Gebze Technical University

Department of Computer Engineering

CSE 312 /CSE 504
Operating Systems Spring 2020

Homework #03
Due Date: June 20th, 2020
Scheduling in Assembly

This homework will be very similar to HW2. The main difference will be that you will keep your process table in SPIM memory as an assembly data structure. Your interrupt handlers will also be in MIPS assembly language.

You are going to develop a kernel that will support multi-programming, interrupt handling and memory management. Please refer to SPIM MANUAL for how SPIM architecture handles interrupts. (http://pages.cs.wisc.edu/~larus/SPIM/spim_documentation.pdf)

In the previous homework, you have modified the SPIM package to add these functionalities. You have implemented a new function named void SPIM_timerHandler() that handles process switching when there is a timer interrupt.

For this homework, your task is to develop kernels called SPIMOS_GTU_X.s that will perform interrupt handling, multi-programming and context switching in assembly language. Your new kernel will again be loaded as an assembly file as SPIMOS_GTU_1.s, SPIMOS_GTU_2.s, or SPIMOS_GTU_3.s.

What we expect from your new OS is

1) Handling multi-programming: you need to develop a Process Table in assembly code that will hold the necessary information about the processes in the memory. This assembly code and data structure will be in your kernel files.
2) Handling Interrupts: Our simulator will generate interrupts(SPIM_timerHandler), and your kernel will handle the interrupts in assembly.
3) Implementing the POSIX system calls fork, waitpid, execve, any other POSIX call that you need. These system calls will be implemented in syscall.cpp using C language.
4) Loading multiple programs into memory: Kernel will be able to load multiple programs into memory. This operation will be a system call.
5) Perform Round Robin scheduling: Every time a timer interrupt occurs, there is a chance to make a process switch as previous homework.
6) 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. ProcessID
    b. ProcessName,
    c. Programcounter
    d. Stack Pointeraddress

Life-Cycle

You will implement 3 different flavors of MicroKernel (SPIMOS_GTU_1.s, SPIMOS_GTU_2.s, and SPIMOS_GTU_3.s). 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 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 **init** process will initialize Process Table, 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 **5** times(Same program **5** 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 SPIM starts, it immediately loads the **MicroKernel file** given by commandline parameters example EX: *./spim -ef exceptions.s -file SPIMOS_GTU_1.s*
- For every timer interrupt, **OS** should handle the interrupt and perform round robin scheduling.
- Your programs finish execution it will acknowledge its termination by calling POSIX **PROCESS_EXIT**.
- Emulator will shut down only after all the programs in memory terminate.

Assembly Files

You will supply us with four different Assembly Files

- BinarySearch.s
- LinearSearch.s
- Collatz.s: You are going to find collatz sequence for each number less than 25. **You should do this starting from 25 to 1 iteratively (Not only for 1 number).** 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.
- Palindrome.s: Create a dictionary that contains 100 words, where 90 of the words are not palindrome, 10 of them are palindrome. You do not have to create the dictionary by taking from the user. Then, you will print out each word and whether they are palindrome or not respectively. When you assign all the words in the dictionary whether palindrome or not, then you ask for the user continue or not, if yes, take an input word and show whether a string given from keyboard is a palindrome by printing the "string" semicolon: "Palindrome" or "Not Palindrome". Otherwise terminate the program.

Output of the dictionary

1: aba: Palindrome
2: how: Not Palindrome
3: notbook: Not Palindrome
4: ada: Palindrome
5: arthas: Not Palindrome
…
…
…
100: last: Not Palindrome

Do you want to continue (y/n)?
y
Please enter the last word:
Ilhan
101: Ilhan: Not Palindrome
Goodbye…

Your homework template includes several code and sample files. Here is a description for them. Do not change SPIM files except syscall.cpp and do not send these files with your homework. Study these files to understand SPIM.

What To Submit

syscall.h
syscall.cpp
SPIMOS_GTU_1.s, SPIMOS_GTU_2.s, and SPIMOS_GTU_3.s: OS Kernel Flavors
Test Programs: LinearSearch.s, Collatz.s, BinarySearch.s, Palindrome.s
README : sample running instructions
REPORT: write a report to brief your work, including your results, comments

Homework instructions

1. **Your code should have comments, for each 4 lines of code, write a line of comment so that we understand your code!**
2. You should never submit all SPIM Source Code. Submit only asked files. **Otherwise your homework will not be graded!**
3. Your code should not have compilation errors
4. Name files as specified
5. You should study what to do when an interrupt occurs.
6. You should study what to do when context scheduling happens
7. Start early, code often!!!
8. Do not forget to change Process States during context switching
9. Remember Your OS can access any part of the memory it wants.
10. During interrupt handling, be careful about interrupts happening again.
11. Download and Install Vmware Player from Official site.
12. Download and install our virtual machine from https://drive.google.com/open?id=1YppX3lNkyTsHV_lvA4w9TomNCUkpLeEg
13. Download the SPIM package from the same location as HW2, and use run.cpp, syscall.h and syscall.cpp as HW2

General Homework Guidelines

1. No cheating, No copying, No peaking to other people homework
2. Follow the instructions very carefully.
3. If you fail to implement one of the requirements, leave it be. Do not send an empty file