Question/Answer Sheet COMPSCI 340 & SOFTENG 370

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

ID: .....

THE UNIVERSITY OF AUCKLAND

SECOND SEMESTER, 2008
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 5*	*/10*
Question 2	*/14*	*Question 6*	*/9*
Question 3	*/12*	*Question 7*	*/9*
Question 4	*/22*	*Question 8*	*/14*
		Total	

CONTINUED

ID: .....

1. Threads and Processes [10 marks]

(a) Describe two advantages user-level threads have over system-level threads.

Works even if the OS doesn't support threads.
Easier to create - no system call.
Control can be application specific.
Easier to switch between - saves two processor mode changes.

(2 marks)

(b) Describe two advantages system-level threads have over user-level threads.

If one thread blocks the other threads in the process can continue.
Each thread can be scheduled separately.
On a multiprocessor different threads can be scheduled on different processors.

(2 marks)

(c) How many processes are started by the following Ruby programs running on a Unix machine? Include the starting process in your answer. Remember that in Ruby the `fork` system call returns `nil` in the child process. You may show your working on the next page.

(i)
```
puts Process.pid
n = 0
while n < 2 do
  if fork.nil?
    puts "#{Process.pid} - #{Process.ppid}"
  end
  n += 1
end
```

4

(3 marks)

(ii)
```
puts Process.pid
n = 0
while n < 2 do
  if fork.nil?
    puts "#{Process.pid} - #{Process.ppid}"
    if fork.nil?
      puts "#{Process.pid} - #{Process.ppid}"
    end
  end
  n += 1
end
```

9

ID: .....

(3 marks)

Space for working on question 1. (c) (i)

Space for working on question 1. (c) (ii)

ID: .....

2. File Systems [14 marks]

(a) How files are stored on disk and the way the file system keeps track of the blocks of a file are called "file space allocation methods". Some file space allocation methods are suitable for direct access to data in files and some are suitable for sequential access to data in files. Describe a file space allocation method that is suitable for sequential access but not suitable for direct access. Explain why it is not suitable for direct access.

Linked list allocation with each section pointing to the next section in the file. To move to any particular byte in the file the sections (or blocks) of the file need to be traversed in order. For direct access this could be a very large number of disk accesses in order to find the required information.

(6 marks)

(b) Briefly describe two ways of keeping track of free blocks on a disk device.

Any two of:
Linked list of free blocks. There is a pointer to the first free block. Each free block points to the next one.
Bitmap. A bitmap with each bit representing a block on the disk. A 1 (or zero) represents a free block.
Start points and lengths. An array (or list) of pointers to the starting point of a section of free blocks and the length of the section.
Any other method that seems reasonable.

(8 marks)

ID: .....

3. Distributed File Systems [12 marks]

(a) Describe what is meant when a distributed file system is called stateless.

The file server does not maintain information about the clients or the clients requests. This means that every request from the client has to include extra information.

(3 marks)

(b) Describe what is meant when a distributed file system is called stateful.

The file server keeps information about all processes accessing the files, including information about where the process was up to when reading a file sequentially.

(3 marks)

(c) Here are some automount maps from NFS on a machine called client1.

```
File: auto_master
Contents:
         */home auto_home

File: auto_home
Contents:
         *sally server1:/export/home/sally
         *greg server1:/export/home/greg
         *tom server2:/export/home/tom
         *grace server2:/export/home/grace
```

(i) Draw the directory tree on `client1` showing all of the directories mentioned in the maps.

ID: .....

```
/ --- home --- sally
           --- greg
           --- tom
           --- grace
```

(4 marks)

(ii) On what machine and in what directory are the files in `/home/grace` actually stored?

Machine - server2. Directory - /export/home/grace

(2 marks)

4. Assignment 2 & Deadlock [22 marks]

(a) Why did we have to create a new lock class DMutex in assignment 2 rather than using the standard Ruby Mutex lock class?

The Mutex class has no way of specifying who the owner of a lock is and also the waiting list of threads is not accessible from outside the class. We needed a lock class which provided these to the deadlock detection routines.

.....

.....

.....

(4 marks)

(b) Given the following lock method for the DMutex class, write the corresponding unlock method.

```
def lock
  @private_lock.synchronize do
    if @locked
      @waiting << Thread.current
      @condition_variable.wait(@private_lock)
      # it is possible that some dead threads have to be thrown away
      thread = @waiting.shift until thread == Thread.current
    end
```

CONTINUED

ID: .....

```
        @owner = Thread.current
        @locked = true
      end
    end
```

```
def unlock
    @private_lock.synchronize do
        @condition_variable.signal
        @locked = false
        @owner = nil
    end
end
```

(6 marks)

(c) The Philosopher objects in assignment 2 each ran in their own processes. The Table object was in another process. The forks were represented by DMutex lock objects and were in the Table process. Explain how the Philosopher objects pick up and put down the forks. Mention any messages and the threads and processes involved.

A philosopher has a distributed Ruby (DRb) connection to the Table process. It sends a message to the Table object in that process via a DRb method call. The parameters are marshalled and sent with the message. At the Table process a thread handles the request, the message is unmarshalled and the pick up or put down method is called locally. The requesting Philosopher is blocked until it gets the result back from the Table. The result is marshalled and sent back to the Philosopher object and process.

(4 marks)

(d) Wait-for graphs can be used to detect deadlock. What is a wait-for graph and how does it show deadlock?

A wait-for graph is a graph where each node is a process or thread and each edge goes from a thread/process waiting on a resource exclusively held by the thread/process it points to. It shows deadlock when a cycle exists in the graph.

CONTINUED

ID: .....

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

(4 marks)

CONTINUED

ID: .....

(e) A Hash or Map object can be used to efficiently manipulate a wait-for graph. In this case the key is the thread waiting on the exclusive lock and the value is the corresponding "owner" of the lock. The following code demonstrates how this graph could be produced in assignment 2.

```
wait_for = Hash.new
DMutex.all_dmutexes.each do |lock|
    owner = lock.owner
    waiting = lock.waiting
    waiting.each { |thread| wait_for[thread] = owner }
end
```

The `DMutex.all_dmutexes` method returns a list of all DMutex locks in the process. The `owner` method of a DMutex returns the thread which currently holds the lock, and the `waiting` method returns a list of all threads currently waiting for the lock.

Unfortunately this code has a race condition. Even with a perfectly correct DMutex class this code occasionally produced a key, value pair where a thread waiting on the lock appeared to be the owner of the lock. Explain how this could happen and describe a possible solution to the problem.

The owner and waiting values are collected at different times. In between times the original owner may have released the lock and tried to gain it again and now be on the waiting list. So it appears that the owner is waiting for itself. The solution is to provide mutual exclusion over the retrieval of both bits of information, so that they are always consistent.

(4 marks)

ID: .....

5. Process Migration [10 marks]

(a) Name, and very briefly discuss, four types of information which must be contained in the request message sent from an RPC client stub to a server stub.

* Name (of the procedure being called)
* Parameters (these are normallypassed by value)
* Timestamp (to allow "at most once" semantics)
* Source (an identifier for the RPC client: typically a port number)
* Version (to support newer RPC protocols without disabling the older ones)
* Format (to indicate what data translations are necessary e.g. big-Endian to little-Endian)

(5 marks)

(b) Some RPC packages offer two types of remote calls. The EO calls have "exactly once" semantics, and the AMO calls have "at most once" semantics. You are writing a distributed program that uses a very large, but very unreliable, set of RPC servers. Each server only stays up for about an hour between crashes. A server takes two minutes to reboot after each crash. Your RPC client software runs on a very reliable computer. You decide to replicate all your RPC calls, sending identical RPC requests to two different servers. Your client will accept the result from the server which responds first. Each of your calls takes about 1 minute to complete -- but will return a result only if a server stays up for that amount of time. Would you use the AMO calls or the EO calls? Explain.

The AMO calls are more appropriate, if I were trying to write efficient code. When just one of the two servers handling a replicated call crashes, there would be no advantage in reissuing the RPC request to that crashed server after a timeout, as would be done by an EO call. If neither server crashes, there is no difference in the EO and AMO semantics. However, to handle the case that neither server returns a result from a replicated AMO call, I would have to write my own timeout-and-retry code. This is moderately difficult to do correctly. So, if were in a huge hurry to get my code working correctly, and if I were not very worried about computational efficiency, then I would use the EO calls even though this seems likely to result in a server doing some useless work after each reboot.

(5 marks)

6. Paged Memory [9 marks]

Consider a tiny computer system with a 32-bit virtual address space and 1 GB (= 2^{30}) of physical memory. Each frame is 1 MB.

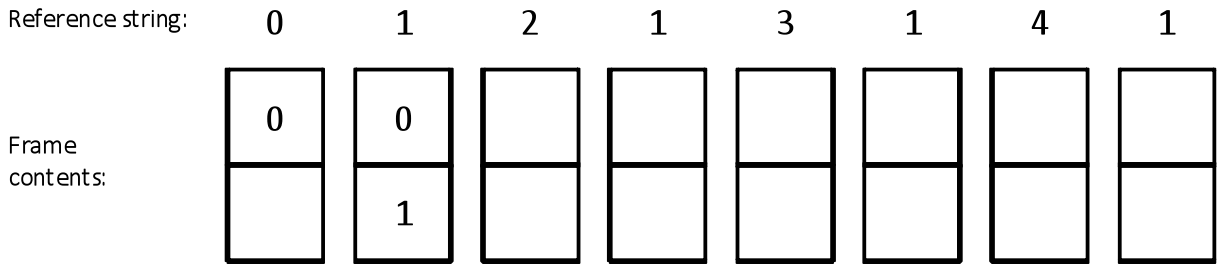
(a) How many entries are in the page table of the tiny computer?

ID: .....

4096 (or 4K).

(2 marks)

(b) Each user-mode process on the tiny computer is allocated just two frames of physical memory. A user-mode process P1 running on the tiny computer has the following reference string: (0, 1, 2, 1, 3, 1, 4, 1). Complete the following diagram to indicate the contents of the two page frames after each reference. Assume that the page-replacement algorithm is FIFO.



(3 marks)

(c) A new OS is being developed for the tiny computer. This OS will maintain a 2-level page table for each user process. The superpages are 1 MB in size, and the subpages are 1 KB. There is no TLB in the tiny computer, however its memory-system hardware is able to read and interpret the level-1 and level-2 PTEs in the 2-level page table. Describe one performance advantage, and describe one performance disadvantage, of the new OS over the older OS.

Every user-level memory operation under the new OS will require three memory operations: two to read PTE1 and PTE2, and one to fetch (or store) the user's data or instruction. This is a significant disadvantage, and may cause the new OS to run processes 50% slower than the old OS (which required only a single PTE read for each user-mode memory operation). The main advantage of the new OS is the much smaller page size, which should greatly decrease the page-fault rate of most programs.

(4 marks)

ID: .....

7. Access Control [9 marks]

(a) A kernel process P1 is the reference monitor for user process P2.
User process P2 has requested, and has obtained, read access to file F1.
User process P2 also has obtained write access to file F2.
In the space below, draw an access matrix for this system. Briefly discuss the non-blank entries in your matrix. Be sure to indicate the domains that P1 and P2 are running in.

P1 is in domain D1, P2 is in domain D2. I have made it the owner of files F1 and F2, because it's pretty clear that P2 can't be the owner (since it had to obtain access from P1 for these files). The kernel P1 has control rights over all domains, including its own. The user process P2 is able to read F1, and it is able to write F2, but it has no other rights.

D\O	F1	F2	D1	D2
D1	o	o	c	c
D2	r	w	-	-

(5 marks)

(b) You are a systems administrator for a very large bank. The bank president has asked you, urgently, to revoke all access rights for a formerly-trusted employee -- a systems operator -- who is suspected of fraud. The immediate worry is that this employee might overwrite entire file systems, causing large losses to the bank, in an attempt to obliterate all traces of their prior fraudulent activity. Authentications for many of your bank's systems are handled through Kerberos. In the space below, explain the relevant TOCTTOU vulnerability of Kerberos.

Kerberos does not have a revocation mechanism. The formerly-trusted employee may have very recently obtained a KAS ticket, authenticating him as a user on the system for the next 24 hours. Furthermore, the formerly-trusted employee may already have a valid TGS ticket for an important file system. This is a TOCTTOU threat: at the time-of-check by Kerberos, the employee was still trusted, but this doesn't imply that the employee will still be trusted at the time-of-use

(4 mark)

ID: .....

8. Devices [14 marks]

(a) Buffers are used in a variety of ways in operating systems. Describe two ways in which buffers are used when dealing with devices. Explain how the buffers help in each case.

Any two of
To help with speed mismatches. A slow device might buffer its output so that a fast device doesn't have to deal with it until there is a reasonable amount of data.
To help with different data transfer sizes. The buffer gets data from one device and dishes it out to the other device in chunks it can deal with.
To cache data. Data from a disk drive could be kept in a buffer cache until the space is needed for something else. In the meantime any access of that data can come from the buffer rather than having to access the disk again.
To preserve copy semantics. Data from a user level buffer can be copied to a kernel buffer on a write call to stop the user modifying the data after the call and before the actual write.
etc.

(8 marks)

(b) Unix rather simplistically separates devices into two main categories, block devices and character devices. Give an example of each type of device and explain the major difference between the two categories.

Block - disks, tapes, memory regions, etc
Character - keyboard, terminal (tty), mouse, anything USB, etc

The main difference is that block devices use the block-buffer cache for communication with the device. IO is carried out in block sized chunks via these buffers. Whereas with character devices IO is usually unbuffered and is written to or read from the device immediately.

(6 marks)