OS Lab Assignment **, 13 March 2015

Simulation of A Basic Virtual Memory

The goal of this assignment is to simulate a basic virtual memory system and demonstrate its working. This document contains the spec of the simulator. You must design appropriate data structures for its operation. The simulator has one primary thread that does some initializations and creates a separate thread for each user process in the system.

1 The primary thread

The primary thread controls the simulation. It reads a file named `init`. This file contains a list of *create* commands, where a *create* command initializes simulation of one process. **Each command appears on a separate line.**

1. The *create* command has the syntax:

   `create <process_no> <process_size> <page_frames_allocated>`

   where

   `<process_no>` is simply an integer. (Do not put '<' and '>')

   `<process_size>` indicates the size of the logical address space of the process, i.e., the number of pages in the logical address space of the process. Note that pages are numbered from 0 onward.

   `<page_frames_allocated>` is a spec `m-n` where `m` and `n` are integers. Note that a process is permanently allocated a contiguous set of frames of memory starting on page frame `m` and ending on page frame `n`.

2. On reading a `create` command for a process, the primary thread creates a new thread, which we will call the *process thread* to simulate operation of the process.

3. **Note:** You MUST use the following fixed format: each command in a separate line where the word `create` appears in character positions 1–6, the process no in positions 8–9, process size in lines 11–13, and page frames allocated in positions 14-20 in the format *nnn-mmm*.

2 The process thread

A separate spec file is read by each process thread. We call it the *spec file of a process*. The spec files of processes $1 \ldots n$ are named `s1`, `s2`, .. `sn`. A spec file contains a list of `read` or `modify` commands that indicate a reference or a modification, respectively, of a word in a page of the logical address space of the process, followed by an `end` command. **Each command appears on a separate line.**

1. The syntax of access/modify commands in the spec file of a process is:

   `access <page no> <word no>`

   `modify <page no> <word no>`

   where `<page no>` and `<word no>` are simply integers, i.e., no <, >. Use the following fixed format for each command: word `access` or `modify` in positions 1–6, page number in positions 8-10, and word number in positions 11-14.

2. The end command has the syntax `end`.

3. On reading an access/modify command, the process invokes the function named `mmu` to simulate operation of the virtual memory hardware. This function returns with status codes to indicate

   (a) Whether a memory protection violation occurred while implementing the access/modify command.

   (b) Whether the access or modify operation could be implemented on the page (i.e., whether the page was found to be present in memory)

   (c) Whether a page fault occurred.

4. In the event of a page fault, the process thead invokes a function named `page_fault_handler`.

5. After executing each access/modify command, the process thread prints a line

   ```
   process number:  attempted to access/modify <page no> <word
   no>
   Reported a page fault/memory protection violation
   Accessed page frame number <no>
   ```

6. On reading the `end` command, the process thread prints the following summary:

   `Number of access operations :` *nnnn*
   `Number of modify operations :` *nnnn*
   `Number of page faults :` *nnnn*

3 The function `mmu`

This function simulates the MMU. It takes a process id and a page number as its inputs. It raises a page fault or reports memory protection violation as appropriate.

4 The `page_fault_handler`

When given the page no of a page for which the page fault occurred, the page fault handler performs LRU-based page replacement on a *local* basis (i.e., replaces one of the pages of the process that caused the page fault, and loads the new page in that page frame), and returns the frame number where the page would be loaded.

It prints report lines when appropriate:

```
Removed page <page no> from frame <no>
Loaded page <page no> into frame <no>
```

5 Evaluation

Your code would be evaluated by an automated script. Hence you MUST strictly follow the specs about names and formats of input files.

6 Corrections in this specification

The right to make corrections/enhancements reserved.