1 Getting started

Begin by getting the repository and code set up...

1. **Login to the server** via `ssh`.

2. **Login to GitLab** in your browser.

3. **Start a new project**: Set the *Project name* to be *sysproj-7*.

4. **Clone the repository onto the course server**:

   ```
   $ git config --global user.name "Your Name"
   $ git config --global user.email "yourusername@amherst.edu"
   $ git clone git@gitlab.amherst.edu:yourusername/sysproj-7.git
   $ cd sysproj-7
   ```

5. **Download the source code**:

   ```
   $ ls -l
   ```

6. **Add/commit/push the source code to the repository**:

   ```
   $ git add *
   $ git commit -m "Starting code."
   $ git push
   ```

2 Looking inside vmsim

For this project, you will be working within *vmsim*. I have provided all of its code (modified somewhat from the previous project), including my own MMU implementation. Your work will all be within *vmsim.c*, modifying to do new things.

The most immediate change is the presence of the *bs.c* and *bs.h* files, which implement a simulated *backing store*—a disk-like larger storage that allows you to read and write whole blocks (each conveniently 4 KB).

Additionally, you can now look inside *vmsim.c* to see how it works. Of particular interest is the function *vmsim_map_fault()*, since it is responsible for handling MMU translations that fail. You should also notice, in *mmu.c*, that the MMU now does two new, important things:
1. **Test the resident bit:** Each page table entry uses the bit in position 0 to indicate whether that simulated page is mapped to an honest to goodness real page that is available and ready for use. If this bit is 0, the translation fails.

2. **Set the referenced and dirty bits:** When a translation succeeds, the bit at position 1 is set (to 1), indicated that this simulated page has been referenced. If the reference is a write operation, then the bit at position 2 is set (to 1), marking the simulated page as dirty.

There are likely other features that you will want to take in, including a number of `#define` macros that I’ve used for manipulating bits, various helpful constants, etc. Get your head wrapped around the code.

### 3 Creating a page swapping mechanism

Notice that the new backing store device is not initially used by the provided code. This code will compile and run, but the real memory is small. Any program that uses 1 MB or more will fail.

**Your task** is to make use of the backing store device to swap pages to and from real memory. Each time you do, the page tables must be updated to reflect the change. Simulated pages backed by real memory should have their resident bit set and their translations should succeed; those not backed by real memory, and held only in the backing store, should have this bit cleared so that translations fail. The `vmsim_map_fault()` function identifies attempted uses of pages on the backing store and initiates a page swap. How you choose to approximate the least recently used policy in order to select a page in real memory for replacement is up to you.

Notice the `reverse_page_map`, which is an array of pointers to lower-level PTE’s. Specifically, for a given PTE that maps to a particular real page, this array then contains an array entry that points back at the PTE itself. This data structure may be quite helpful in choosing a page to replace, and then replacing it.

### 4 How to submit your work

First, be sure that the most recent versions of your work are up-to-date on the GitLab server by performing an `add/commit/push` with `git`. Then, go to GitLab with your browser, and add me (`sfkaplan`) as a `Developer` to your repository.

This assignment is due on Thursday, May-05, 11:59 pm.

---

1. Notice, also, that `mmu_translate()` now has a second parameter that indicates whether the memory reference is a read or write operation.

2. Of the 5 MB in the default real memory size, the first 4 MB + 4 KB are reserved for the page table; slightly less than 1 MB is available for backing simulated pages.