Gebze Technical University
Department Of Computer Engineering
CSE 312 /CSE 504
Operating Systems
Spring 2019

Homework #02
Due Date: March 29th 2019
Interrupts, Processes, Multi-Programming

In the last assignment we flexed your assembly coding and multi-byte precision skills. Now we are going to dive into OS kernel itself. In this homework you are going to develop a kernel that will support multi-programming, interrupt handling and just a bit of memory management. Please refer to your course textbook for multi-programming and study your cookbook for how 8080 architecture handles interrupts.

We have updated the underlying emulator to support multi-programming with primitive instructions and hardware.

Please inspect the given emulator codes very carefully and understand the mechanism.

We have included Memory Base and Limit registers while accessing the memory. These registers will be explained in Chapter 3 but before that you will hear about them during the lectures. You remember what these registers do from your classes. The limit registers is currently non-operating but we will heavily use this registers in future assignments.

We have altered the behavior of instruction **PCHL.** In previous versions of the machine, execution of this instruction would update **PC** register with the contents of **HL.** In updated version it also performs an update on the base register with the contents of **DE** register. It also removes last two entries from stack and replaces the values in **DE and HL.** Table below shows an explanation about the change.

Instruction	**OLD OPERATION**	**NEW OPERATION**
PCHL	CPU.PC <== CPU.HL	MEM.BASEREG<== CPU.DE CPU.PC <== CPU.HL

Table-1

1. Your hardware now supports interrupts and clocks. Check the given cookbook to understand interrupt behavior of 8080 electronics.

2. Aside from the official behavior, our hardware is a very thoughtful machine. Whenever an interrupt occurs,
it copies all the state information (registers, sp, pc, PSW, base and limit_registers) to memory address starting from **256d** (Decimal)**, that is hardwired to the machinery.** We call this wiring **interrupt buffer**. So we can use this information to help it achieve much more than originally intended. Table 2 shows the contents of the addresses of the interrupt buffer.

3. As we mentioned before your operating system supports clocks now! In order to get a clock interrupt when your operating system starts you should immediately enable interrupts and your CPU will start counting. You can control your clock interrupts by the function **setQuantum** in **8080emu.cpp**. (Default is 80)

4. Whenever clock count reaches the quantum value, it will generate interrupt **RST 5.** Learn about this interrupt from your 8080 cookbook.

ADDRESS	**CONTENTS**
256d	A
257	B
258	C
259	D
260	E
261	H
262	L
263	SP.LOW
264	SP.HI
265	PC.LOW
266	PC.HI
267	BASEREG.LOW
268	BASEREG.HI
269	CPU.CC

Table-2

You task is to develop a part of the kernel called MicroKernel.asm that will perform interrupt handling, multi-programming and context switching. Be careful about hardware constraints (Interrupt Addresses). Beware the kernel location in memory! If your kernel or any part of your processes is located into an interrupt address, your system will fail.

What we expect from MicroKernel.asm:

1) Loading multiple programs into memory: Kernel will be able to load multiple programs in to memory by selecting appropriate locations. This operation will be a system call and will be detailed in Table 1.

2) Handling multi-programming: you need to develop a **Process Table** that will hold the necessary information about the processes in the memory.(You should study what **Process Tables** holds. You can read carefully throughout the chapter 2 of the course book or any other online resource).

3) Handling Interrupts: Our emulator will generate interrupts, and your kernel will handle and respond.

4) Perform Round Robin scheduling: Your kernel will inform the emulator with a virtual time quantum (cycle count) and emulator will generate a scheduling interrupt you will trap this interrupt, and perform a context switching by selecting an appropriate process from the memory.

5) Whenever a context scheduling occurs, you will print all the information about the processes in process table including but not limited to the entries in the list below.

   a. Process ID
   b. Process Name,
   c. Program counter
   d. Memory Base Register Content
   e. Memory Base Register Address
   f. Stack Pointer address

You will keep the previous system calls from the first homework and add the new ones below:

Call	**Params**	**Function**	**Cycle**
LOAD_EXEC	REGISTER A = 5 Register BC = address of the filename Register HL = Start address of the program	it loads the program specified in the filename starting from the start address and immedieatly causes scheduling interrupt to start scheduling.	100
PROCESS_EXIT	REGISTER A = 9	This will cause the Operating system to clear and update the resources of the finished process, process tables and return control to the scheduler.	80
SET_QUANTUM	REGISTER A =6 REGISTER B=QUANTUM_TIME	System call to change the QUANTUM_TIME of Round Robin Scheduler	7

Table-3

Emulator Life-Cycle

You will implement 3 different flavors of MicroKernel. Don't worry 90 percent of the code is same between the Micro Kernels. We further explain the details below.

When your kernel is loaded Your OS will enable the interrupts with appropriate instruction, set the virtual quantum timer (Clock Timer). It will start a process named **init** with **process id 0.** In different Micro Kernels Init process will load programs into memory differently

- In the first strategy First Strategy is **init** process will initialize Process Table, enable interrupts and load 3 different programs (listed below) to the memory start them and will enter an infinite loop until all the processes terminate.

- Second strategy is randomly choosing one of the programs and loads it into memory 10 times (Same program 10 different processes), start them and will enter an infinite loop until all the processes terminate.

- Final Strategy is choosing 2 out 3 programs randomly and loading each program 3 times start them and will enter an infinite loop until all the processes terminate.
- When Emulator starts, it immediately loads the **MicroKernel file** given by command line parameters example EX: **./emulator MikroKernel1.com 0.**

- For every RST **5 Interrupt,** in other words scheduling signal, **OS** should handle the interrupt and perform round robin scheduling.

- Instead of calling **HLT**, if a program finishes execution it will acknowledge its termination by calling **PROCESS_EXIT**.

- This call will transfer the control to the operating system and your operating system will perform necessary actions.

- Emulator will shut down only after all the programs in memory terminate.

Assembly Files.

You will supply us with three different Assembly Files.

- Sum.asm: which was delivered to you with first homework, Update the System call's. **Ex Output;** 20 : 210

- Primes.asm: you are going to find primes less than 256 and print them on to screen. **Ex Output;** 17: Prime

- Collatz.asm: You are going to find collatz sequence for each number less than 25. You can find information about (Collatz conjecture on internet). For each number you will show the number being interested in, and its collatz sequence and go to next number **Ex Output;** 7: 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Your homework template includes several code and sample files. Here is a description for them. Do not change Emulator Files and do not send these files with your homework. Study these files to understand the 8080 assembly code and the emulator.

8080emu.cpp : Emulator implementation for 8080. You will not change this file
8080emuCPP.h: Emulator interface for 8080. You will not change this file

gtuos.h : sample header for GTUOS. You will rewrite this file
gtuos.cpp : sample implementation for GTUOS. You will rewrite this file
main.cpp : sample main function for this HW. You will rewrite this file
MicroKernel1.asm,MicroKernel2.asm,MicroKernel3.asm: OS Kernel Flavors
Sum.asm, Collatz.asm, Primes.asm : Test Programs
NO COM FILES, NO NOTHING!!!
Pay attention to file naming

Tips & Tricks

1. You should study what to do, when an interrupt occurs.

2. You should study what to do, when context scheduling happens (What to save, what to update).

3. Start early,code often!!!

4. In our emulator there are no NMI's you can count on that. (If you dont know what it is look it up).

5. Do not forget to change Process States during context switching

6. Remember Your OS can access any part of the memory it wants.

7. During interrupt handling you should disable the interrupts by appropriate instruction and perform necessary tasks (Scheduling)

8. Be careful about the interrupt addresses, OS addresses, try to understand usage of **PCHL.**

9. **We have changed the Homework guidelines, pay attention to those instructions.**

10. Do **not** use **HLT in user processes**, Do **not** use **PCHL in user processes**, remember this is a special instruction.

11. Do not Disable or enable interrupts on child processes.

12. Please inspect the interrupt code and changes in the 8080emu.cpp to understand it.

General Homework Guidelines

1. No cheating, No copying, No peaking to other people homework
2. Follow the instructions very carefully.
3. Send required files only. Do not share your whole file system with us.
4. If you fail to implement one of the requirements, leave it be. Do not send an empty file
5. Respect the file names! Our HW grading is case-sensitive.
6. Failing to comply any of the warnings above will result in getting a **0** for your current homework.

Homework Instructions

1. Download and Install Vmware Player from Official site.
2. Download and install our virtual machine from https://drive.google.com/open?id=1YppX3lNkyTsHV_lvA4w9TomNCUkpLeEg
3. Carefully examine 8080 Cookbook and instruction set.
4. Use the assembler at http://asdasd.rpg.fi/~svo/i8080/ to develop your assembly code. You can download the output .com file directly using this assembler.