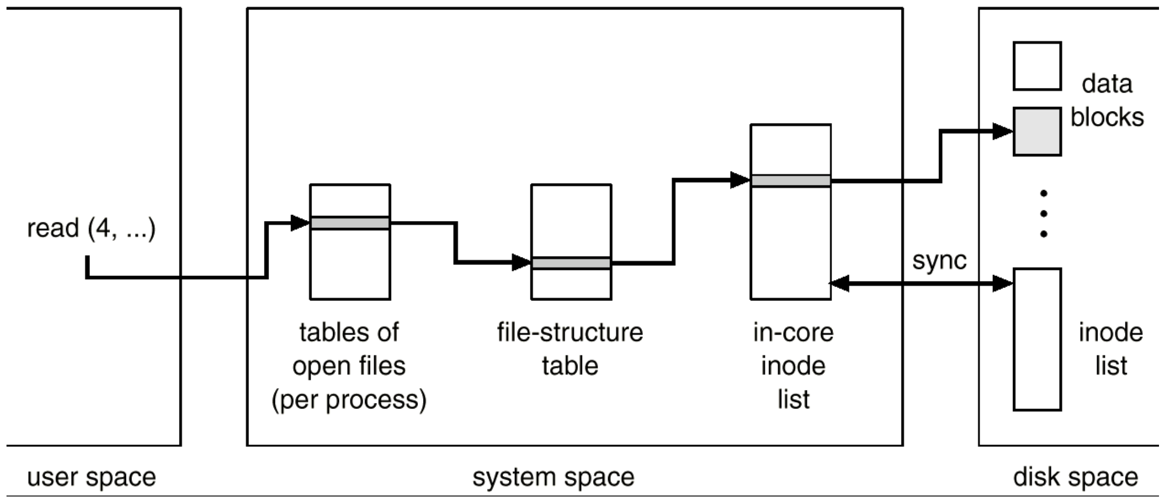
For markers only:

| | | | |
|---|---|---|---|
| *Question 1* | */10* | *Question 6* | */14* |
| *Question 2* | */7* | *Question 7* | */14* |
| *Question 3* | */6* | *Question 8* | */14* |
| *Question 4* | */7* | *Question 9* | */16* |
| *Question 5* | */12* | | |
| | | *Total* | |

**CONTINUED**

ID: ......................................................................

1. File systems [10 marks]



(a) This diagram from the lecture notes and the textbook shows the main data structures used by a Unix file system.
Describe the process of opening a file under Unix. You must mention how the tables and lists in the diagram are changed by the open operation. Assume that the inode corresponding to the file has been found on disk and that the file was not already open.

Read the inode into the in-core inode list. Lock the inode.
Check the permissions to see if the operation is permitted. Clean up and return if not allowed.
Allocate a file-structure table entry with the access type (e.g., append, read, write). This points to the in-core inode entry. Set the reference count in the in-core inode entry to 1.
Allocate an entry in the per-process open file table to point to the corresponding file-structure table entry.
Unlock the inode.
Return the index into the open file table as the file descriptor (handle).

(6 marks)

CONTINUED

ID: ....................................................................

(b) If two unrelated processes have the same file open for appending to the end of the file there is a race condition. The first process could get the position of the end of the file and before it can write it is pre-empted. Describe what could go wrong and give a solution to the problem.

The second process could append at that position and when control returns to the first process it appends data at the same position, overwriting the data from the second process.
Lock the incore inode from the start of the append write call until the write has completed.

(4 marks)

2. Concurrency [7 marks]

The following Ruby program was written by a poor programmer to deposit and withdraw money from an account.

```
account = 0.00

deposit = Thread.new do
    1000.times do
        account = account + 1.00
    end
end

withdraw = Thread.new do
    1000.times do
        account = account - 1.00
    end
end

puts account
```

(a) When this code was run the value printed was `230.0`. Explain how this could have happened.

Two answers are acceptable.
Access to the account variable is uncontrolled. Both threads read and modify it simultaneously, leading to some changes being overwritten.
Also (and the real reason in this case) the main program does not wait for the threads to complete before printing the result. So by the time the “puts” statement is run, the deposit thread is part way through and the withdraw thread hasn’t begun yet.

(3 marks)

(b) When the programmer added the line

**CONTINUED**

ID: ......................................................................

```
deposit.join
```

before the `puts` statement, the value printed became `-624.0`. Explain why the answer was different from before.

The deposit thread now runs to completion. While it is running the withdraw thread also runs. The withdraw thread must be consistently overwriting the account value of the deposit thread.

(3 marks)

(c) Describe what the programmer should do to fix the code?

Lock access to the account variable and make sure both threads complete.

(1 mark)

3. Distributed deadlock [6 marks]

Deadlock detection in a distributed system can suffer from false cycle detection. Describe how this can occur.

Using a centralized deadlock detector.
The sites send their wait-for graphs to the deadlock detector, which generates the system wait-for graph by combining them. Because the graphs from each site are from different times the information in one may be inconsistent with the information in another, making it look as though deadlock has occurred.

(6 marks)

CONTINUED

ID: ......................................................................

4. Scheduling [7 marks]

Here are some processes with their burst times.

| **Process** | **Burst time** |
|---|---|
| A | 5 |
| B | 3 |
| C | 10 |
| D | 1 |
| E | 12 |

(a) Explain what burst time is.

The CPU time the process would use before blocking or terminating.

(1 mark)

(b) Draw a Gantt chart for the processes above with the following scheduling algorithms. Also give the average waiting time of the processes.

(i) First Come First Served (FCFS) schedule

A(0-5), B(5-8), C(8-18), D(18-19), E(19-31)
(0 + 5 + 8 + 18 + 19) / 5 = 10

(2 marks)

(ii) Round-robin schedule with a time-slice of 5

A(0-5), B(5-8), C(8-13), D(13-14), E(14-19), C(19-24), E(24-29, 29-31)
(0 + 5 + (8 + 6) + 13 + (14 + 5)) / 5 = 10.2

(2 marks)

(iii) Shortest-Job First (SJF) schedule (no-preemption)

D(0-1), B(1-4), A(4-9), C(9-19), E(19-31)
(0 + 1 + 4 + 9 + 19) / 5 = 6.6

(2 marks)

CONTINUED

ID: ......................................................................

5. Memory [12 marks]

(a) How much memory is used by a full page table on a system with a virtual address space of 40-bits and pages of size 8Kbytes (2^13 bytes), and where each page table entry is 8 bytes long? Show your working.

8K = 2^13 so 13 bit offset. This leaves 40 – 13 = 27 bits to index the page table.
2^27 x 8bytes = 2^27 x 2^3bytes = 2^30 bytes or 1 gigabyte.

(3 marks)

(b) When a page of memory is accessed but not found in physical memory, a frame must be found for it to be stored in. Different page replacement algorithms and different numbers of frames give different results.
Given this reference string of memory requests 1, 2, 3, 4, 1, 5, 2, 4, 1 complete the following tables showing the contents of memory, using the specified algorithms.

(i) With the First In First Out selection algorithm and 3 frames.

| ***page request*** | ***1*** | ***2*** | ***3*** | ***4*** | ***1*** | ***5*** | ***2*** | ***4*** | ***1*** |
|---|---|---|---|---|---|---|---|---|---|
| frame 0 | 1 | 1 | 1 | 4 | 4 | 4 | 2 | 2 | 2 |
| frame 1 | | 2 | 2 | 2 | 1 | 1 | 1 | 4 | 4 |
| frame 2 | | | 3 | 3 | 3 | 5 | 5 | 5 | 1 |

(2 marks)

(ii) With the First In First Out selection algorithm and 4 frames.

| ***page request*** | ***1*** | ***2*** | ***3*** | ***4*** | ***1*** | ***5*** | ***2*** | ***4*** | ***1*** |
|---|---|---|---|---|---|---|---|---|---|
| frame 0 | 1 | 1 | 1 | 1 | | 5 | | | 5 |
| frame 1 | | 2 | 2 | 2 | | 2 | | | 1 |
| frame 2 | | | 3 | 3 | | 3 | | | 3 |
| frame 3 | | | | 4 | | 4 | | | 4 |

(2 marks)

(iii) With the Least Recently Used selection algorithm and 3 frames.

| ***page request*** | ***1*** | ***2*** | ***3*** | ***4*** | ***1*** | ***5*** | ***2*** | ***4*** | ***1*** |
|---|---|---|---|---|---|---|---|---|---|
| Frame 0 | 1 | 1 | 1 | 4 | 4 | 4 | 2 | 2 | 2 |
| Frame 1 | | 2 | 2 | 2 | 1 | 1 | 1 | 4 | 4 |
| Frame 2 | | | 3 | 3 | 3 | 5 | 5 | 5 | 1 |

(2 marks)

CONTINUED

ID: ......................................................................

(iv) With the Least Recently Used selection algorithm and 4 frames.

| ***page request*** | ***1*** | ***2*** | ***3*** | ***4*** | ***1*** | ***5*** | ***2*** | ***4*** | ***1*** |
|---|---|---|---|---|---|---|---|---|---|
| Frame 0 | 1 | 1 | 1 | 1 | | 1 | 1 | | |
| Frame 1 | | 2 | 2 | 2 | | 5 | 5 | | |
| Frame 2 | | | 3 | 3 | | 3 | 2 | | |
| Frame 3 | | | | 4 | | 4 | 4 | | |

(2 marks)

(c) Which of the four parts above had the smallest number of page replacements?

(ii) and (iv)

(1 mark)

6. Distributed Systems [14 marks]

(a) Distributed file systems can use two different methods to provide file data – remote service or local caching.

(i) What advantages does a remote service distributed file system have over a caching system?

•simpler to implement because of no consistency problem
•uses less local memory (primary or secondary)
•matches local file access – e.g. a read call is turned into a real read of the file

(3 marks)

(ii) What advantages does a caching distributed file system have over a remote service system?

•faster (in the general case) – less frequent use of the server, fast access to data when it is in the local cache
•scales better – the server is not the bottleneck in the same way

(3 marks)

**CONTINUED**

ID: ......................................................................

(b) Memory paging hardware has the ability to flag pages as read-only. This is useful in many distributed processing situations.

(i) Describe how read-only pages can be used to provide distributed shared memory.

On first access the DSM pages are copied from the server to the local machine and are flagged read-only. If the process writes to the page, it causes a page access fault. This gets recognized as a modification to a DSM page. The change is sent back to the server and the local page is modified (also modified when a change is broadcast from the server).

(4 marks)

(ii) Describe how read-only pages can be used to migrate the memory of a running process from one machine to another. The technique minimises the process down time. Page dirty-bits can also be used to accomplish this.

Mark all pages read-only and start the memory transfer to the destination machine. As the process modifies pages it will cause page access faults which are used to keep track of pages which need transferring a second time as they have been altered. When all original pages have been transferred, halt the process and copy the dirty pages.

Could also just use read-only pages to copy them first dealing with dirty (writable) pages later.

(4 marks)

7. Security & Protection [14 marks]

(a) Ruby produces a warning message when performing an `exec` or similar system call if there is a world writable directory in the user's path. Explain what is wrong about having a directory in your path which anyone can write to.

If anyone can write to this directory they can put any program they like there. If the program has the name of a command you want to run, then you may run it inadvertently and it assumes your privileges. All of your files and processes are then compromised.

(4 marks)

**CONTINUED**

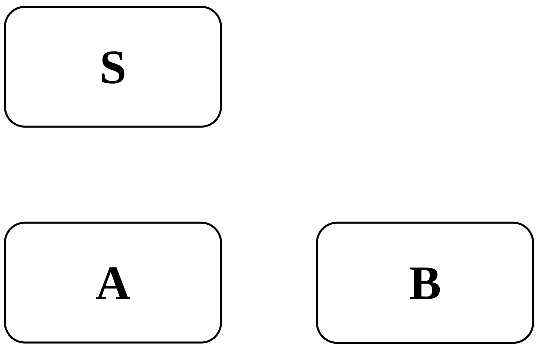
ID: ......................................................................

(b) Imagine a capability-based protection scheme where the capabilities of a process are stored on disk and loaded into memory when the process is running. Describe two different ways the integrity of the capabilities could be maintained.

Only allow the OS to read and write the capability store for the process. Read them in to system memory.
Encrypt the capability so that any modification will invalidate it. Then only allow the process itself access to its store of capabilities.

(4 marks)

(c)



By referring to this diagram briefly describe the Needham-Schroeder Protocol. (You may add extra information to the diagram to help your description.)

(6 marks)

**CONTINUED**

ID: ......................................................................

8. Assignment 2 [14 marks]

Here is part of the `start.rb` program from Assignment 2.

```
PHILOSOPHERS = 5

# Start the ring server.
exec("ruby", "ring_server.rb") if fork.nil?

sleep 2 # solution to problem one

# Put the forks on the table.
exec("ruby", "table.rb",  PHILOSOPHERS.to_s) if fork.nil?

sleep 2 # solution to problem two

# Start the philosophers.
PHILOSOPHERS.times do |n|
  exec("ruby", "philosopher.rb", PHILOSOPHERS.to_s,  n.to_s) if fork.nil?
end
```

Two `sleep` methods were called to solve two problems. The first was to allow the tuplespace server to start running before any processes used it, the second was to make sure the forks were on the table before the philosophers ran.

(a) Using `sleep` is not a good solution to these problems; explain why not.

It is possible that even after the sleep time the state may not be the required one, e.g., the tuplespace server may not have started completely.

(2 marks)

(b) How else could the first problem be solved?

This is difficult because the only shared resource for synchronization is the tuplespace itself. Therefore unless we resort to some other mechanism, such as another server, we have to put error handling code in our processes. If they attempt to access a non-existent tuplespace they catch the error and try again.

(6 marks)

CONTINUED

ID: ......................................................................

(c) How else could the second problem be solved? In this case you may assume that the tuplespace server is already running. Include a description of messages you could send to the tuplespace and read from the tuplespace in order to make your solution work.

A better solution would have the philosophers block until the forks were on the table. This could be done by having the philosophers do:
`tuplespace.take([:start, phil_number])`
with `phil_number` representing the particular philosophers.
After putting the forks on the table, the table program should also call:
`tuplespace.write([:start, phil_number])` for each philosopher
In reality this wasn't needed, as waiting for a fork would cause the philosophers to block in just this way.

(6 marks)

9. Assignment 3 [16 marks]

The following statements use a correct `file_system.rb` program from Assignment 3.

```
require 'file_system'

disk = Disk.format("drive1.txt", 32)
volume = Volume.format(disk, "question volume")
file1 = volume.open("file 1")
file2 = volume.open("file 2")
file1.write(0, "1")
file2.write(0, "2")
file2.write(10, "2")
puts file1.size
puts file2.size
puts file1.read(0, file1.size)
puts file2.read(0, file2.size)
volume.unmount
```

(a) Show the output from the above statements.

```
1
11
1
2        2
```

(4 marks)

**CONTINUED**

ID: ......................................................................

(b) Explain in detail what the two different format methods do.

Disk.format
Creates a text file to be used as the virtual disk.
Allocates blank spaces of 64 bytes for each virtual block and writes separator marks between the blocks.

Volume.format
Adds volume information to the virtual disk.
The volume information includes:

- The number of blocks for the volume information (1 in this case).
- The name of the volume (“question volume”).
- The number of blocks in the volume (32).
- The volume bitmap.
- The number of the root directory first index block (31).

Fills in some information in block 31 (this was variable, in the sample solution it was 16 zeros to indicate no blocks had been allocated to the root directory yet).

(8 marks)

(c) For a file to be stored on the disk the file system needs to keep track of the blocks which are used by the file. Describe one method of storing this information on the disk which allows a file to grow to any size, limited only by the size of the disk.

Linked blocks.
Linked index blocks.
Linked extents etc.

(4 marks)