CPS 536: Homework 3 **Dr. Clark**

Nachos Assignment #3: Multiprogramming and Virtual Memory
Double Assignment Worth 200 Points
Due Friday, April 4 Note: Your exam is Wednesday March 26.

The goal of this assignment is to add multiprogramming and demand paging to the Nachos system. As we saw in the last assignment, Nachos is currently able to run a single user level program using the "-x progname" command line argument.

1. The first step is to study the way user programs are executed in Nachos. You should focus on the following: progtest.cc, addrspace.cc, machine.cc, and translate.cc.

   Run nachos with the various debug statements turned on, especially "-d a". Also, practice using the debugger. This assignment will be almost impossible to do without using a debugger. From now on you should probably always run nachos from the debugger so that you can figure out what happened when an *ABORT* occurs.

2. Add the *-DVM* to the DEFINES line in your Makefile. You should work in the userprog directory. Just make sure you add only the VM definition and **do not add -DTLB**. The TLB definition turns on the Translation Lookaside Buffer support that you will not be using.

3. Remove the frame allocation and loading code from *AddrSpace()*. When an address space is created you should create a page table with all of the pages set to invalid. The pages will not become valid until they are loaded. Therefore, *Exec()* should not cause any physical memory to be allocated.

4. Add support for handling a machine page fault to *exception.cc*. A page fault is triggered when an address is referenced that falls in an unloaded page. Your page handler should determine which page is needed, allocate an available frame, and then initialize the frame with the correct data.

   You can determine the faulting address after a page fault from the machine register *BadVAddrReg*. There are three types of data to consider when loading a frame: code from the executable, initialized data from the executable, or uninitialized data (all zeros). **Warning: Some pages may contain more than one type of data.**

   To simplify things, you may assume that the initialized code and intialized data segments are consecutive on disk and in memory. This is the case with the NOFF executable format we are using. This assumption lets you combine the code and initialized data into a single segment with a start and ending address. You must still worry about the case where a page has some initialized data and some uninitialized data that should be set to all zero.

5. Prove that your system works by running a large program that allocates an address space larger than physical memory but doesn't actually use it all. An example is a program with large arrays that are never used. You might want to bump the NumPhysPages variable in *machine.h* down to around 8 for testing.

6. Implement the process management system calls.

   (a) *SpaceId Exec(char *name)* - Create a user process for the given executable file name and run that process. Return the identifier (an integer) for the created process.

   (b) *int Join(SpaceId Id)* - Wait for the process identified by Id to finish. Return the exit status from the process.

(c) *void Exit(int status)* - End the current process, returning the given status value to the parent process. The Exit function doesn't itself return a value since it never returns to the calling process. Note that Exit is called by all processes once they've completed even if you don't call it explicitly.

To do this assignment you will need to provide significant enhancements to the way an address space is allocated. In the code provided, we assume that every virtual address is the same as its physical address – this restricts us to running one user program at a time. You will generalize this to allow multiple user programs to be run concurrently.

You will need to come up with a way of allocating physical memory frames so that multiple programs can be loaded into memory at once (see *bitmap.h* for one helpful tool). The good news is that nachos already uses a page translation table. You should use these tables on a per process basis to support allocation and then make sure that the correct page table is always used.

7. I have given you some additional user level test programs for this assignment. You can copy these files from */home/rjc/pub/cps536/proj2* to your test directory. As always, you should write lots of others on your own.