Phase 5 Notes and Hints

- Goal:
    - Implement a virtual memory system that supports demand paging.
- Single-level page table for a region of address space.
    - Each process will have its own idea of what is in the VM memory region.
- **`VmInit()`**:
    - Initializes the VM region:
        - Installs the handler for **`USLOSS_MMU_INT`** interrupt (into **`USLOSS_IntVec`**).
        - Creates pager daemons (no more than **`MAXPAGERS`**).
        - Calls **`USLOSS_MmuInit()`** to initialize the MMU.
        - Number of mappings should be equal to the number of virtual pages.
            - Return **`-1`** if not.
        - Current tag: set to **`0`**. Will not change this. (Does change, possibly, if attempting extra credit.)

To use gdb with the MMU, add the following to *.gdbinit*:

```
handle SIGUSR1 nostop noprint
handle SIGSEGV nostop noprint
handle SIGBUS  nostop noprint
```

The first line should already be there. The other two are needed when using the MMU. Note: this may result in gdb ignoring a segmentation fault or bus error in your code(!!).

- **`VmDestroy()`**:
  - Shuts down the VM system.
  - Page daemons killed.
  - Any allocated memory freed.
  - Turns off MMU.

- Processes:
  - Each has its own page table.
  - Can use **`malloc`** to allocate these
    - We do not have an upper limit on the possible number of pages or frames that can be in the VM region.
  - Page table records whether the page is in memory and/or on the disk (can be both places!).
  - Page tables are managed by **`p1_fork`**, **`p1_switch`** and **`p1_quit`** (invoked from phase 1).
  - Page table created by **`p1_fork()`**.
  - Page table destroyed by **`p1_quit()`**.

- **`p1_switch()`**:
    - Unloads all the mappings from the old process (if any), and loads all valid mappings (if any) from the new process.
    - A valid mapping is one where there is a frame for the page.
        - If there is no frame, the MMU should not have a mapping for that page.
- When there is a reference in the code to a non-existing page, there will be an MMU interrupt (page fault) and a page will then be allocated.
- Complication: **`p1_fork`**, etc. are called before **`VmInit()`** has been called. These routines should do nothing in this case.

- Page Fault:
  - Occurs when the MMU does not have a mapping for an accessed page. **`USLOSS_MMU_INT`** interrupt.
  - Handled by one of several pager daemons.
    - *Kernel* processes running at priority 2 with interrupts *enabled*.
    - Responsible for moving pages between memory and the disk (uses only disk1).
    - There are several (up to **`MAXPAGERS`**) to allow multiple page faults to be serviced simultaneously.
    - Forked in **`VmInit()`** and killed in **`VMCleanup()`**.
  - New mapping is not loaded in the MMU by the pager, because this must be done by the process itself (what happens in context switch?).
  - Pager then initializes the contents of the frame.
    - If the page is on the disk, it must be read with disk_read.
    - If this is the first time the page is referenced, it will not have any contents, and the pager will erase the old contents of the frame by setting all the bytes of the frame to zero.
    - In either case, the pager must get access to the frame.
      - The pager must load a mapping into its own MMU to get access. This mapping should be removed after the page is filled.

- More on Pager daemons:
  - The **`DiskDriver`** will not be able to access a page frame.
    - It will not have the mapping.
    - It was created before the VM system was initialized, so it <u>cannot ever</u> have a mapping.
    - Simplest solution: the pager uses an intermediate buffer that is outside the VM region for disk reads and writes.
  - Use the “clock” algorithm to find the victim frame when there is no free page frame.
    - One clock hand is shared between all the pagers.
  - Mutual exclusion needed for free list, page frames, and clock hand.
  - All pages live on the disk, and some are in memory. Must keep track of the disk location for each page.

- Incremental Implementation:
  - Do this! (We will assume you are doing so when you ask questions!)
  - This is complex code: It is best to be able to rely on some of it while debugging other parts.