Gebze Technical University
Department Of Computer Engineering
CSE 312 / CSE 504

Operating Systems

Homework #02
Due Date: Apr 28th 2017
System Calls, Processes and Simple Virtual Memory

In this homework, we will use our 8080 Emulator with a more advanced OS that supports simple virtual memory. Our OS now can run more than one process using multitasking techniques that we saw in the lectures. Since we will keep more than one process in the memory, we will need a very simple virtual memory such as base and limit registers described in your textbook.

Each process has its own base and limit registers. When a process wants to access the memory, then the accessed address is added to the base register. If the accessed location is beyond the limit, then a segmentation error occurs.

We have added a new base class to our 8080 emulator called MemoryBase. You will derive a new memory class from this base class and implement two virtual access functions that supports base and limit registers. We also modified the other emulator files to make the base memory class work. Please carefully study the differences between the HW1 version and HW2 version of the emulator.

Our new OS will offer the following system calls in addition to the calls of HW1. This time all of your PRINT system calls will put an '\n' character at the end. Note that you should update your ASM files to include the definitions of the new system calls. Your logical address space is still 16K and your physical machine has 64K of memory. This means that you can run at most 4 processes at the same time.

Call	Params to pass	Function	How many cycles
FORK	Register A = 7	It works like the fork system call of UNIX systems. It returns the result in register B. On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, it returns 1 and no new process is created. PID numbers 0 and 1 are not a valid process numbers, they are used for returning results.	50
EXEC	Register A = 8 Register BC = address of the filename	It works very similar to UNIX execl systems call without the arguments. It loads the program specified in the filename and lets the scheduler execute the program. The file name is the null terminated string at address BC.	80
WAITPID	Register A = 9 Register B holds the PID of	Blocks the calling process until the process with PID is terminated.	80

	the process		

Since you have more than one process in the memory, you need to keep a process table, the process table will hold at least the following properties for each process

- The name of the process (filename)
- The saved register set of 8080 chip
- The base and limit registers
- The PID
- The parent PID
- Starting time of the process (the cycle number of the CPU)
- How many cycles the process has used so far
- The state of the process (ready, blocked, running)
- The physical address of the memory location of the process

Any other data structure can be added to the process table as needed.

When your simulated computer starts running, it will first read a program file and start running the program.

For this homework, you will do the following

- Update your OS class with the system calls
- Derive a new memory class from the MemoryBase class that can be used with the new 8080 CPU
- Write a robin robin scheduler (quantum time: 100 cycles) that keeps the process table updated and does the context switching
- Your new system should automatically choose memory location for the processes loaded.

You will write two ASM programs for your new system

- You will write a simple shell ASM program (shell.asm) similar to what the textbook provides at Figure 1-19. You will read commands from the keyboard, run the command (execute the ASM files) and repeat the same process.
- You will also write another ASM program (parallel.asm) that runs 3 processes at the same time. One process prints numbers from 0 to 50, the other process add numbers from 1 to 10 and prints the result and final process prints the numbers from 100 to 50.

You will provide the main simulation program as follows

- Write a simulation program that runs your systems with some command line parameters. There should be parameters for the program name and debug flag.
  - `sim8080 exe.com 1` : will read the program from exe.com. In debug mode 1, the status of the CPU will be printed to the screen after each instruction execution. At the end of the simulation, the whole memory will be saved to exe.mem as a text file of hexedecimal numbers. Each line of the file will start with the memory addres, then it will show 16 bytes of hexadecimal numbers separated with spaces.
  - In Debug mode 0, the program will be run and the contents of the memory will be saved as in Debug mode 1.
  - In Debug mode 2, every time a process switch happens, the information is printed to the screen. The information printed will be
    - the process names (both processes)
    - the total cycles spent for the blocked process
  - In Debug mode 3, information about each process is printed on the screen when a process switch occurs. This information will be very similar to "ps -ef" command in UNIX systems. It will include the process name and all of its process table entries

We will provide the submission instructions in a separate document. You should strictly follow these instructions otherwise your submission may not get graded.
You will submit the following files for this homework

- memory.h and memory.cpp: the MMU files to handle virtual memory processing
- main.cpp
- gtuos.cpp and gtuos.h
- shell.asm and parallel.asm as described above
- The sample ASM programs (print numbers, add numbers, etc.) that you write for HW1