Indian Institute of Technology Guwahati
Department of Computer Science & Engineering

CS 588: Systems Lab
Assignment – 2

Instructions:

- Make sure that you read, understand, and follow these instructions carefully. Your cooperation will help to speed up the grading process.
- Following are generic instructions. Make sure that you also check carefully and follow any specific instructions associated with particular questions.
- In this assignment, you will explore the dynamics and challenges of time and memory management in operating systems.
- Complete the assignment with the best of your own capabilities. Create a single zipped/compressed file containing all your programs and related files.
- The compressed file must contain:
  - All your program files
  - Makefile to compile your programs
  - Readme file to explain your program flow (individual)
  - No output files
- The file name should be as your roll no. Example: 174101001.zip. When done, submit the file in Moodle on or before the submission deadline.
- **Submission deadline: 08:00 PM, Wednesday 14th March, 2018 (Hard Deadline).**

Ethical Guidelines (lab policy):

- Deadlines: Deadlines should be met. Assignments submitted after their respective deadlines will not be considered for evaluation.
- Cheating: You are expected to complete the assignment by yourself. Cases of unfair means and copying other's solution will not be tolerated, even if you make cosmetic changes to them. If we suspect any form of cheating, we are compelled to award ZERO marks.
- If you have problem meeting a deadline, it is suggested that you consult the instructor and not cheat.

Question-1: Time Triggered Tasks

The objective of this question is to familiarise with timed triggering of tasks. Standard Linux implementation uses **cronshell** and ***/etc/crontab*** for managing periodic activities. Write a customised version of your cronshell, named "**mycronshell**". The configuration of scheduled activities are to be stored in a file ***/home/mycrontab***. One sample entry of this file is given below.

```
15      30      *       *       *       cd /home/abhijit/CS558
*       *       *       *       *       |
-       -       -       -       -       +------command to be executed
|       |       |       |       |
|       |       |       |       +------ day of week (0 – 6) (Sunday = 0)
|       |       |       +------ month (1 -12)
|       |       +------ date of month (1 -31)
|       +------ hour (0 – 23)
+------ minute (0 – 59)
```

mycronshell must parse the configuration file, whenever there is a change of content. Based on the configuration file, it should invoke tasks as per the schedule.

Question-2: Page Table Simulation

Page Table (PT)

A structure with the following fields:

- ***Valid*** : If the page is valid
- ***Frame*** : Alloted frame
- ***Dirty*** : If the frame is modified (in case of write)
- ***Requested*** : If a frame is requested for the page

Operating System (OS)

Takes the following arguments:

- Total count of pages (***P***)
- Total count of frames (***F***)

Memory Management Unit (MMU)

Takes the following arguments:

- Total count of pages (***P***)
- A memory trace conatining virtual memory address and mode of operation:
  - 0xe5450 R //Virtual address in Hex form, Mode of operation
  - 0x200 R
  - 0xe5450 R
  - 0x105ff0 W
  - . . . .
- Page number bits (**PAGE**)
- Offset bits (**OFFSET**)
- PID of the Operating System (**OS**)

Detailed Explanation:

1. There are two separate simulations: **OS** and **MMU** and they communicate via a shared memory.
2. The OS simulation will create a **Page Table** in shared memory using page count (***P***) and page table structure (**PT**).
3. **OS** simulation then initialises the **Page Table** by setting the ***Valid*** field of all the pages to **0**.
4. **OS** simulation will now sit in a loop and wait for a signal (**SIGUSR1**) from **MMU**.
5. **MMU** simulation will attach itself with the **Page Table** (which is in shared memory).
6. **MMU** will then translate virtual addresses (from memory trace) one by one to identify the page number.
7. For each identified page number:
   (a) Check if the page is in the table. If yes, increment **Page_Hit** counter. If not, increment **Page_Fault** counter.
   (b) If not in the table, Write the PID of **MMU** into the ***Requested*** field of that page.
   (c) Simulate a page fault by signalling (**SIGUSR1**) the **OS** and go to sleep.
   (d) Block until a continue signal (**SIGCONT**) is not received from the **OS**.
   (e) If the Mode is Write, set ***Dirty*** field to 1.
8. When all the memory accesses are processed, **MMU** detaches from the **Page Table** and signal **OS** (one last time).
9. Upon receiving a signal (**SIGUSR1**) from **MMU**, the **OS**:
   (a) Traverses the **Page Table** for a non-zero ***Requested*** field.
   (b) If found, it is the PID of **MMU** and indicates that the **MMU** is requesting a page (to load a frame) at that index.
   (c) If there is a free frame, allocate it to that page.
   (d) If there are no free frames, choose a victim page with a page replacement policy (**will be given later**).
   (e) If the victim page is dirty, simulate write back to the disk by sleep and increment **Disk_Access** counter.
   (f) Simulate page load by sleep and increment **Disk_Access** counter.
   (g) Update the page table entry by setting ***Valid*** field to 1, ***Frame*** with a number, ***Dirty*** to 0 and ***Requested*** to 0.
   (h) Send continue signal (**SIGCONT**) to **MMU** indicating that the page is now loaded.
   (i) If no non-zero ***Requested*** field is found, destroy the **Page Table** and exit.
10. Print the updated Page Table at regular intervals.
11. Compare the page fault count with the following replacement algorithm:
    (a) Roll_number % 4 = **0** : FIFO and LRU
    (b) Roll_number % 4 = **1** : FIFO and MRU
    (c) Roll_number % 4 = **2** : FIFO and LFU
    (d) Roll_number % 4 = **3** : FIFO and NMRU

Implement **OS** simulation and **MMU** Simulation separately in C and make them communicate using shared memory for **Page Table**.

Sample Run:

OS	MMU
./a.out 1024 32 //P= 1024, F = 32; remember P >> F Page Table created Page Table initialised Shared Memory ID (PID) if 2018	./a.out 1024 memtrace.out 12 10 2018 //P = 1024, Trace = memtrace.out, PAGE = 12, OFFSET = 10, OSPID = 2018 MMU is attached with Page Table Page Table 0: Valid=0 Frame=-1 Dirty=0 Requested=0 1: Valid=0 Frame=-1 Dirty=0 Requested=0 2: Valid=0 Frame=-1 Dirty=0 Requested=0 . . . . Request for Page = 2 in Mode = Read Not present in Page Table : Page Fault
Procees = 2019 has requested for Page = 2 Allocate Page = 2 to free Frame = 0 Resume MMU operation	Page Table 0: Valid=0 Frame=-1 Dirty=0 Requested=0 1: Valid=0 Frame=-1 Dirty=0 Requested=0 2: Valid=2 Frame=0 Dirty=0 Requested=0 . .