The third phase of Nachos is to support virtual memory. There is no new code provided for this assignment; your job is to modify your implementation for Project 2 to provide the abstraction of an (almost) unlimited virtual memory size to each user program.

In order to set up the directory structure for this assignment, do the following:

1. Save a copy of your project 2 directory (i.e., `cp -r nachos2 nachos3`).
2. Change into your nachos3 directory.
3. Copy from the course webpage the file nachos3.tgz into the current directory.
4. Execute: `tar -xzvf nachos3.tgz`

This replaces Makefile and Makefile.common in your nachos3 top-level directory. It also creates the `vm/` subdirectory with its own Makefile. You may have to edit one or more Makefiles to handle any specifics of your current implementation (such as new files you added for the second project). When you invoke make in the Nachos top-level directory, a nachos binary will be produced in `vm/`, but it will use source files in the `threads/` and `userprog/` directories.

The illusion of unlimited memory is provided by the operating system by using main memory as a cache for the disk. Page tables were used in project 2 to simplify memory allocation and to isolate failures from one address space from affecting other programs. For this assignment, page tables serve a few more purposes: the valid bit in the page table entry tells the hardware which virtual pages are in memory and which are stored on disk, and the hardware sets the use bit in the page table entry whenever a page is referenced and the dirty bit whenever the page is modified.

When a program references a page that is not resident in memory (detected when the page table entry is not valid), the hardware generates a page fault exception, trapping to the kernel. The operating system kernel then must read the page in from disk, set the page table entry to point to the new page, and then resume the execution of the user program. Of course, the kernel must first find space in memory for the incoming page, potentially writing some other page back to disk, if it has been modified.

As with any caching system, performance depends on the policy used to decide which things are kept in memory and which are only stored on disk. On a page fault, the kernel must decide which page to replace; ideally, it will throw out a page that will not be referenced for a long time, keeping in memory those pages that are soon to be referenced.

---

1 From the original Nachos Assignment #3 by Tom Anderson; modified by Phil Kearns and later Joel Hollingsworth, Barry Lawson and Lewis Barnett.
Your assignment is to implement virtual memory in Nachos. For this, you will need kernel routines to move a page from disk to memory and from memory to disk. Use a single SynchDisk as your backing store (swap space). The SynchDisk has the useful property that a disk sector and a page are exactly the same size. In order to find unreferenced pages to throw out on page faults, you will need to keep track of all of the pages in the system which are currently in use. A simple way to do this is to keep a “core map” (or a “frame table”), which is basically a reverse page table — instead of translating virtual page numbers to physical pages, a core map translates physical page numbers to the virtual pages that are stored there. (This is in addition to the bitmap you used in the last project.)

You may find it useful to reduce the size of main memory (in machine.h) to more quickly incur page faults. When a page fault is incurred, increment the variable stats->numPageFaults to see how many faults were incurred when Nachos exits.