Gebze Technical University

Department Of Computer Engineering

CSE 312 /CSE 504

Operating Systems Spring 2020

Homework #02

Due Date: June 1st 2021

Interrupts, Threads, Multiprogramming

In this homework you are going to develop a kernel that will support multi- threading, interrupt handling and inter-process communication. Please study SPIM MANUAL for how SPIM architecture handles interrupts. (http://pages.cs.wisc.edu/~larus/SPIM/spim_documentation.pdf)

We have modified the SPIM package to make it easier for your too add these functionalities. We have added a new function named void SPIM_timerHandler() that gets called every time there is a timer interrupt. The timer interrupt happens about every 100ms.

You task is to develop a part of the kernel called SPIMOS_GTU_X.s that will perform interrupt handling, multi-threading and context switching. Note that this is not a multi process handling modification, you will handle only multi threads within a single process. Therefore, you will not need more than one address space, all address spaces will be shared. Do not forget to handle separate spaces for each thread!

Your new kernel will be loaded as an assembly file. In other words, instead of loading assembly files, you will load SPIMOS_GTU_1.s, SPIMOS_GTU_2.s, or SPIMOS_GTU_3.s

What we expect from your new OS is

1) Design and implement these POSIX thread system calls: create, join, exit, mutex lock, mutex unlock and any other POSIX call that you need. These system calls will be in syscall.cpp. Do not modify any other CPP files from the SPIM package.
2) Handling multi-threading: you need to develop a **Thread Table** that will hold the necessary information about the threads in the memory.
3) Handling Interrupts: Our simulator will generate interrupts, and your kernel will handle and respond by modifying SPIM_timerHandler function in syscall.cpp
4) Perform Round Robin scheduling: Every time a timer interrupt occurs, there is a chance to make a process switch.
5) Whenever a context switching occurs, you will optionally print all the information about the processes in process table including but not limited to the entries in the list below.

    a. Thread ID
    b. Thread Name,
    c. Program counter
    d. Stack Pointer address

You will implement 2 different MicroKernel(SPIMOS_GTU_1.s, SPIMOS_GTU_2.s). When your kernel is loaded your OS will start a process named **init.** In different Micro Kernels Init process will create different threads as explained below.

- In the first case, **init** process will implement a multi-threaded merge sort such as defined in https://www.geeksforgeeks.org/merge-sort-using-multi-threading/ . Change your array size and the number of threads in your experiments.

- In the second case, init process will implement a producer consumer problem using one producer thread and one consumer thread. The problem will be very similar to what we have seen in our lectures. Note that we don't have space for a single buffer, so using only mutexes should be enough for inter process communication. You will have two versions of this case: in the first version do not use any mutexes and show that there is a race condition. In the second, implement a version that has no race condition problems.

When SPIM starts, it immediately loads the **MicroKernel file** given by command line parameters example EX: `./spim -ef exceptions.s -file SPIMOS_GTU_1.s`

Your programs finish execution it will acknowledge its termination by calling POSIX **PROCESS_EXIT**.

I

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

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.

syscall.h : sample header for SPIM OS. You can rewrite this file
syscall.cpp : sample implementation for SPIM OS. You will rewrite this file
SPIMOS_GTU_1.s and SPIMOS_GTU_2.s: OS Kernels
README : sample running instructions
HW2 Report: Your design decisions with figures, your thread table structure, your experiment results with version 1 and version 2 and any other information that needs to be reported.

Pay attention to file naming Tips & Tricks

1. You should study what to do, when an interrupt occurs.
2. Do not forget to change Thread States during contextswitching
3. Remember Your OS can access any part of the memory it wants.
4. During interrupt handling, be carefull about interrupts happening again (Schedulin
5. Please inspect the interrupt code and changes in the run.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. Download the SPIM package from the same location as HW1, replace run.cpp, syscall.h and syscall.cpp with our source codes.