CSC 341

Operating System Virtual Memory

Assigned 02/20/13 Due 03/20/13

Purpose:

Archectectural Design:
Machine Instruction and Operation Codes:
Machine Instruction Format:
Address Mode Flags:
Registers:
Hardware:

ASM: Unchanged - see program 1 instructions for details. The interpreter will need to change to call the MMU routine with logical addresses which will then return absolute addresses

Operating System Design

Changes will include the introduction of a simple load system and an adjustment to the semaphore or lock feature to control only specific memorylocations for each user. In terms of the components:

UI: Unchanged. However the run command should now call a loader routine that places the program in fixed size pages with the page location chosen randomly. If the page chosen is in use (keep a global bit vectore or used pages?), then the routine should continue to choose until a free page is found. It will still be "Interactive command-line". See program 1 for details.

Memory:

A 1-D array (size = 256) of short int (16 bit words). For this lab, memory will be divided into 4 word (intructions or data) sized frames. The initialization routine will need to be change to include a page table for each user. The page table will maintain the logical page to actual memory frame mapping (handled by the MMU routine). As the loader finds frames to place pages into memory it must call the MMU routine to update the page table for that user. The MMU routine will handle both initial placement calls from the loader and translation calls from the interpreter to handle memory addresses and check valid ranges. When the interpreter encounters a halt instruction in a user program, the frame contents for the programm should be displayed on the user console. A call can then be issued to the MMU to clear the page table and free the frames. All locks should work on a frame basis rather than the entire memory. Both user programs should start at 0 in the logical address space (see below). Access to memory and the MMU should be controlled by a lock or a semaphore system that protects memory from use by more than one user at a time. For the current version, the system can continue to provice a lock/semaphore queue and appropriate data structures for control outside of main memery (i.e. in the o/s initialization area). Your team might again consider how to integrate into main memory.

Disk: User 1 program should be stored at location 0 and user 2 program should be stored at location 100 on the disk. You may use these hardcoded addresses for the loader (which will be addressed later).

Clock: Unchanged from version 2.

Scheduler: Round robin: user1, user2, o/s, unchanged from version 2.

Organization: Modify the initialization routine (modify data structures, add tables, etc.) and add variables (like the memory bit vectore -256 elements 1 bit size each array). Add the loader and the MMU components. As usual, each of these routine should be written in a modular form so that additional operations can be added easily. The interpreter wll be modified to call the appropriate routines during address translation.

Data: Test interatively and hardcoded. All other data will be found in the instr directory to test your programs. The third O/S version will assume that the porgrams are located in fixed positions on the disk (done during intitializations). For this version, use the O/S run command to callt he loader to load the instructions and data into memory (4/page) and the MMU to set up the map to the physical frames. Use the program in the first version for user 1 and use the following program for user 2 in this O/S version:

ASM code	Machine Code (Hex)	Machine Code (Bin)
.org 0		
.st		
LOD I R0 #25	0819	0000100000011001
STO R0 106	1006	0001000000000110
LOD I R1 #5	0905	0000100100000101
SUR R1	5100	0101000100000000
STO R0 7	1007	0001000000000111
HLT	F000	1111000000000000
.nd		

Note: The uses should input hex code, for a program, and it can be convert to binary with I/O tools or separate routines (hex2bin, bin2hex) that you write.

Turn In: You should turn in your program via the submit command (text-based only). Also you should turn in a report including a cover page (name, team, account, date), a summary of your work including a statement of the problem, any difficulties encountered (or parts not working), and any additional observations including a detailed table of the time spent by each team member. A complete listing of a sample execution with the data provided should be included as an appendix with the account name and file listings used for the submit process.

Hints: Build your program incrementally. Design and implement your data structures. Get the operations working with simple stubs and then fill in the details. For example, the fetch functionmight initially indicate (print out) the code fetched. This could then include decode processing, etc. Focus on functionality and make sure each tema member understand the program. You may want to read teh software engineering information for guidance.