CS401: ADVANCED OPERATING SYSTEMS

Assignment 2

Duration: 18 Days Total Marks: 20

In this assignment, you will add the implementation of the memory management aspects of an Operating System to your earlier process simulator. These are the experiments for you to do and submit the results by the due date.

1. Add **timing structures and functions** to your simulator, if you have not already done so, for measuring

   - RESPONSE TIME: the time interval between process creation and the first time it is scheduled
   - RUNNING TIME: the total time taken for a process to complete execution; this is the time measured from the time it is first scheduled to the time it completes
   - WAIT TIME: the total time process spent in *ready* and *wait* states

   Include the header file `sys/time.h` and use the function `gettimeofday()` to measure time with *microsecond* resolution.

2. Add **memory-related data structures and variables** necessary for implementing the virtual memory manager.

3. **Modify the process** from the previous assignment and include a *logical memory address reference* with every instruction. You have a choice here:

   - Implement your own memory address reference generator with the following properties:
     (a) The first ≈ 100 address references are from the first page and then page faults frequently while adequate number of pages are loaded into physical memory.
     (b) The process then alternates between stable phases with few page faults and transitions with frequent page faults

     OR

   - Use the address reference generator provided by me
     (a) Include the header file `ScisSosMem.h`
     Include `ScisSosMem.h` only in the C file with memory related functions. Don't include it in more than one C file!!!
     (b) Use the function

     ```
     int *memory_gen_addrefstrings(int, int)
     ```

     The first parameter is `size` (the process size); the second is `mtype` (memory type: `GOOD, BAD` or `UGLY`.
     `GOOD` means that the working set sizes are small and the transition states are also small so that the process generates relatively few page faults.
     `BAD` is a process whose working set size is larger and generates more page faults.
     `UGLY` is a process whose working set is large (may not be easy to load into memory) and generates many page faults: may be *spaghetti code*.

To access the function
Dowload the files `libscismem.a` and `ScisSosMem.h`
Add `#include <ScisSosMem.h>` in the source code
Compile as
`gcc <C source file> <other object files> -L. -lscismem`

You can incorporate the generated address references into the process designed in the previous assignment.

4. Write a **virtual memory manager** that implements

   - VIRTUAL PAGING: when a process is created, the virtual memory manager loads *its first page into physical memory* and then moves it from *new* to *ready* state.
   - HANDLE PAGE FAULTS: when a process requests for a memory address that is not currently in a page loaded in memory, the manager loads the required page into main memory and then moves the process into *ready* state.

     If there is no *free frame* in main memory, it calls a *global page replacement algorithm* (see below) to identify a frame that may be replaced with the requested page. After replacing the page, it moves the process into *ready* state

Implement at least **two page replacement algorithms**. Both must be *global replacement* algorithms.

1 Data Structures

Implement the following data structures.

Reference Window: is an integer Δ which represents a sliding time window for determining whether a page in memory has been referenced. Set $\Delta = 1024$. We can vary this parameter later.

Main Memory: This is an array of $K = 64$ frames. You can experiment with K in the experimentation. Each entry in the main memory contains the following

<Page Number>, <PID>, <flags>

PAGE NUMBER is the current page number in that frame, PID is the PID of the process whose page number is in the frame. FLAGS are two bits: one, the *dirty* bit, indicates if the page has been referenced in the previous Δ address references by that process. The second bit is a *use* bit which is set to one if the page has been modified by the process. Currently, this is unused.

2 Experiments

Do the following to test the simulator as well as to gain some insights into the working of a virtual memory system.

1. Create 10 processes of different types: COMPUTE INTENSIVE, IO INTENSIVE and REGULAR. Assume that the memory size = process size. Let all the processes be either GOOD or BAD from the memory management perspective.

(a) **Turn OFF** the memory management functions and let the processes run. Save their timing information.

(b) **turn ON** the memory manager and observe how the timings get impacted. Write your findings clearly.

2. Create 10 processes of the **same type** computationally. Let 3 of them be GOOD and BAD each and 4 be UGLY with $K = 64$. See how the timings get impacted as you vary K from $K = 32$ to $K = 128$ in steps of 16 each. Write down your observations clearly.

3 Implementation Instructions

Please organise your code in multiple files if you implement the assignment in C language (preferred). You may also choose to implement it in C++, Java or Python although the last two are not really nice!

Here are some suggestions:

- Create a *header* file to store all your constants, data types and function declarations.
- Create one file for all process related activities such as `create process, run process, delete process, create PCB, print PCB`, etc.
- Create a separate file containing functions for the simulator. You really need two functions here: `initialise OS parameters` and `scheduler`.
- Create a separate file for the scheduling algorithm your group (see below) chooses to implement.
- Create a separate file for virtual memory related functions including the page replacement algorithms.
- Write a final file that contains the `main()` function.
- Compile the first four files separately (using the `-c` option to `gcc`) and then link them together when compiling the file containing the `main()` function.

  You may also need to use the library `libscismem.a`.

You may form groups of two each for writing the overall simulator. Also, discuss among the groups so that every group's page replacement algorithms can run with another group's virtual memory manager.

You will have to demonstrate the simulator to me or the TAs at a scheduled date and time after the due date which is **6:00 PM on Monday, 24 November 2025.**