CS350 **Operating Systems** **Spring 2004**

Assignment Two Requirements

This assignment requires you to add support for virtual memory management to the Nachos operating system.

1 Design Requirements

The specific requirements for this assignment are as follows:

1. Implement support for the hardware TLB. To enable the TLB in the simulated hardware, you must edit the Nachos `Makefile` and then rebuild `nachos` from scratch. (In the `Makefile` add `-DUSE_TLB` to the `DEFINES` line.) After you have enabled the TLB, the hardware will look *only* in the TLB to find address translations. The hardware will no longer look up address translations in your page tables.

   The TLB itself is an array of `TLBSize` page table entries. Your OS can manipulate (read and modify) these entries through the `tlb` member of the `Machine` class, e.g.,

   ```
   kernel->machine->tlb[i]
   ```

   refers to the `i`th entry in the TLB.

   Use only a simple FIFO algorithm for TLB replacement.

2. Provide support for *demand paging*. In particular, rather than loading all pages of the program into memory initially, load them only as required. If a page is never referenced, then it should never be loaded. This means that initially no pages of a process should be loaded into memory. The program actually begins execution without any pages in memory and pages each page in (including the first page referenced) as the result of handling page fault exceptions. Also, only write a page to the swap device if it has been modified; that is, load a page repeatedly from the executable file until the page content changes. You may assume that the executable file does not change while the program is running.

3. Provide support for larger virtual address spaces (and more concurrent processes) by implementing paging, so that all virtual pages do not need to reside in primary memory.

   You will implement paging out to NachOS simulated disk. So, the level of multiprogramming of your OS will be limited by the size of the disk, not by the size of RAM.

4. Implement two different page replacement algorithms: FIFO and another of your choosing. By default Nachos should use your new algorithm. By using a -F option on the command line, your version of Nachos should use the FIFO page replacement algorithm.

5. Keep and print statistics for the number of TLB faults, page faults, pages paged out and the dirty pages written (to disk). In order to do this you are permitted to modify `code/machine/stats.h` and `code/machine/stats.cc`.

2 Testing Requirements

Your test suite is required to do the following:

1. Demonstrate that the TLB replacement is handled properly by running a program with many TLB faults. Show that consistency between the TLB and the page tables is maintained correctly.

2. Demonstrate that demand paging is implemented. Use a program with an address space which is larger than physical memory and show that only those pages that are touched are loaded.

3. Demonstrate that a process with an address space larger than the machine's physical memory can run.

4. Demonstrate that a set of processes with a combined address space size that is larger than physical memory can run.

5. Demonstrate the behaviour of your system when it is faced with too many processes (i.e., too many processes for the memory resources available to run them.)

6. Write some test programs and run a few simple experiments that compare the number of page faults when using each of the two page replacement algorithms (FIFO and the algorithm you've chosen).

3 Additional Features

In addition to the required design given above, you must implement **one** of the following requirements. Your test suite should demonstrate that this additional requirement is functioning correctly.

In designing and implementing your extra feature you may need to add system calls. This is fine, but make sure that what you add and how to use it is well documented in your design document and in syscall.h.

- Support on-demand memory allocation, i.e., allow a process to request additional heap space for storing data. You will need to add a system call that allows processes to request additional space.

- Support automatic growth of the stack part of the process address space in the event that the stack fills the space allocated to it. You will need to decide on a way for your kernel to detect that the stack needs to grow, and on a mechanism for expanding the address space.

- Support a general shared-memory facility. For example, you could add a system call to allow two or more processes to request a chunk of shared memory of a specified size.

- Provide support for automatic sharing of read-only code pages between address spaces. That is, if the same program is loaded into more than one process' address space, any pages that hold program code should not appear more than once in physical memory.