CS354/CS350 Operating Systems Winter 2004

Assignment Two Requirements

Design and Preliminary Testing Document Due: Tuesday March 2, 11:59 am
Full Assignment Due: Tuesday March 9, 11:59 am

Be sure to obtain and use the new version of syscall.h that is provided for this assignment. It contains the definitions of the new system calls and system call numbers that will be required for this assignment.

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`. In the `Makefile` add `-DUSE_TLB` to `DEFINES` line. Then you must rebuild `nachos` from scratch. That is, do `make clean` followed by `make depend` followed by `make`. 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.

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 using the NachOS simulated disk. Therefore, the degree of multiprogramming of your OS will be limited by the size of the disk, not by the amount of available memory.

4. Implement the FIFO (first in first out) replacement algorithm for use with TLB and page replacements. Add the ability for NachOS to handle `-FIFO` option on the command line and use that to indicated that the FIFO replacement algorithm should be used.

5. Implement the enhanced second chance replacement algorithm (as described in the course notes) for both TLB and page replacements. Add the ability for NachOS to handle `-E2NDCHANCE` option on the command line and use that to indicated that the enhanced second chance algorithm will be used.

6. Implement a `-LOCKCODE` option to NachOS to specify that code pages should be paged in and should never be paged out while the program is running. Once the program is finished executing the code pages become regular candidates for paging out.

   IMPORTANT NOTE: this option will be used to test your implementation so you should ensure that it is implemented early and that it works.

7. Keep and print statistics required to implement the following system calls.
IMPORTANT NOTE: these calls will be used to test your implementation so you should ensure that they are implemented early and that they work.

   - `int GetNumPages(); // number of pages of physical memory`
   - `int GetNumTLBEntries(); // number of TLB entries`
   - `int GetPageSize(); // returns the size of a page (bytes)`
   - `int GetTLBFaults(); // TLB faults (total)`
   - `int GetTLBDataFaults(); // TLB faults non code/text pages`
   - `int GetPageFaults(); // Total page faults`
   - `int GetDataFaults(); // Page faults for non code/text pages`
   - `int GetPageOuts(); // Total page outs`
   - `int GetDataPageOuts(); // Total page outs for non code/text.`
   - `int GetDirtyPageOuts(); // Page outs of dirty pages`
   - `int GetDirtyPageIns(); // Page ins of dirty pages`

   You should ensure that your kernel works properly when the size of the TLB or the number of pages is changed as we may change these values for testing purposes. You should also try to implement some simple programs that exploit the system calls described above when testing your implementation.

8. Support a general shared-memory facility. Add a system call to allow two or more processes to request a chunk of shared memory of a specified size.

   The system call interfaces required to support this feature are:

   - `void *SharedMemOpen(char *name, unsigned int bytes);`
   - `int SharedMemClose(char *name);`

   The first process to call `SharedMemOpen` with a given name defines and sets the size of the shared memory region. A successful call returns a pointer to the new region of shared memory. An unsuccessful call simply returns a null pointer (i.e., 0) indicating failure. Subsequent calls to an already open shared memory region that use the same name but a different size fail and return 0 to indicate that the call failed.

   The `SharedMemClose` call closes the shared memory region for the calling process only. When the last process that has opened a shared memory region closes the region it is removed from the system and the name can be reused using a different region size. The `SharedMemClose` call returns 0 on success and a negative error code on failure. Possible error codes are `ENOENT` (i.e., the name is not found), `EINVAL` (i.e., trying to close a shared region the caller hasn't opened or the pointer to name is not in the address space of the caller).

9. For this assignment in addition to the initial design document you are also required to complete and hand in a two page preliminary testing strategy document in which you should be sure to discuss the following:

   (a) Look at the original program `code/test/matmult.c` (if you've modified it in previous assignments have a look at the original). Assuming that you increased the size of each matrix considerably, in your initial design document explain why this application is not a very good application to use for testing your implementation in this assignment. Also explain what could and should be done to make this a much better test program.

   (b) Explain how you plan to show that different aspects of your TLB and page replacement algorithms have been implemented correctly. In each case you should only use simple accesses to an array in order to generate data page faults. Describe how you would choose the size of and access an array

`char array[SIZE];` to generate the desired behaviour to test each algorithm. Assume that the array would be declared globally and it would therefore be part of the uninitialized data segment. Be sure to utilize the system calls described above but assume that they have been implemented correctly.

You will be permitted up to a one page addendum for your design document. You will be permitted up to two additional pages for your testing document to include extra information that wasn't included in the preliminary testing strategy, and/or corrections to the preliminary testing strategy.

2 Some Basic Testing Requirements/Hints

In addition to the testing strategies outlined above your testing is also required to do the following:

1. Ensure that the TLB replacement is handled properly by running a program with many TLB faults. Try to show that consistency between the TLB and the page tables is maintained correctly.
2. Show 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. Show that a process with an address space larger than the machine's physical memory can run.
4. Show that a set of processes with a combined address space size that is larger than physical memory can run.
5. Show 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 page replacement algorithms that you've implemented. Ensure that you implement at least one program that demonstrates a significant difference in the number of page faults.