SCHOOL OF COMPUTING, QUEEN'S UNIVERSITY

CISC324: Operating Systems

Assignment 1

due Tuesday, Jan 21 at the 4:30 lecture

January 2, 2020

Student Full Name: Ryan Protheroe

Student Number: 20069587

Group Number: 5

1 Computer Architecture Questions

1. Consider a computer system with a 32-bit address bus and 8-bit data bus. Then: (i)How many memory locations can be addressed through the address bus? (ii) How many bits are per address location? and (iii) What is the maximum size of the main memory that can be used?

    i. 2^{32}=4,294,967,296 memory locations are addressable

    ii. Determined by data bus, 8-bits per address location

    iii. (2^{32})*8 = 4GB is the maximum size of the main memory that can be used

2. The MDR (Memory Data Register) and the MAR (Memory Address Register) are two important CPU internal registers. Explain when a CPU uses these two registers?

    The CPU uses these two registers when as part of the instruction execution cycle. The MAR holds the address of a memory location and the MDR stores the data at that memory location to be transferred to and from the immediate access storage, they work in tandem.

3. Consider the following x86-based assembly code:

```
[0x153F]    Mov Ax, 5
[0x1540]    Mov Bx, 3
[0x1541]    Mul Ax, Bx
[0x1542]    Xor Ax, Ax
[0x1543]    Halt
```

   a) If the CIR (Current Instruction Register) contains the instruction stored in memory address 0x1540, then what is the value of the IP (Instruction pointer) and Ax register?

      Assuming that the decoder is not finished/ cycle only up to the point of CIR
      IP: 0x1540
      AX: 5

   b) If the decoder unit has just finished decoding the instruction stored in memory ad-dress [0x1542] then what is the value of the IP (Instruction pointer), Ax register, and Bx register?

      IP: 0x1543
      AX: 15
      BX: 3

4. Write a C-program (some instruction statements) for the following **two** x86-based assembly codes (*Hint: Think about while-loops and do-while-loops*):

```
[0x153F]    Mov Ax, 1       [0xB53F] Jmp X
[0x1540]    X Nop           [0xB540] Y Nop
[0x1541]    Cmp Ax, 1       [0xB541] X Cmp Ax, 1
[0x1542]    Je X            [0xB542] Je Y
[0x1543]    Halt            [0xB543] Halt
```

```
#include <stdio.h>
int main()
{
        int x = 1;
        do{
                if(x==1){
                goto X;
                }
        }while(x == 1);
        X: goto Y;
        Y: goto X;
}
```

5. Can a uniprocessor computer run multiple programs in fake concurrency? Explain.

   Yes, it is stressing the word "fake" though. The multiple processes do not execute at the same time, the multiple context switches create the illusion of concurrency when in reality one process is being executed at a time.

6. Give an example of a multi-tasking and single-tasking operating system?

   Multi-tasking: Windows 10
   Single-tasking: Palm OS

7. What is a system call? Give three examples and explain their function.

   A system call is an interface between the user (programs) and the services that are made available by an operating system. It is a programmatic way a computer program can request a service from the operating system.

   i. Kill(): This is a process control system call, used to kill a process
   ii. Open(): This is a file management system call, used to open a file
   iii. Release(): This is a device manipulation system call, used to release a device

2 Processes, Interrupts, and Context Switching

1. Give three reasons that may lead a program to be suspended.

    i Swapped out of RAM to free some space for other process

    ii Swapped out of RAM when debugging the main program

    iii Swapped out by the parent process.

2. The operating system uses a data structure to represent a process and keep track of it. What is that structure? Give three information fields that are kept in that data structure. Also, explain the content of the memory management information field and the scheduling information field (do not cite these two fields among the three fields that you are asked to cite).

    The structure used to represent each process is the process control block (PCB). Three examples of information fields kept in this data structure are: process state, program counter and CPU registers. The memory management field stores information of the page tables, base and limit registers, and segment tables depending on memory used by the operating system. The scheduling information field includes a process priority, pointers for scheduling queues, and various other scheduling parameters.

3. How does the operating system locate that data structure for a given process?

    Each PCB is stored in another data structure, the process table, to which the operating system uses to acquire the PCB for the given process.

4. What is the Zombie state of a process in Unix-based systems? How can you simulate that in a C-program?

    The zombie state of a process on a Unix-based system is when a process has completed execution but still has an entry in the process table. This can be simulated in a C-program by first calling fork(), sleeping the parent, then exiting the child process.

5. What is an orphan process in UNIX-like operating systems?

    An orphan process is a process whose parent process no longer exists.

6. What is context switching? and why is that needed?

   Context switching refers to the process of storing the state of a process so that it can be restored and executed from the same point at a later time. It is important for time-sharing systems as it:

   i. Increases the CPU throughput
   ii. Increases the CPU response time
   iii. Prevents certain processes from hogging the CPU infinitely
   iv. Allows high-priority processes to be executed
   v. Reduces process turn-around-time

7. What is an interrupt? and what is the relation with context switching?

   An interrupt is an event that alters the sequence of instructions executed by the CPU. A dedicated program called the Interrupt Service Routine (ISR) is executed as a consequence. The context switching occurs when the ISR starts execution as it has top priority to handle the current interrupt. The process that was interrupted has its state stored so it can be restored and resumed after the interrupt.

8. Can we have an interrupt without context switching? If yes given an example.

   Yes, an interrupt can occur without context switching. An example of this would be a keyboard interrupt.

9. Explain what does the *fork*() function call perform in a C-program running on a UNIX-like system? What will happen when a process executes that function (in the memory)?

   The *fork*() function call creates a child process (another process). The OS duplicates the PCB, allocates a new memory space and updates the PCB data fields.

10. What will happen when the ALU (Arithmetic and Logical Unit) starts executing the instruction at the address 0x330F in the following program?

```
[0x330B]    Mov Ax, 0x000A
[0x330C]    Add Ax, 0x0002
[0x330D]    Mul Ax, 0x000A
[0x330E]    Xor Ax, Ax
[0x330F]    Div Ax
[0x3310]    Halt
```

The preceding instruction, [0x330E], results in Ax to be equal to 0. The instruction [0x300F] is Div 0, and we know division by zero is not possible, so the CPU traps to an exception handler.

11. Consider the following x86-based assembly code that is currently executing on a computer that has a memory which is Byte-addressable and a CPU that has a word size of 2-Bytes:

```
Mov Ax, [0X3F55]      (encoded in 2 words)
Mov Bx, [0X3F56]      (encoded in 2 words)
Mul Cx, 0x0000 (encoded in 1 word)
Xor Bx, Cx     (encoded in 1 word)
Inc Ax   (encoded in 1 words)
Halt     (encoded in 1 words)
```

   i. Assuming that the address of the first instruction in this program is 0xFBC0 and that an interrupt occurs after the completion of the 4th-instruction i.e., Xor Bx, Cx, what will be the value of the IP register that will be pushed onto the stack?

      Since the IP increments during the execution of an instruction, the IP should have the address of the proceeding instruction. So the value will be 0xFBC6.

   ii. Which component of the computer system is responsible for pushing the PC value into the stack? why?

      The CPU (hardware) pushes the current PC value onto the stack. This is because an interrupt alters the sequence of instructions executed by the CPU, so it must be the one to arrange a call to the interrupt handler, which requires a certain CPU state.

   iii. Which stack are we referring to?

      The process stack.

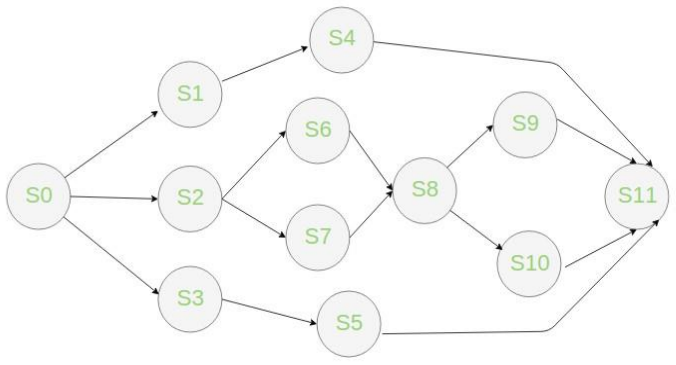
12. Explain why polling is not a good mechanism for managing hardware events.

    Polling is not a good mechanism for managing hardware events because it relies on the CPU to ask if any events have occurred. This takes up CPU cycles and effectively impacts performance as well as drives up power consumption. Interrupts work in the opposite way, the I/O device requests the CPU/ says there is an event instead of the CPU asking, it makes more sense.

13. What is the difference between software-based interrupts and hardware-based interrupts? Give one example for each type of interrupts.

    Software-based interrupts occur as a result of an instruction in a program code. Hardware-based interrupts are triggered by a hardware unit such as the keyboard or mouse to get the attention from the OS. An example of a software-based interrupt is the fork() function on a UNIX-like system, it makes a system call to create a new process. An example of a hardware-based interrupt is the hard disk signalling that it has read a series of data blocks.

14. Using (Begin-End) for sequential executions and (ParBegin-ParEnd) for parallel executions, give the pseudo-code for the following PPG (ProcessPrecedenceGraph), where S_0 is the first instruction statement:



```
Begin
   S0
   ParBegin
      Begin
      S1
      S4
      End
      Begin
      S2
         ParBegin
         S6
         S7
         ParEnd
      S8
         ParBegin
         S9
         S10
         ParEnd
      End
      Begin
      S3
      S5
      End
   ParEnd
   S11
End
```

15. If the program (which you just constructed above) runs on single-core CPU and a timesharing operating system then: Do the blocs that are supposed to be executed in parallel execute in real concurrency? Explain.

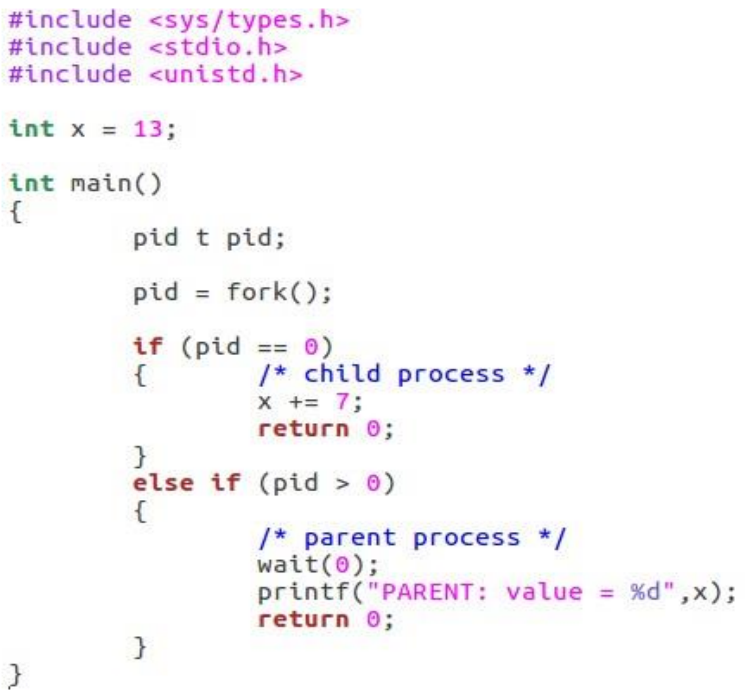
    Similar to question 5 of part 1, yes, the blocs are executed in real concurrency on a single-core CPU. Interleaving of processes is used to create the concurrency, where tasks can start, run and complete in overlapping time periods. Essentially, the tasks are split up into small steps, and the CPU switches back and forth between the processes to complete the desired tasks, therefore concurrently. As well, they cannot be executed in parallel on a single, non-hyperthreaded core CPU.

16. Does the PC increments (updates its value) after or before the current instruction is decoded? Provide two arguments to support your answer.

    The PC increments after the current instruction is decoded. This can be deduced from lecture 2 as part of the Instruction Execution Cycle slides, 2.1 states that the CU decodes and transforms the instruction, and 2.3 states the PC is incremented, therefore after the decode. Similarly in the example given a few slides under, where it showcases the process, we can clearly see that the PC is incremented after the instruction is decoded. (Googling leads me to believe that it actually occurs before the instruction is decoded, however the slides say differently so I'm not sure what to put here…)

17. In the following program, a parent process executes the primitive fork() to create a child process. Initially, the variable *x* has the value 13, the child process code snippet has been implemented in such a way so that it increments the value of *x* by 7, and the parent process displays the value of *x* after its child process terminates. Explain the value displayed by the parent process

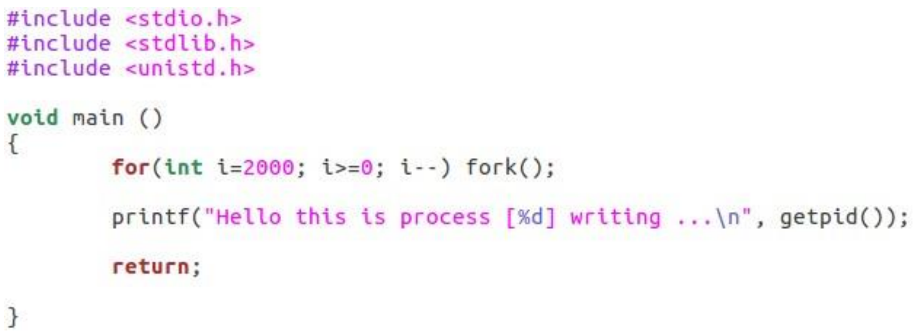
    The child process is completely independent of the parent and there is no exchange of information from the child to the parent to get an incremented x value. So, the output remains as 13 until the result of the child process is fetched.



```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int x = 13;

int main()
{
        pid t pid;

        pid = fork();

        if (pid == 0)
        {       /* child process */
                x += 7;
                return 0;
        }
        else if (pid > 0)
        {
                /* parent process */
                wait(0);
                printf("PARENT: value = %d",x);
                return 0;
        }
}
```

18. In the following program, give the final number of processes that display the printf message (**Caution**: do not execute that program as is. It may crash your computer).

    The final number of processes for this loop can be expressed as 2^n (this includes the parent), where n represents the number of loops. So, in this case it would be 2^{2000} which is essentially infinite, at least according to the google calculator.



```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void main ()
{
        for(int i=2000; i>=0; i--) fork();

        printf("Hello this is process [%d] writing ...\n", getpid());

        return;

}
```