

Concurrency and processes Answer

1. Consider the following code, with the assumption that each executes in a separate thread running concurrently:
 - a. Is it guaranteed that someone wins? Explain
 - b. Is it possible for someone to win? Explain

```
i=0
while (i<10)
  i++;
printf("a wins");
```

```
i=0
while (i>-10)
  i--;
printf("b wins");
```

No. If the left thread increments, then the right thread decrements before the left does a test, then the left thread increments before the right does a test, etc. they could keep going forever. Also if the instructions that make up the increment and decrement interleave, the same thing could happen.

Yes. If one has a chance to get through the loop enough times while the other isn't running (e.g. its on a faster processor or a multiprocessor system, or it finishes during one time slice while the other is waiting on the ready queue), it would win.

2. In a given operating system its scheduler may perform a context switch 100-1000 times a second. What is a context switch, and what does the OS have to do during one?
 - a. Why is a context switch performed so often?
 - b. Why isn't a context switch performed more often?

A context switch is the stopping of one process and the starting of another. The OS must save the registers (PC, SP, and general-purpose) and any other state information in the process' PCB.

- a. *To provide adequate response time for interactive users*
- b. *Because a context switch takes time, and if that time is high in proportion to the time slice too much CPU time will be spend on context switches.*

3. Consider the following pointless loop written in BASIC that counts upwards indefinitely

```
10 i = i + 1
20 GOTO 10
```

What would happen in this case

It would freeze the system while the loop is running because it is in a 'tight loop', which is a loop of code that executes without releasing any resources to other programs or the OS.

4. Describe 3 non-computer activities that could cause deadlock.
 - a. If a computer is in a deadlock state how would it feel to the user? How would you know if it was deadlock and not just a program running a never-ending loop?
 - b. We can have deadlock detection methods and also deadlock avoidance mechanisms. How would a deadlock avoidance mechanism run?
 - c. What are the advantages of a deadlock avoidance mechanism?
 - d. What are the disadvantages of a deadlock avoidance mechanism?

Non-computer activities, roads (traffic), negotiations e.g. peace negotiations where one side wants the other to do something good before they will do good and the other wants to do the same, road blocks, etc

- a. It was hang around doing nothing, perhaps it would tell the user. Typically feedback depends on the system running. A DBMS will tell the user its experienced deadlock a homemade program might not. You might use Top or some task manager to see what's happening – if there's high CPU then it's the never-ending loop, otherwise it could be deadlock.
- b. Deadlock algorithms such as the Banker's algorithm considers each request for resources and determines whether or not that request will lead to a safe state. If so, the request is granted; if not, it's postponed. No need to terminate a process or preempt resources and roll back the state (that is if there was an update of a value then that value must return to its original state = rollback).
- c. Advantage is that it prevents the system freezing rather than letting it freeze and then doing something about it.
- d. Huge overheads in performing the algorithm for every resource request, as it needs to keep the graph and update it for every request.

Memory Management

5. We are have a two dimensional array, *A*

var A: array[1..200,1..200] of character;

The contents of the array are stored in memory in row-major order starting at memory address 200 (i.e. A[1,1] is at address 200, A[1,2] is at address 201, ... A[1,100] is at address 299, A[2,1] is at address 300, etc.)

In our computer each page of memory consists of 400 bytes (1 character takes up 1 byte), and there are three physical frames available to a process, one of which is taken up by the code segment currently being run. Assuming only the code segment page is initially loaded into memory, and an LRU page replacement algorithm is being used, determine how many page faults are caused by the array accesses in the following two code segments:

```
for j := 1 to 200 do
  for i := 1 to 200 do
    A[i,j] := 'x'
```

*The page containing the code segment will be repeatedly accessed, i.e. will never be the selected victim of LRU. Observe that each page contains 400 bytes of data, i.e. page 1 contains A[1,1] through A[2,200] page 2 contains A[3,1] through A[4,200] etc. A page fault will occur on every second step of the inner loop, i.e. there will be $100*100 = 10,000$ page faults in the execution of this code segment.*

```
for i := 1 to 200 do
  for j := 1 to 200 do
    A[i,j] := 'x'
```

For this code segment, a page fault will occur on every second step of the OUTER loop, thus only 100 page faults will occur

*Understanding the system you are working on can allow you to make intelligent coding choices, which can in turn significantly affect the efficiency of your programs.*¹

¹ Final question mostly taken from Copyright © 1997, David M. Wessels.