Tony Maldonado
CS 474
September 30, 2020

Homework #1

Questions from Chapter 1

1. What are the three main purposes of an operating system?
    - To execute user programs and make solving user problems easier, to make the computer system convenient to use, and to use the computer hardware in an efficient manner.

2. Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?
    - The difference between symmetric and asymmetric multiprocessing is that in symmetric, all processors can perform the same tasks – they're treated as equals. But in asymmetric, there is one master processor and the others are slaves to the master, that get assigned tasks by the master.
    - Advantages to multiprocessors include saving money, they're quicker in program execution, and are more reliable. One disadvantage is that they are more complex than uniprocessor systems.

3. What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?

    - The purpose of interrupts is to cause change-of-flows within the CPU. With interrupts, a handler is called to deal with the interrupt, with a trap, it's a software-generated interrupt. A trap can be used by the user to signal a completed I/O service. A trap can also be used to call the OS routines or to find ALU errors.

Questions from Chapter 2

1. What is the purpose of system calls?

    - System calls provide the OS services to the user program through an API.

2. Describe three general methods for passing parameters to the operating system.

    - One way to do parameter passing is to pass them in registers, another is to place the parameters in a table and pass the address of the table as a parameter in a register, and another is to push onto the stack by the program and pop off the stack by the OS.

Tony Maldonado
CS 474
September 30, 2020

3. What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

   - Benefits to the microkernel approach is that it is easier to extend a microkernel and easier to port the OS to new architectures, on top of that is that it is more reliable and secure. User programs and system services interact by using message passing. A disadvantage is the performance overhead of user space to kernel space communication

Questions from Chapter 3

1. Describe the differences among short-term, medium-term, and long-term scheduling.

   - Short term scheduling is when the CPU scheduler selects processes in the ready state in memory and allocates CPU time to it. Medium-term is when the memory manager selects processes that are ready or blocked in the queue and removes them, planning to reinstate them later to run again. Long-term scheduling is when the job scheduler selects which processes should be brought into the ready queue.

2. Describe the actions taken by a kernel to context-switch between processes.

   - First, the OS saves the state of the current process, then invokes a service interrupt, then selects the new user process to run, and finally it restores the state of the new process and restarts the CPU.

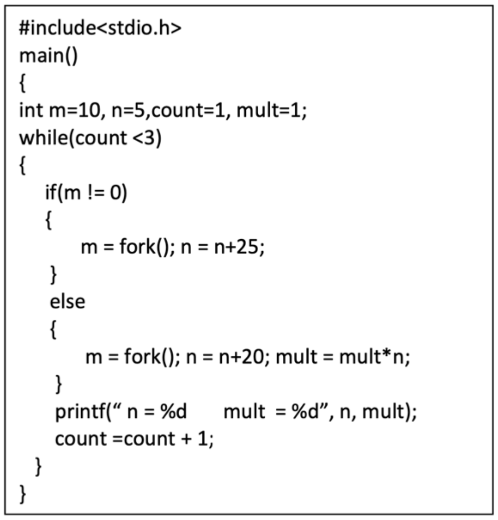
3. Including the initial parent process, how many processes are created by the program as shown:

```
#include<stdio.h> #include<unistd.h> int main() {

int i;
for (i=0; i<4; i++)

fork();

return 0; }
```

   - 2^4 = 16 processes

Tony Maldonado
CS 474
September 30, 2020

4. Assume that the following program contains no syntax errors. As it executes it will create one or more processes. Simulate the execution of this program and show how processes are created.



```
#include<stdio.h>
main()
{
int m=10, n=5,count=1, mult=1;
while(count <3)
{
    if(m != 0)
    {
         m = fork(); n = n+25;
     }
     else
     {
          m = fork(); n = n+20; mult = mult*n;
      }
      printf(" n = %d      mult  = %d", n, mult);
      count =count + 1;
   }
}
```

- The processes are created for every fork call, until the count value becomes 3. So, when m is not 0 a new process is created, and when m = 0 then a new process is created and the n value is incremented to 50.
- So, the total number of processes created is 6:
  - Process 1: when m != 0 then n = 30, mul = 1
  - Process 2: when m = 0 then n = 55, mul = 1
  - Process 3: when m != 0 then n = 30, mul = 1
  - Process 4: when m = 0 then n = 55, mul = 1
  - Process 5: when m = 0 then n = 50, mul = 50
  - Process 6: when m = 0 then n = 50, mul = 50
  -
- Output:
  - n = 30 mult = 1
  - n = 55 mult = 1
  - n = 30 mult = 1
  - n = 55 mult = 1
  - n = 50 mult = 50
  - n = 50 mult = 50

Tony Maldonado
CS 474
September 30, 2020

Questions from Chapter 4

1. Under what circumstances does a multithreaded solution using multiple kernel threads provide better performance than a single-threaded solution on a single-processor system?

   - In times when a program has a lot of page faults, a multithreaded kernel thread would be a lot better than a single-threaded solution in a single-processor system. This is because a single threaded solution can't just switch to another thread in the case of a fault.

2. Which of the following components of program state are shared across threads in a multithreaded process?

   a. Register values
   b. Heap memory
   c. Global variables
   d. Stack memory

   - They all share the Heap memory and global variables.

3. Consider a multiprocessor system and a multithreaded program written using the many-to-many threading model. Let the number of user-level threads in the program be greater than the number of processors in the system. Discuss the performance implications of the following scenarios.

   a. The number of kernel threads allocated to the program is less than the number of processors.

      - Some processors wouldn't be used because the scheduler maps threads to processors.

   b. The number of kernel threads allocated to the program is equal to the number of processors.

      - All processors could possibly be used simultaneously except in the case of a page fault, where the corresponding processor would be in an idle stage.

   c. The number of kernel threads allocated to the program is greater than the number of processors but less than the number of user- level threads.

      - When a thread faults, another would be swapped in if it's in a ready state.