CSci 460: Operating Systems **Spring 2001, Zemoudeh**
Handout 4: Phase III – Memory Management **Due Wednesday June 6**

California State University, San Bernardino
Computer Science Department

In this phase we will add the memory management layer to our toy OS. We will implement demand-paging. Let us fix the page size at 16 words, therefore there are 16 frames in our 256 word memory. Each entry in the page table consist of the frame number and the valid/invalid bit. We will use two page replacement algorithms, FIFO and LRU. When a page-fault occurs, the offending process is placed in the wait queue with the trap completion time set to 64 time units. After the page fault has been serviced, that is 64 or more time units has passed, the process is moved to the ready queue and the new page is loaded in the memory. In addition to the information gathered in the second phase, compute Number of Page-faults and Hit Ratio for each process. Here Hit Ratio is percentage of non-page fault memory references.

We will assume the VM (CPU) has a Translation Look-aside Buffer (TLB). Without a TLB, every memory reference results in a trap to the OS, clearly an unrealistic situation. This assumption is also meant to simplify the OS; every time there is a memory reference, if it already exist in the page table, then it is handled in the hardware and it takes 4 time units, otherwise, it traps to the OS.

By the same token, the OS doesn't want to check the validity of `pc` every time it is incremented. At any time, `base` and `limit` registers point to the currently executing page, and the hardware makes sure that the value of `pc` is bound by them. Whenever `pc` goes out of bound, then an OS trap is generated.

To support LRU, add 16 registers to the hardware (VM), one per frame. Again this set of registers can be represented by a vector of integers. Each time a frame is accessed, the hardware saves the current time in its corresponding register. OS accesses these registers to perform the LRU page replacement algorithm. Note that FIFO doesn't need such registers and it can be done entirely by the OS. This is because the time a page is brought into the memory can be recorded by the OS in its own (software) vector (of registers).

Run your OS twice, each time with one of the page replacement algorithms but for the same set of programs. This way the merits of the two algorithms can be compared. Use `seekp()` to replace a page in the `.o` file which serves as the simulated disk. Note that you don't need to replace a page if its corresponding frame has not been modified. As far as stack is concerned, it takes up as many pages as necessary in high memory. Memory locations taken up by stack will not be available for paging purposes. If the stack grows too large trying to overwrite an allocated frame, that frame has to move out (and replaced on disk if

needed) before the stack can grow any larger.

Let us fix the degree of multiprogramming at 6; only 6 PCBs are in either ready queue, wait queue, or CPU at a time. When the OS starts, it assembles all the `.s` files to their corresponding `.o` files, loads in the first page of the first 6 `.o` files into memory, sets up their PCBs in ready queue, sets up their page tables, and start executing the first process. As the processes are executed, more pages are brought in based on the availability of frames and the requests of the processes. Only when a process terminates a new process is added to the system to maintain a degree of multiprogramming of 6. Note in this phase a context switch happens either when an IO operation is encountered, the time slice of the running process is over, the running process terminates, a run time error occurs, or a page fault occurs.

Add two more `.s` programs, `addVector.s` and `subVector.s`, to the list of programs from phase 2 for a total of 8 programs. These programs are found at

`http://www.csci.csusb.edu/cs460`

Since the `ls` command lists files in alphabetical order, the order in which the programs are brought in is as follows: `addVector.s`, `fact1.s`, `fact2.s`, `io.s`, `sub.s`, `subVector.s`, `sum1.s`, and `sum2.s`.

Demonstrate your program and hand in printouts of your source code, and all `.s`, `.o`, `.in`, and `.out` files for each page replacement algorithm. The same grading criteria as phase I and II holds.