Operating Systems
CS4348

Project #1: Exploring Multiple Processes and IPC

Due Date: Saturday, September 30, 2023

I. Project Organization

You should do the following pieces to complete your project. Each piece is explained below:

• Code	60 points
• Output	30 points
• Summary	10 points

Each piece is separately graded. A missing piece will result in losing all of the points for that piece.

Code

The actual code of your program should be in this section.   It should be nicely formatted with plenty of comments. The code should be easy to read, properly indented, employ good naming standards, good structure, etc.

Output

Output will be graded by running your program on four sample programs posted on eLearning, plus one that you have written. The one you write should be different from the samples and at least as complex. Each is 6 points.

Summary

The summary section should discuss three things: (1) project purpose, (2) how the project was implemented, and (3) your personal experience in doing the project. It should be at least one page in length. A summary that is lacking will not receive full credit.

II. Project Description

Language/Platform

The project must be written in C, C++, or Java.
If using C or C++, you must use a Unix fork to create processes and a Unix pipe for communication.
If using Java, you must use the Runtime exec method to create processes and streams for communication.
Your project will receive no credit if not using processes or if using threads instead of processes.
All code must run successfully on our cs1.utdallas.edu server or csgrads1.utdallas.edu server.
Any other method requires instructor approval.

Problem Overview

The project will simulate a simple computer system consisting of a CPU and Memory.
The CPU and Memory will be simulated by separate processes that communicate.
Memory will contain one program that the CPU will execute and then the simulation will end.

Objectives

1) Learn how multiple processes can communicate and cooperate.
2) Understand low-level concepts important to an operating system.
   a. Processor interaction with main memory.
   b. Processor instruction behavior.
   c. Role of registers.
   d. Stack processing.
   e. Procedure calls.
   f. System calls.
   g. Interrupt handling.
   h. Memory protection.
   i. I/O.
   j. Virtualization/emulation

Problem Details

CPU
   It will have these registers: PC, SP, IR, AC, X, Y.
   It will support the instructions shown on the next page of this document.
   It will run the user program at address 0.
   Instructions are fetched into the IR from memory. The operand can be fetched into a local variable.
   Each instruction should be executed before the next instruction is fetched.
   The user stack resides at the end of user memory and grows down toward address 0.
   The system stack resides at the end of system memory and grows down toward address 0.
   There is no hardware enforcement of stack size.
   The program ends when the End instruction is executed. The 2 processes should end at that time.
   The user program cannot access system memory (exits with error message).

Memory
   It will consist of 2000 integer entries, 0-999 for the user program, 1000-1999 for system code.
   It will support two operations:
      read(address) - returns the value at the address
      write(address, data) - writes the data to the address
   Memory will read an input file containing a program into its array, before any CPU fetching begins.
   Note that the memory is simply storage; it has no real logic beyond reading and writing.

Timer
A timer will interrupt the processor after every X instructions, where X is a command-line parameter.
The timer is always counting, whether in user mode or kernel mode.

Interrupt processing
There are two forms of interrupts: the timer and a system call using the int instruction.
In both cases the CPU should enter kernel mode.
The stack pointer should be switched to the system stack.
The SP and PC registers (and only these registers) should be saved on the system stack by the CPU.
The handler may save additional registers.
A timer interrupt should cause execution at address 1000.
The int instruction should cause execution at address 1500.
The iret instruction returns from an interrupt.
Interrupts should be disabled during interrupt processing to avoid nested execution.
To make it easy, do not allow interrupts during system calls or vice versa.

Instruction set

1 = Load value	Load the value into the AC
2 = Load addr	Load the value at the address into the AC
3 = LoadInd addr	Load the value from the address found in the given address into the AC (for example, if LoadInd 500, and 500 contains 100, then load from 100).
4 = LoadIdxX addr	Load the value at (address+X) into the AC (for example, if LoadIdxX 500, and X contains 10, then load from 510).
5 = LoadIdxY addr	Load the value at (address+Y) into the AC
6 = LoadSpX	Load from (Sp+X) into the AC (if SP is 990, and X is 1, load from 991).
7 = Store addr	Store the value in the AC into the address
8 = Get	Gets a random int from 1 to 100 into the AC
9 = Put port	If port=1, writes AC as an int to the screen If port=2, writes AC as a char to the screen
10 = AddX	Add the value in X to the AC
11 = AddY	Add the value in Y to the AC
12 = SubX	Subtract the value in X from the AC
13 = SubY	Subtract the value in Y from the AC
14 = CopyToX	Copy the value in the AC to X
15 = CopyFromX	Copy the value in X to the AC
16 = CopyToY	Copy the value in the AC to Y
17 = CopyFromY	Copy the value in Y to the AC
18 = CopyToSp	Copy the value in AC to the SP
19 = CopyFromSp	Copy the value in SP to the AC
20 = Jump addr	Jump to the address
21 = JumpIfEqual addr	Jump to the address only if the value in the AC is zero
22 = JumpIfNotEqual addr	Jump to the address only if the value in the AC is not zero
23 = Call addr	Push return address onto stack, jump to the address
24 = Ret	Pop return address from the stack, jump to the address
25 = IncX	Increment the value in X
26 = DecX	Decrement the value in X
27 = Push	Push AC onto stack
28 = Pop	Pop from stack into AC
29 = Int	Perform system call
30 = IRet	Return from system call
50 = End	End execution

- Don't add to this list without approval

Input File Format

Each instruction is on a separate line, with its operand (if any) on the following line.
The instruction or operand may be followed by a comment which the loader will ignore.
Anything following an integer is a comment, whether or not it begins with //.
A line may be blank in which case the loader will skip it without advancing the load address.
A line may begin by a period followed by a number which causes the loader to change the load address.
Your program should run correctly with the any valid input files.

Sample Programs

The input program filename and timer interrupt value should be command line arguments, for example:

```
java Project1 program.txt 30
```

Here are two sample programs for illustration purposes:

This program gets 3 random integers and sums them, then prints the result.
Note that each line only has one number.

```
 8   // Get
14  // CopyToX
 8   // Get
16  // CopyToY
 8   // Get
10   // AddX
11   // AddY
 9   // Put 1
 1
50  // End
```

This program prints HI followed by a newline to the screen. To demonstrate a procedure call, the newline is printed by calling a procedure.

```
 1   // Load 72=H
72
 9   // Put 2
 2
 1   // Load 73=I
73
 9   // Put 2
 2
23  // Call 11
11
50  // End
 1   // Load 10=newline
10
 9   // Put 2
 2
24  // Return
```

IV. Project Guidelines

Submitting

Submit your project on eLearning. Include in your submission the following files:

1) A Word or text document for the summary.
2) Your source files.
3) The sample5.txt file you created.
4) A “readme” file listing your files, a description of each file, and how to compile and run your project.

Partial or Missing Submissions

It is your responsibility to upload all of the right files on time. It is recommended that you double-check the files you upload to make sure they are the right ones. Once the deadline passes, changes to the submission are not accepted without a late penalty.

Academic Honesty

This is an individual project. All work must be your own. Comparison software will be used to compare the work of all students. Similar work will be reported to the Office of Community Standards and Conduct for investigation.

Grading

The written portions will be graded subjectively based on completeness and quality. The code will be graded based on points allocated for each key part of the processing as determined by the instructor. The output will be graded based on expected results for the input programs.

Resources

Examples were given in class and are available on eLearning. Code from these examples may be freely used in your project. The web also has many good articles on this topic. You may also find information in books on Unix or Linux programming.