Assignment 10

Operating System Lab (**CS342**)

Department of CSE, IIT Patna

Date:- 26-Mar-2018 **Time:-** 3 hours

Instructions:

1. All the assignments should be completed and uploaded by **1-Apr-2018, 5pm.**
2. Markings will be based on the correctness and soundness of the outputs. Marks will be deducted in case of plagiarism.
3. **Proper indentation and appropriate comments are mandatory.**
4. You should zip all the required files and name the zip file as ***roll_no*.zip**, eg. **1501cs11.zip.**
5. Upload your assignment (**the zip file**) in the following link: https://www.dropbox.com/request/ZOKRSGd68eA21k2UwZfC

1) Virtual memory allows a process of P pages to run in F frames, even if F < P. This is achieved by use of a page table, which records which pages are in RAM in which frames, and a page fault mechanism by which the memory management unit (MMU) can ask the operating system (OS) to bring in a page from disk. The page table must be accessible by both the MMU and the OS, and an IPC facility is needed for communication between the MMU and OS. Given the MMU simulator below, write a program that does the work of the OS to maintain a process' use of RAM and the page table. The page table is held in shared memory, and signals are used for IPC. You do not need to simulate the RAM, only the page table maintenance. However, to make things realistic, your OS process must sleep(1) whenever a disk access (write out page to disk, read in page from disk) would be necessary in a real implementation.

The Page Table

The page table has four fields in each page table entry:

- A Boolean Value indicating if the page of that index is in RAM.
- An integer Frame giving the frame number of the page in RAM.
- A Boolean Dirty indicating if the page has been written to.

- An integer Requested which is non-zero only if that page is not in RAM and has been requested by the MMU. In this case it's value is the PID of the MMU.

The OS process must create the page table in shared memory, and initialize it to indicate that no pages are loaded (all Valid fields set to 0). You may need to add more fields for your OS, with corresponding changes in the MMU.

The MMU

This memory management unit simulator takes several arguments:

- The number of pages in the process.
- A reference string of memory accesses, each of the form <mode><page>, e.g., W3 is a write to page 3.
- The PID of the OS process.

The MMU attaches to the shared memory (using the OS PID on the command line as the key), then runs through the reference string. For each memory access, the MMU:

1. Checks if the page is in RAM.
2. If not in RAM, writes its PID into the Requested field for that page.
3. Simluates a page fault by signalling the OS with a SIGUSR1.
4. Blocks until it receives a SIGCONT signal from the OS to indicate that the page has been loaded (well, as mentioned above, the load is not done in this project, just simulated by sleep(1) delays).
5. If the access is a write access, sets the Dirty bit.
6. Prints the updated page table.

When all memory accesses have been processed, the MMU detaches from the shared memory and signals the OS one last time, but without placing its PID in any Requested field. That must be detected by the OS, at which point it can destroy the shared memory and exit.

The OS

The OS simulator must take two arguments:

- The number of pages in the process.
- The number of frames allocated to the process.

For simplicity, assume that the pages and frames are numbered 0, 1, 2, ...

After creating and initializing the page table in the shared memory, the OS must sit in a loop waiting for a SIGUSR1 signal from the MMU. When it receives a signal, it must:

1. Scan through the page table looking for a non-zero value in the Requested field.
2. If a non-zero value is found, it's the PID of the MMU, and indicates that the MMU wants the page at that index loaded.
3. If there is a free frame allocate the next one to the page.
4. If there are no free frames, choose a victim page.

    - If the victim page is dirty, simulate writing the page to disk by sleep(1) and increment the counter of disk accesses.
    - Update the page table to indicate that the victim page is no longer Valid.
5. Simulate the page load by sleep(1) and increment the counter of disk accesses.
6. Update the page table to indicate that the page is Valid, in the allocated Frame, not Dirty, and clear the Requested field.
7. Print the updated page table.
8. Send a SIGCONT signal to the MMU to indicate that the page is now loaded.
9. If no non-zero Requested field was found, the OS exits the loop.

You can use whatever algorithm you want for choosing a victim page. You may need to add extra fields to the page table, and make *minor* modifications to the MMU code. Before terminating the OS must print out the total number of disk accesses, and destroy the shared memory.