Name:
Login (UPI):

COMPSCI340SC & SOFTENG370SC 2008

Operating Systems Test

Tuesday 26th August, 6:30pm – 7:30pm

- Answer all questions in the spaces provided.
- The test is out of 70 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	*/15*	
Question 2	*/12*	
Question 3	*/10*	
Question 4	*/6*	
Question 5	*/11*	
Question 6	*/16*	
	Total	

Name:
Login (UPI):

Question 1 – History and development of Operating Systems (15 marks)

a) Give three hardware changes that needed to be made in the first computers before multiprogramming could be used safely and efficiently. Describe why each change was necessary.

More memory – to hold more than one program at a time

Interrupts – to allow processing to continue while IO operations were carried out.

Memory protection – to keep the memory of each program safe from the other programs.

Also accept disk drives – to enable faster access to programs and data.

6 marks

b) It was common for batch systems to provide the user with several queues to submit jobs to the system. What purpose did the queues have? Give an example.

Different queues signified different resource limitations. e.g. The amount of CPU time allocated to the job, the number of lines of printer output, the number of tapes required to be loaded to run ...

2 marks

c) What is a microkernel and how does it differ from a monolithic kernel?

A kernel which provides a reduced number of services (at least message passing and some interrupt handling). Many services can be provided by user level processes. Requests for service are made via message passing.

Whereas a monolithic kernel has most services compiled into one program. Different parts of the operating system can directly access other parts without the need for message passing.

3 marks

Name:
Login (UPI):

d) Early microcomputer or personal computer operating systems did not require user authentication and security was non-existent. Explain why and then describe the changes that occurred in the use of these systems which required security to be taken seriously.

They were single user systems. The only information that could be tampered with belonged to the user.

One change was implementing multi-user systems on PCs; another was networking. As soon as the information for several people could be accessed by the system, authentication had to be provided and used as the basis for security.

2 marks

e) An emulator is a program which mimics the instruction set of a real processor. In this way a program designed to run on one architecture can be run on another. Explain how Virtual Machines as discussed in this course are different from emulators.

A VM must run on the actual hardware as much as possible in order to maintain the performance requirement. An emulator usually doesn't do this and will be correspondingly slower.

2 marks

Question 2 – Processes (12 marks)

a) The following Ruby program is run on my computer:

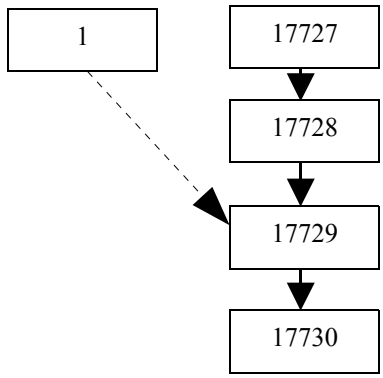
```
def recursive_fork(n)
  if fork.nil?
    puts "child: #{Process.pid}, my parent: #{Process.ppid}"
    if n > 0
      recursive_fork(n - 1)
    end
    puts "process: #{Process.pid}, my parent: #{Process.ppid}"
  end
end

recursive_fork(2)
```

Name:
Login (UPI):

And it produces this output:

```
child: 17728, my parent: 17727
child: 17729, my parent: 17728
child: 17730, my parent: 17729
process: 17730, my parent: 17729
process: 17730, my parent: 17729
process: 17730, my parent: 17729
process: 17729, my parent: 17728
process: 17728, my parent: 17727
process: 17729, my parent: 1
```

Draw a graph showing the relationships between all of these processes. Clearly label each node with its process id.



4 marks

b) In part a) why does the line
`process: 17730, my parent: 17729`
appear three times?

By the time process 17730 is created the code is down 2 levels of recursion. When the recursive calls return they also print the output. Similarly with 17729(but one less level).

3 marks

c) In part a) why does process 17729 have two different parents?

It real parent was process 17728. When this process has completed process 17729 is still running. When a process completes its child processes get given init (process 1) as a replacement parent process.

2 marks

Name:
Login (UPI):

d) Does the code in part a) produce any zombie processes? Explain your answer.

Probably. Process 17730 would be a zombie for a very short period of time after finishing and before its parent process 17729 has finished.

3 marks

Question 3 – Scheduling (10 marks)

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

Process	Arrival time	Burst time
A	0	6
B	2	4
C	3	1
D	7	3
E	8	2

From this table draw a Gantt chart showing a Shortest Job First schedule and calculate the average waiting time. If burst times are the same do NOT pre-empt the running process. Do pre-empt the running process if a newly arriving process has a shorter burst time.

Also use this space for working.

```
A A A C A A A D D D E E B B B B
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

Average waiting time = (1 + 10 + 0 + 0 + 2)/5 = 2.6

5 marks

b) Given the following real-time processes calculate a cyclic schedule using Least Slack Time (LST). If the deadlines are the same, do NOT unnecessarily pre-empt the running process. Check every time unit and pre-empt the running process as soon as another process has a smaller slack time. Show the schedule as a Gantt chart. The three numbers are Compute time, Period, and Deadline.

Process A (3, 8, 8) Process B (3, 6, 6) Process C(3, 24, 9)

Name:
Login (UPI):

Also use this space for working.

```
B B B A A C C A C B B B A A A B B B B A A B B A
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
```

5 marks

Question 4 – Deadlock (6 marks)

a) Deadlock can most easily be *prevented* with one very simple scheme. What is the scheme and how does it work?

Resource (or process) ordering. A process can only acquire resources in a particular order. It prevents cycles occurring.

3 marks

b) Another way of preventing deadlock is the removal of locks using lock-free algorithms. What is a lock-free algorithm and what problems do they have?

A lock-free algorithm provides multiple threads with access to a resource without using locks and without corrupting the resource.

They are difficult to write correctly and unless they are wait-free can lead to indefinite postponement.

3 marks

Question 5 – Concurrency (11 marks)

a) The semaphore wait and signal operations are defined as indivisible (or atomic) operations. Why do they have to be indivisble?

If they weren't indivisble it would be possible for multiple waits and signals to occur simultaneously (on a multiprocessor) or at least interleaved with each other. This would corrupt the value of the semaphore and prevent it working properly.

2 marks

Name:
Login (UPI):

b) Because they are indivisible does that mean that all other processes running on a multiprocessor system must stop when a wait or signal operation is executed? Explain why or why not.

No. Strictly only processes accessing the same semaphore at that time need to stop. Any processes not wanting to change or access the semaphore should be allowed to continue.

3 marks

c) Describe one scheme to implement mutual exclusion in a distributed environment. Say how a process requests exclusive access to the resource, how that exclusive access is maintained and how the resource is made available to a new process when the current one has finished with it.

Either a centralized or distributed solution is acceptable. See the notes.

6 marks

Name:
Login (UPI):

Question 6 – Assignment 1 (16 marks)

a) What output is produced by the following program (with a correctly functioning `process_message_system`)?

```
require 'process_message_system'

consumer = MessageProc.fork do
  loop do
    receive(
      message(:stop) do
        exit
      end,
      message(:data) do |value|
        puts value
      end
    )
  end
end

6.times do |value|
  consumer.give(:data, value)
end
consumer.give(:stop)
```

0 1 2 3 4 5

2 marks

b) How many processes are started (including the original process) by the code in part a)?

2

1 mark

c) What happens if the `:stop` message is never sent in part a)?

The consumer never terminates. It will stay alive but be blocked waiting for another message.

3 marks

Name:
Login (UPI):

d) Here is the `give` method from my sample solution. Explain what is happening on each numbered line.

```
1:class Fixnum

2:  def give(*message)
3:    unless @pipe_wr
4:      @pipe_wr = File.open("/tmp/pipe#{self}", File::WRONLY)
      end
      begin
5:      Marshal.dump(message, @pipe_wr)
6:    rescue IOError
      end
    end

  end
```

1: Opening up the Fixnum class to add the give method. This way a process id can be used as the destination.

2: The give method definition. It takes a variable number of parameters – usually a message identifier and message contents.

3: Only opens the pipe once. If it already is open it does nothing.

4: Opens the pipe for writing.

5: Serializes the message object (an array actually) and writes it to the pipe.

6: Catches errors. Broken pipe errors occur when the process tries to write to a pipe that no process has open for reading.

6 marks

e) What is the expected output of the following program?

```
require 'process_message_system'

coordinator = MessageProc.fork do
  resource_free = true
  loop do
    receive(
      message(:request, lambda { resource_free }) do |pid|
        resource_free = false
        pid.give(:reply)
      end,
      message(:release) do
        resource_free = true
      end
    )
  end
end
```

Name:
Login (UPI):

```
for name in 1..3
  MessageProc.fork(name) do |name|
    coordinator.give(:request, Process.pid)
    receive(
      message(:reply) do
        puts "#{name} has access to the resource"
      end
    )
    coordinator.give(:release)
  end
end
```

1 has access to the resource

2 has access to the resource

3 has access to the resource

2 marks

f) The output to the program in e) will not always be exactly the same. Explain how it might be different and why.

The order of output could be different because there is no control over which processes sends its request to the coordinator first.

2 marks

Name:
Login (UPI):

Overflow space for answers.

Name:
Login (UPI):

Overflow space for answers.

This page may be used for working.