FINAL EXAMINATION – MAY 2005
CSC 262 – OPERATING SYSTEMS
NICHOLAS R. HOWE

This is a closed-book exam. You may use two double-sided 8.5x11 sheets of notes.

All answers to this exam should be written in your exam booklet(s). Start with the questions that you know how to do, and try not to spend too long on any one question. Partial credit will be granted where appropriate. You will have two hours and twenty minutes. Good luck!

Vocabulary (16 points)

Define the following terms in a sentence or two. Elucidate any acronyms.

a). Busy Waiting

b). Working Set

c). Linux Kernel Module

d). Inode

e). ELF (as pertaining to Linux)

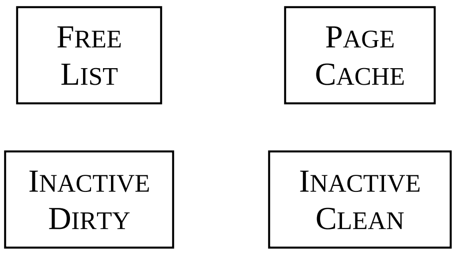
f). TLB

g). Kernel

h). File System

Linux Memory Management (12 points)

Page frames in Linux belong to one of four non-overlapping sets: the free list, the page cache, the inactive_dirty list and the inactive_clean list. Draw a diagram similar to the one below, with arrows indicating transitions that a particular page can make. Label each arrow with a description of the circumstances that would cause such a transition.



Replacement Policies (16 points)

Suppose that a process running on a paged virtual memory with four available frames has the history shown below. Give the number of the frame whose page would be evicted (or **none** if the page would already be in memory) according to each of the replacement policies that follow, for the next four requests (underlined). Additional requests are also shown, but you should only give answers for the underlined items.

Request String:	1	2	3	1	4	2	2	1	4	1	*5*	*6*	*2*	*7*	6	1	2	5	4	3
Frame 1	1	1	1	1	1	1	1	1	1	1										
Frame 2		2	2	2	2	2	2	2	2	2										
Frame 3			3	3	3	3	3	3	3	3										
Frame 4					4	4	4	4	4	4										

a). FIFO

b). LRU

c). LFU

d). OPT

Concurrent Processes (12 points)

A computer is running multiple concurrent processes. The operating system allows a number of different communication and synchronization styles. Below is show the actual sequence of interleaved actions during a particular time period. Assume that all message queues are initially empty.

P0: *mutex_init(m1);*
P0: *mutex_init(m2);*
P0: *mutex_init(m3);*
P0: *mutex_init(m4);*
P1: *lock (m1);*
P2: *lock (m2);*
P3: *lock (m3);*
P4: *lock (m4);*
P4: *asynch_send(msg,P1);*

(continued)
P4: *lock (m3);*
P3: *lock (m4);*
P1: *synch_receive(msg,P2);*
P5: *synch_receive(msg,P6);*
P6: *synch_receive(msg,P7);*
P7: *synch_receive(msg,P5);*
P1: *lock(m2);*
P8: *asynch_receive(P7);*

a). Draw the resource allocation diagram at the end of this sequence. (You may treat an unsent message as a resource that is held by the potential sender.)

b). Which set(s) of processes are deadlocked?

Virtual Memory (16 points)

On a 16-bit machine, the designers of a new operating system wish to implement a segmented memory system. They would like to provide a large number of segments, and also to provide segments of large size. This is difficult to do in only 16 bits, so they settle on a compromise. The new system will have two-level segmentation. Each 16-bit address will be split into three parts: two for the *main segment index* $\mathbf{s_1}$, six for the *secondary segment index* $\mathbf{s_2}$, and eight for the *segment offset* **w**. Alternately, $\mathbf{s_2}$ and **w** may be combined to give a *large segment offset* **w'**. A bit in the *main segment descriptor* indicates which behavior is desired.

a). What is the maximum number of large segments that can be created with this setup? What is their maximum size?

b). What is the maximum number of small segments that can be created? What is their maximum size?

c). Suppose that the system has two large segments in use at a particular point in time. How many small segments may be in use concurrently?

d). Draw a diagram showing the process of converting from a virtual address to physical address in a small segment. (This should be similar to the diagrams we drew inclass, showing the index into the appropriate tables, etc.) Give a formula for the conversion.

Working Set (12 points)

Comment on the relationship between page placement policy (i.e., local vs. global frame allocation) and the possibility for thrashing to occur in a system. In particular, state whether there are any differences in thrashing behavior to be observed under the two different policies, and how the solution to thrashing may differ. [Note: we did not explicitly discuss this in class, but given an understanding of the two concepts, you should be able to infer how they will interact.]

Process Scheduling (16points)

The Simple Simon operating system uses round-robin scheduling, and a standard quantum of 100ms. If an interrupt occurs in the middle of a process's assigned quantum, the scheduler will try to restart the interrupted process. If it cannot do so because the previously running process has blocked, then it will choose the next ready process in the queue. Preparing a different process to run requires 10 ms of system overhead. Newly created processes are added to the front of the run queue, i.e., they will be chosen to run before any previously existing process. Unblocked processes are assigned a spot in the run queue based upon the time of their last run.

Following are details about six user processes that are scheduled on the system:

Process 0: Created at time t = 0 ms. After 60 ms, will block until Process 3 has run and terminated. Will terminate after 800 ms of execution total.
Process 1: Created at time t = 100 ms. Will terminate after 300 ms of execution.
Process 2: Created at time t = 280 ms. Will block after 90 ms of execution, and not unblock.
Process 3: Created at time t = 300 ms. Will terminate after 20 ms of execution.
Process 4: Created at time t = 600 ms. Will terminate after 40 ms of execution.
Process 5: Created at time t = 650 ms. Will terminate after 120 ms of execution.

a). Draw a Gantt chart showing the processes executed over a period of one second (1000 ms).

b). Compute the CPU utilization and throughput for user processes during this period. For processes that terminate, compute the latency.