CS 4348 Project 2

The goal: Simulate a memory system

Due: April 9th, 2024

GitHub Classroom Link: https://classroom.github.com/a/_FveF2T0

The Setup

As before, all you need to get started will come from GitHub. This time there are 3 JUnits, detailed below. Splitting the tests should make it easier to tackle the project a step at a time, and it allows you to see your project progress in GitHub (since each JUnit class is run separately this time).

You will create 2 new classes, and modify two lines in `StudentSubmissionFactory` (to create those two classes).

The other classes you'll work with (but should not touch!) are:

- `Frame`: represents a physical block in memory. All you'll do with it is put pages there and remove pages from there.
- `LookupInfo`: holds the input *and output* for a logical->physical address conversion. The logical address info (the address, the process, etc.) will be given to you, and you will fill out the matching physical address and if there was a TLB hit for this request.
- `MainMemory`: base class for your Memory class (see below).
- `PageTableEntry`: a…page table entry. 😀 This is what you'll assign to/remove from a `Frame`.
- `Process`: a process, which includes its page table. You will barely use it, mainly to look up pages.
- `StudentSubmissionFactory`: where you will `new` your main memory and TLB classes. The factory lets the rest of the code be ignorant of the specific type you're using.
- `TLB`: base class of your TLB implementation
- `Util`: contains a few constants you should find useful 😀

Running All the JUnits

Here's how to run all the JUnits at once from within IntelliJ

1. Go to `Run->Edit Configurations` (If I remember right, in Windows the menu is hiding at the top of the window near the name of the project)
2. Press the '+' button and select "`JUnit`"
3. Give it a name 😀
4. For the module, any of the dropdown options should be fine
5. For "`-cp`" select `cs-4348-project-2.test`
6. Change "Class" to "All in Package" and type in `edu.utdallas.cs4348`
7. Your configuration should now show up towards the top-right; the play button will run them.

What You'll Do (in the order you should do them)

1. `TestAddressConversion`: conversion of a logical address to a physical one.
    1. Create a subclass of `MainMemory` named after your *last* name, such as `PetersonMainMemory`. Update `StudentSubmissionFactory.createMainMemory` to create an instance of your class, passing the two arguments along to the constructor.
    2. The constructor for your subclass should pass those two arguments to the superclass. (`MainMemory`'s constructor will create the array of `Frames` you will use below.)
    3. In `addPageToMemory()`, put the given `PageTableEntry` into the first empty slot in the `frames` array. (Don't worry about running out of slots yet; that's for the next test. 😀)
    4. Do the logical->physical address conversion like we did in class (the slides with the bit shifting; we also did an example on the board in class). `LookupInfo` has what you need to get the proper `PageTableEntry` which, in turn, will have the matching `Frame`. **`Util` has 2 constants that will help!** If the page isn't in main memory (which will happen!), `getFrameNumber()` will return -1, which you should make the "physical address" for that logical address.
2. `TestFrameVictimization`: dealing with main memory's limits.
    1. We aren't doing any fancy page replacement: just evicting the least-recently-**added** (not used) page.

        1. And since you will be the only one removing pages out of main memory (i.e. no pages will leave “on their own”), your algorithm for choosing the next page to evict should be simple. 😀
    2. At the end of `addPageToMemory()` you’ll cover the case where you can’t find an empty frame by replacing the least-recently-**added** one with the new one.
3. `TestTLB`: add a TLB to the memory system
    1. Create a subclass of `TLB` named after your *first* name; mine was `ErikTLB`. You don’t need to create a constructor; the superclass does the necessary setup.
    2. Update `StudentSubmissionFactory.createTLB` to `new` an instance of your new class (no arguments needed).
    3. For `addEntry()`, attempt to put the page into the first empty slot you can find. If you can’t place it, you will evict the least-recently-**used** entry. (PageTableEntry has the last access time for a page, and the `access()` method will update that timestamp.)
        1. BIG NOTE: to get the needed time precision we’re using `System.nanoTime()` for our timestamps. (`Date` only does millisecond precision, which isn’t enough for us. So if you need the current time, use `System.nanoTime()`.)
    4. For `lookup()`, scan through the entries in the TLB table to see if there’s a match for that process and page table entry combination. If so, `access()` that page table entry to show that it was hit and return it. Return `-1` if you can’t find a match.
    5. Add a TLB lookup call to `<YourMainMemory>.getPhysicalAddress()` before you look up the page via the process’s page table. *Note that there won’t always be a TLB*…such as in the previous tests. 😀 If it’s a TLB hit, note that hit in `LookupInfo` and use the match as the frame number.