Name:
Login (UPI):

COMPSCI 340SC & SOFTENG 370SC 2010
Operating Systems Test

Monday 23rd August, 9:05am – 9:55am

- Answer all questions in the spaces provided.
- The test is out of 60 marks. Please allocate your time accordingly.
- Make sure your name is on every piece of paper that you hand in.
- When you are asked to explain something or to give reasons for something you can give your answers as a series of points. Be brief.
- The last page of the booklet may be removed and used for working.

Name:

Login (UPI):

For markers only:

Question 1	*/7*		
Question 2	*/6*		
Question 3	*/8*		
Question 4	*/10*		
Question 5	*/3*		
Question 6	*/12*		
Question 7	*/14*	*Total*	

Name:
Login (UPI):

Question 1 – History and Development of Operating Systems (7 marks)

a) Computer systems now provide memory protection. Describe two different ways things could go wrong if there was no memory protection of the **operating system code**?

The operating system code could be compromised and made to do anything, by-passing any security or protection arrangements for users, processes or resources.

The operating system code could be corrupted causing the computer to crash.

4 marks

b) Apart from memory protection to prevent user-level code from directly accessing the operating system memory what else do processors require in order to safeguard operating system code from user-level code? Also explain what could happen if a processor did not have this ability.

Privileged instructions.

Without privileged instructions user-level code would be as powerful as operating system code and could do anything it wished, such as modifying the memory protection registers or tables.

3 marks

Question 2 - Design and Implementation (6 marks)

a) Give two reasons why C is used to implement many operating systems?

It provides direct access to memory addresses - useful both for memory and device registers.

It maps easily to machine instructions on most processors.

It has small runtime requirements. It is portable. Etc.

It is possible to give hints to the compiler to use registers for particular values and has storage class qualifiers (“volatile”) which are useful for concurrent processing.

2 marks

Name:
Login (UPI):

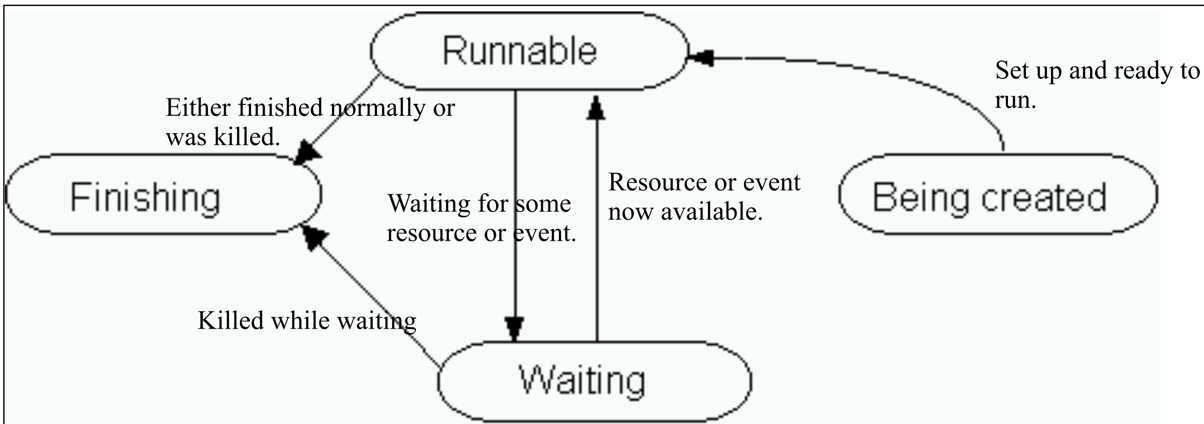
b) Describe *operating system level virtualization* and how it differs from traditional virtual machine technology such as VMWare player and IBM's VM operating system.

Operating system level virtualization provides several different containers which look
and work like separate machines, similar to traditional VMs, but they are all based on
the same kernel so that only one kernel provides the underlying services. They are less
flexible than traditional VMs because all virtual machines have to run the same kernel
but they are more efficient because the host and guest operating systems are the same.

4 marks

Question 3 – Processes (8 marks)

a) Draw a diagram showing the states a process (or thread) goes through. Show and briefly describe the transitions between the states.



4 marks

Name:
Login (UPI):

This C program runs on Unix.

```
int main(int argc, char** argv) {
    int i;
    for (i = 0; i < 2; i++) {
        printf("One\n");
        fork();
        printf("Two\n");
    }
}
```

b) How many times would this program print "`One`" to the display?

3

2 marks

c) How many times would this program print "`Two`" to the display?

6

2 marks

Question 4 – Scheduling (10 marks)

a) Here are the arrival and burst times for a number of processes:

Process	Arrival time	Burst time
A	0	7
B	1	5
C	2	4
D	3	1
E	6	2

From this table draw a Gantt chart showing a Shortest Job First schedule with pre-emption and calculate the average waiting time. If the times are the same do NOT pre-empt the running process.

0-1:A 1-3:B 3-4:D 4-7:B 7-9:E 9-13:C 13-19:A

average wait time: (12 + 1 + 7 + 0 + 1) / 5 = 21/5 = 4.2

5 marks

Name:
Login (UPI):

b) Given the following real-time processes calculate a cyclic schedule using Least Slack Time. If the slack times are the same, do NOT unnecessarily pre-empt the running process. If the slack times are the same for a number of non-running processes, choose the alphabetically lowest. e.g. If at time 9 both Process B and Process D have the same amount of slack time and neither process was running at time 8 then choose B. Show the schedule as a Gantt chart. The three numbers are Compute time, Period, and Deadline.

Process A (3, 8, 8) Process B (2, 6, 6) Process C(3, 12, 12) Process D(1, 24, 24)

0-2:B 2-5:A 5-7:C 7-9:B 9-10:C 10-13:A 13-15:B 15-16:C 16-18:A 18-19:B 19-21:C 21-22:A 22-23:B 23-24:D

5 marks

Question 5 - Interprocess Communication (3 marks)

What is a Unix pipe and what are its limitations?

A Unix pipe is a communications mechanism between processes.

It has a finite size, but its greatest limitation is that it can only be used between related processes because the way to refer to a pipe is with a file descriptor (which are shared between parent and child processes).

3 marks

Name:
Login (UPI):

Question 6 – Concurrency (12 marks)

a) Explain why the following lock does not work? `locked` is a boolean which indicates the lock is currently locked.

```
while (locked) {
    // do nothing
}
locked = TRUE;
```

If two threads retrieve the value of “locked” in the while statement at the same time and it is false then they would both set it to true and carry on to the critical section. The problem is that reading the value of “locked” is separate from setting it.

2 marks

b) Describe a hardware assisted solution to the problem in part a), how does it solve the problem?

By providing an atomic instruction on a processor such as testandset the value of the “locked” variable is returned at the same time as it is set. This means there is no way two threads can find the lock free at the same time.

2 marks

c) Condition variables are used in monitors. Explain what a condition variable is and why it is used?

A condition variable is a queue which threads wait on in a monitor until some state change which means they can progress. There are two functions associated with condition variables - wait and signal.

Condition variables are used to allow threads to wait inside a monitor until the state has changed, e.g. a buffer is emptied, which means the thread can continue.

4 marks

Name:
Login (UPI):

d) Condition variables are also available for use with pthreads (with type `pthread_cond_t`). Here are the relevant sections from the *man* pages of the *wait* and *signal* operations on pthread condition variables.

```
NAME
     pthread_cond_wait -- wait on a condition variable

SYNOPSIS
     int
     pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);

DESCRIPTION
     The pthread_cond_wait() function atomically unlocks the mutex
argument and waits on the cond argument. Before returning control to the
calling function, pthread_cond_wait() re-acquires the mutex.


NAME
     pthread_cond_signal -- unblock a thread waiting for a condition
variable

SYNOPSIS
     int
     pthread_cond_signal(pthread_cond_t *cond);

DESCRIPTION
     The pthread_cond_signal() function unblocks one thread waiting for
the condition variable cond.
```

e) If a thread calls `pthread_cond_wait` what has to happen before the call is returned from?

Another thread must call pthread_cond_signal on the same condition variable.

2 marks

f) Why does the `pthread_cond_wait` function unlock the mutex before waiting on the condition variable?

Because that mutex is the one locking the shared resource. The waiting thread must unlock this so that another thread can possibly enter the critical section and eventually call signal to restart the waiting thread.

2 marks

Name:
Login (UPI):

Question 7 – Assignment 1 (14 marks)

a) Here is the output from the `A1part1` program when it creates the threads and then *forks* on Linux:

```
creating thread 0
creating thread 1
parent #1888 process #1993 pthread #7ffaf4d5f910 thread #1995 count #1
parent #1888 process #1993 pthread #7ffaf5560910 thread #1994 count #0
process 1993 joined thread 0
process 1993 joined thread 1
process 1996 joined thread 0
process 1996 joined thread 1
```

Here is the output from the `A1part1` program when it *forks* before creating its threads on Linux:

```
creating thread 0
creating thread 1
parent #2332 process #2333 pthread #7fb3819eb910 thread #2334 count #0
parent #2332 process #2333 pthread #7fb380fd3910 thread #2335 count #1
process 2333 joined thread 0
process 2333 joined thread 1
creating thread 0
parent #1888 process #2332 pthread #7fb3819eb910 thread #2336 count #0
creating thread 1
parent #1888 process #2332 pthread #7fb380fd3910 thread #2337 count #1
process 2332 joined thread 0
process 2332 joined thread 1
```

Explain the important differences.

In the first one the two threads only run in the parent process. The output from the threads is only seen once.

In the second one the processes are forked first and then the threads are created.

This means that two threads run in each of the processes so that we see the output of four threads.

4 marks

b) Explain what happens to the threads when a process with multiple pthreads calls *fork* from one of its threads in Linux.

The only thread which continues to run in the child process is the one which called fork.

2 marks

Name:
Login (UPI):

c) Some Unix-based systems release (unlock) `pthread_mutex` locks when a process *forks*. Describe an advantage of doing this and a disadvantage.

Advantage: Threads in the child process (originally only one) will not block if they access
a pthread_mutex which was locked in the parent process. If the pthread_mutexes are
still locked after the fork by a thread (which wasn't the thread which called fork) then any
thread which attempts to get the lock will block forever.
Disadvantage: If a mutex was held by a thread in the parent process it is because it is in
a critical section. Forcing the mutex to be released in the child process allows another
thread to access the critical region which is possibly in an unsafe or inconsistent state.

4 marks

d) The man page for `pthread_atfork` contains the following information:

```
NAME
     pthread_atfork -- register handlers to be called before and after
fork()

SYNOPSIS
     int
     pthread_atfork(void (*prepare)(void), void (*parent)(void), void
(*child)(void));

DESCRIPTION
     The pthread_atfork() function is used to register functions to be
called before and after fork() The prepare handler is called before
fork(), while the parent and child handlers are called after fork() in
the parent and child process, respectively. Any of the handlers may be
NULL.
```

Explain how you can use `pthread_atfork` to ensure that critical sections of code in the *parent* process are in a consistent or safe state in the *child* process.

The prepare function handler needs to grab all of the locks which threads in the
child process will possibly access. In this way the fork is actually delayed while any
thread is in a critical section and when the fork is carried out the states of all critical
sections should be consistent.
The parent and child handlers could both release the locks gained by the prepare
handler. All shared resources will be in a safe state and can be used by threads in both
the parent and child processes.

4 marks

Name:
Login (UPI):

Overflow space for answers.

Name:
Login (UPI):

Overflow space for answers.

This page may be used for working.