Operating Systems

Tutorial Worksheet 02

Description. This tutorial worksheet covers basics in CISC-based computer architecture and operating systems' process management.

1. **Basic CISC x86 Assembly and Instruction Execution Cycle**

   Exercise 1

   Discuss the following questions:

   a) How many logical cores: 4 CPU sockets, 4 cores per socket, and 2 threads per core?

   b) How many physical cores: 2 CPU sockets, 4 cores per socket, and 2 threads per core?

   c) Which feature of the CPU has direct impact on the maximum addressable space?

   d) Can a uni-processor computer run multiple programs in fake concurrency? Explain.

   Exercise 2

   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 PC (Program Counter) and Ax register?

   b) If the decoder unit (which is part of the CPU's control unit) has just finished decoding the instruction stored in memory address `[0x1542]` then what is the value of the PC (Program Counter), Ax register, and Bx register?

   c) Does the PC increments after or before the current instruction is decoded? Provide two arguments to support your answer. Does it always get incremented?

Exercise 3

Write a C instruction statements reflecting the following **two** x86-based assembly codes:

```
[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
```

Exercise 4

What will happen when the ALU (Arithmetic and Logic 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
```

Exercise 5

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)
```

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

b) Which component of the computer system is responsible for pushing the PC value onto the stack? why?

c) Which stack are we referring to?

Exercise 6

The following code snippet is an x86-assembly code to be run on an Intel 8086 microprocessor. Without going deep into the details, analyze the code and infer what this code can be used for. The instruction `int` creates a software interrupt, in this case, a system call. The call is made for a function, which code is placed on the `AH` part of the the `AX` register.

```
section .data ;Data segment
    msg db 'The National School of Artificial Intelligence!'

section .text   ;Code segment
    mov ah, 09h         ; Here we are indicating a function code (09h)
    mov dx, OFFSET msg  ; Grab the address of variable msg
    int 21h             ; Perform system call (user wants function 09h)

    hlt
```

2. Process Management

Exercise 1

Discuss the following questions:

a) What is a process? and what's the relationship with a program?

b) What is a PID, and how can you use it to terminate the execution of a program?

c) The operating system uses a data structure to represent each process in the system and keep track of it. What is that structure called? Give five information fields that are kept in that data structure.

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

e) Explain what does the `fork()` system call perform in a C-program running on a Unix-like system? Describe what happens when a process executes that function.

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

g) What is a `Zombie` state in Unix-like systems? How can you simulate that? (Lab)

h) What is an `orphan` process in Unix-like operating systems?

i) What is a context switching? and why is that needed?

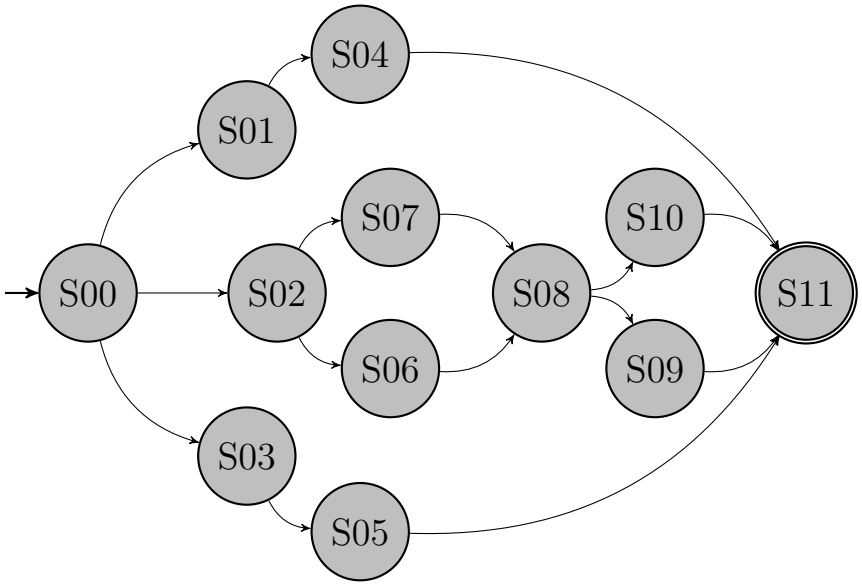
j) What is an interrupt? and what is the relation with context switching?

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

l) What is the difference between software-based interrupts and hardware-based interrupts? Give three examples for each type of interrupt.

Exercise 2

Using (`Begin-End`) for sequential executions and (`ParBegin-ParEnd`) for parallel executions, give the pseudo-code for the following PPG (Process Precedence Graph), where S_0 is the first instruction statement:



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

Exercise 3

In the following two C-programs, give the final number of processes that display the `printf` message ("Hello this is process ...").

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

int main()
{
    for(int i=4000; i>=0; i--) fork();

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

    return;
}

```

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

void main()
{
    for(int i=0; i<4; i++) fork();

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

    return;
}
```

Exercise 4

In the following program, a parent process executes the primitive `fork()` to create a child process. Initially, the variable x has the value 100, the child process code snippet has been implemented in such a way so that it increments the value of x by 100, and the parent process displays the value of x after its child process terminates. Explain the value displayed by the parent process.

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

int x = 100;

int main()
{
    pid_t pid;
    pid = fork();

    if(pid == 0)
    {
        /*child speaking*/
        x +=100;
        return 0;
    }
    else if (pid > 0)
    {
        /*parent speaking*/
        wait(0);
        printf("PARENT: Value = %d", x);
    }

}
```

Exercise 5

How many times the message "Hello Students" will be displayed by the following two process-forking programs? Explain.

```c
void main() {

    if(fork() && fork())
    {
        printf("Hello Students\n");
    }

    return;
}
```

```c
void main() {

    if(fork() || fork())
    {
        printf("Hello Students\n");
    }

    return;
}
```

Exercise 6

In the following program, a parent process executes the primitive `fork()` to create a child process. Initially, both variables x and y have the value 100, the child process code snippet has been implemented in such a way so that it increments the value of both x and y by 10, and the parent process displays the value of $x + y$ after its child process terminates. Explain the value displayed by the parent process.

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

int main()
{
    pid_t pid; int x = 100; int y = 800;
    pid = fork();

    if (pid < 0)
    {
        fprintf(stderr, "fork() system call failed"); return;
    }
    else if (pid == 0)
    {
        x = x + 10;
        execlp("/bin/ls","ls", NULL);
        y = y + 10;
        printf("[Child]: x + y = %d \n", x+y);
    }
    else
    {
        wait(0);
        x = x + 500;
        printf("[Parent]: x + y = %d \n", x + y);
    }
    return;
}
```