McGill

# Midterm Exam: ECSE 427/COMP 310 – Operating Systems

Thursday, March 1, 2012
**Place:** ENGMC 304 & ENGTR 0070; **Time:** 10:30am – 11:30am

**EXAMINER:** Prof. M. Maheswaran

**STUDENT NAME:** _____ **McGill ID:** _____

**INSTRUCTIONS**:

- This is a **CLOSED BOOK** examination.
- Answer **DIRECTLY** on exam paper.
- No **CRIB SHEETS** permitted.
- You are permitted **TRANSLATION** dictionaries **ONLY**.
- **FACULTY STANDARD CALCULATOR** permitted **ONLY**.
- **THIS EXAMINATION PAPER MUST BE RETURNED.**

| QUESTION | POINTS |
|---|---|
| Question 1 | /20 |
| Question 2 | /20 |
| Question 3 | /20 |
| Question 4 | /20 |
| Question 5 | /20 |

[1] When **`fork()`** is executed by a process in UNIX, a child process is created. The child process is a clone of the parent in most aspects because it gets a copy of all key data structures. One of them is the file descriptor table. The file descriptor table points to a table containing context information about the file (we refer to this here as File Systems Table). The **`fork()`** does not copy this table, why? Explain what kind of operations would break if **`fork()`** makes a copy of this table.

The File Systems Table maintains two key pieces of information: file offset and reference count. The file offset gives the current location of access in the file (in some operating systems reads and writes have separate pointers). The reference count maintains the number of file descriptor entries that are pointing to a particular File Systems Table entry. If fork() copies this table, parent process' output will be overwritten by the child process' output or vice-versa. That is we cannot get the "append" behavior we expect.

The Data and BSS segments contain data. Why do we have two different segments for data? What are the disadvantages of merging them into one segment for the sake of simplification?

The BSS is for uninitialized data whereas the Data segment is for initialized data. By having two different segments, different strategies can be used to compress the on-disk image of the segments. For instance, if BSS includes a global uninitialized array, the BSS size would not be as big as the array. When the BSS segment is loaded, the loader will setup the memory map accordingly.

If Data and BSS are combined, the file image will be larger because uninitialized data will be taking up space. Program startup times will increase and disk space consumption will be higher.

What are the advantages of dynamic libraries over static libraries?

With dynamic libraries, the objects (shared objects) are loaded only when needed. The executable file is incrementally loaded as the execution proceeds. Portions of the executable (that are stored in user created libraries) or system library code that are not needed by a particular run are not loaded.

Also, because the shared objects are loaded at run-time, with dynamic libraries it is possible to interpose another version of the library that wraps a preexisting library (function call signature has to be same). By securely exploiting this feature, at run time, program behavior can be changed.

[2] What is the difference between an interrupt and a trap?

An interrupt is caused by an external event. A trap is generated by the program activity. For example, floating point overflow caused by divide by zero can trigger the "floating point exception" trap.

What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?

The advantage of implementing the threads in user space is that it is efficient. No context switch associated with thread switch. If important functionality such as networking can be carried out by user-level libraries, then user space threads can give a significant performance boost. However, most OSes by default provide important functionality through system call interfaces.

The disadvantage is that a system call that blocks a thread ends up blocking the whole application unless special care is taken to create a different user-level wrapper that calls the underlying system call in a non blocking manner and implements call waiting inside the threading library at the user level.

A multi-threaded process forks and creates a child process. Soon after creating the child process, the parent and child processes run into a deadlock. Explain how this could have happened? If such a deadlock cannot happen, explain why it cannot.

Consider a hypothetical scan-print application. It has two threads one to access the scanner to scan a document and another to access the printer. A process to get the full workflow (scan and print), needs to hold the scanner and also hold the printer. Suppose we fork a child process to multi-task (that is to keep the scanner busy while the printer is printing out the first document). The parent and child could run into a deadlock if the parent holds the scanner (remember the scanner and printer are controlled by two different threads) and the child holds the printer.

This deadlock scenario happens when the thread forking model allows the same number of threads as the parent in the child. If the child has only thread no matter how many threads the parent has, then this deadlock would not happen.

Some shell commands are run in separate processes and other are run inside the shell's process. Explain the distinction between the two types of commands that require different implementation strategies.

Some shell commands such as "cd" change the state of the shell. In particular, the cd command changed the current working directory. Any command that interacts with the file system will be impacted by the change made by cd. If cd were to run in a separate process, the change made by running the command would not change the state of the shell. That is subsequent commands would not be impacted by cd.

[3] There is a memory block of 128 KB that is going to be allocated by an OS for requests that arrive at run-time. Suppose the request sequence is 17KB, 9KB, 5KB, and 3KB and the whole block is free at the beginning of the sequence. Show the allocation if the binary buddy algorithm is used for the allocation process. In your diagram show where the memory is allocated and how much memory is used from each "block" that is allocated. For the whole sequence, find the memory wasted due to internal fragmentation (total amount of memory left unused inside each block). Suppose the buddy algorithm used the Fibonacci sequence to break the block, show the allocations and compute the internal fragmentation. The Fibonacci sequence is given by $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$.

With binary buddy, the following allocation will take place.

128 -> [64, 64] -> [[32 (allocated to 17), 32], 64]
- ➔ [[32 (allocated to 17), [16 (allocated to 9), 16]], 64]
- ➔ [[32 (allocated to 17), [16 (allocated to 9), [8 (allocated to 5), 8]]], 64]
- ➔ [[32 (allocated to 17), [16 (allocated to 9), [8 (allocated to 5), [4 (allocated to 3), 4]]]], 64]

Total internal fragmentation = (32 – 17) + (16 – 9) + (8 – 5) + (4 -3)
= 26KB

With Fibonacci series, the following sizes are available: 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 (bigger than 128).

So we can split 128 KB as follows using Fibonacci numbers

128 -> [89, 39] – lets ignore the 39 for now. Fibonacci can be used to split it as well. Either way, it does not change the results for internal fragmentation.

89 -> [55, 34] -> [55, [21 (allocated to 17), 13]]
- ➔ [55, [21 (allocated to 17), 13 (allocated to 9)]]
- ➔ [[34, 21], [21 (allocated to 17), 13 (allocated to 9)]]
- ➔ [[34, [13, 8]], [21 (allocated to 17), 13 (allocated to 9)]]
- ➔ [[34, [13, [5 (allocated to 5), 3 (allocated to 3)]]], [21 (allocated to 17), 13 (allocated to 9)]]

Total internal fragmentation = (21 – 17) + (13 – 9) + 0 + 0

[4] You need to develop a monitor-based solution for a ***student consultation office problem***. In this problem, students visit their student advisor to get advise. The advisor has grouped the students in two groups. There are n chairs for the students to wait in the office, if the advisor is busy. If the advisor is not busy, an arriving student starts getting the advise immediately. If the advisor is busy, student takes a chair if one is free. If all chairs are taken, student leaves without advise. Also, if a student belonging to one group is taking advise, the advisor does not want students belonging to the other group in the office. The students belonging to the other group can come into the office only after all of the students belonging to the current group have left the office. Develop a solution based on monitors. Identify the most important problem you can face with the solution you propose. Suggest an approach for solving it (No need to implement it – just describe the approach in words).

```
Monitor Student_Consultation {

   int groupType;
   int advisorState = {BUSY, DONE, FREE}
   condvar advisorFree;
   List chairs;

   int requestAdvisor(int mygroup) {
         if (advisorState = FREE) {
               advisorState = BUSY
               groupType = mygroup;
               return TRUE;
         }
         if (!((groupType == mygroup) && ((mychair = chairs.getAChair()) !=
NULL)))
               return FALSE;
         while (mychair != chairs.TopChair())
               advisorFree.wait();
         advisorState = BUSY;
```

```
    }
    releaseAdvisor() {
            if (chairs.empty())
                    advisorState = FREE;
            else {
                    advisorState = DONE;
                    advisorFree.broadcast();
    }

    init () {
            chairs.init(n);
            advisorState = FREE;
    }


}

// Assume a List structure that does not own its objects.
// TopChair would always be distinct. It does not depend on the slot - it
// depends on the actual object
```

[5] What are the necessary and sufficient conditions for deadlocks to occur?

(a) Hold and wait
(b) Circular wait
(c) Mutual exclusion
(d) No preemption

Other equivalent statements of the above four conditions are acceptable answers.

Which of these conditions are easy to prevent in a system where deadlocks should not occur by design? Briefly explain your answer.

Circular wait is the easiest to break by design. We could create an ordering which should be respected when seeking more than one resource. This way deadlocks can be easily prevented.

Hold and wait is another condition that can be prevented as well. However, this means the process needs to retry for the resource, which it once held. Implementing this strategy can increase the complexity of the program.

Mutual exclusion is a correctness issue. Some activities cannot be performed by more than one process in a simultaneous manner. For such activities mutual exclusion is essential.

There is a computer system with 250KB of main memory. A Bankers algorithm is in place to avoid deadlocks from happening as the memory allocation proceeds. Suppose there are four processes P1, P2, P3, P4 already admitted with the following amount of memory allocated to them: 90KB, 30KB, 50KB, and 20KB. What is the maximum value of total claim the four processes can have if the allocation is in safe (not very safe) state?

Total maximum claim is 830KB.

Total allocation at this point is 190KB. Memory available is 60KB.

Allocating this memory to P1 is the best option because it frees the most amount of memory when P1 completes. So the best sequence is P1 – P3 – P2 – P4.

P1 claim = 60 + 90 = 150KB
P3 claim = 150 + 50 = 200KB
P2 claim = 200 + 30 = 230KB
P4 claim = 230 + 20 = 250KB

Total claim = 830KB