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
Describes a process that repeatedly checks the status of a blocking variable to see if it can proceed, thus using up cpu resources without accomplishing anything.

b). Working Set
The set of pages that will be needed by a process in the near future.

c). Linux Kernel Module
A piece of code that may be linked with the kernel as it is running; typically used for device drivers and other services.

d). Inode
A data structure used to keep track of a disk file under Unix. Stores all administrative details about the file except for its name.

e). ELF (as pertaining to Linux)
Executable and Linking Format. The newer format used for compiled object and executable files in Linux (as opposed to a.out format, which is older).

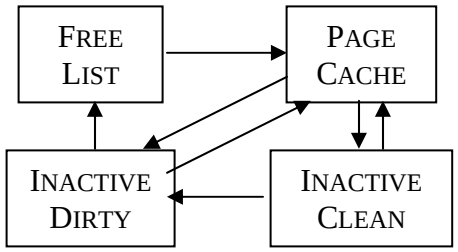
f). TLB
Translation Lookaside Buffer. A cache that holds information about recently accessed pages and/or segments to help speed up the Memory Management Unit.

g). Kernel
The most basic part of the operating system, responsible for interrupt handling and process scheduling.

h). File System
Refers to the data structures and mechanisms for organizing files on persistent storage media.

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.



Free → Page Cache: Frame allocated
Page Cache → Inactive Dirty: Dirty page unused
Page Cache → Inactive Clean: Clean page unused
Inactive Dirty → Page Cache: Request for page
Inactive Clean → Page Cache: Request for page
Inactive Dirty → Inactive Clean: Written to disk
Inactive Clean → Free: Frame recovered by OS

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: *1, 2, 3, 4*

b). LRU: *3, 2, 4, 1*

c). LFU: *3, 3, none, 3*

d). OPT: *3, 4, none, 3*

Concurrent Processes (12 points)

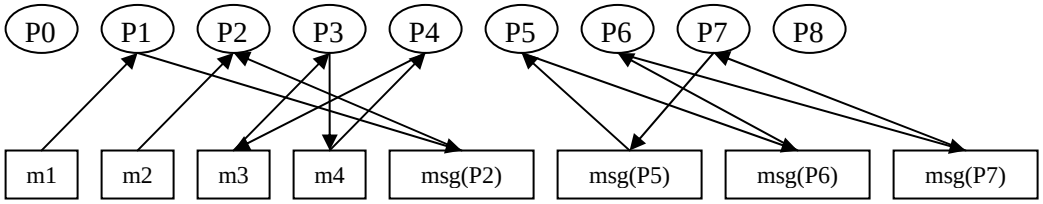
A computer is running multiple concurrent processes. The operating system allows a number of different communication and synchronization styles. Below is show the actual

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?

P3 and P4 are deadlocked, as are P5, P6, and P7.

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.

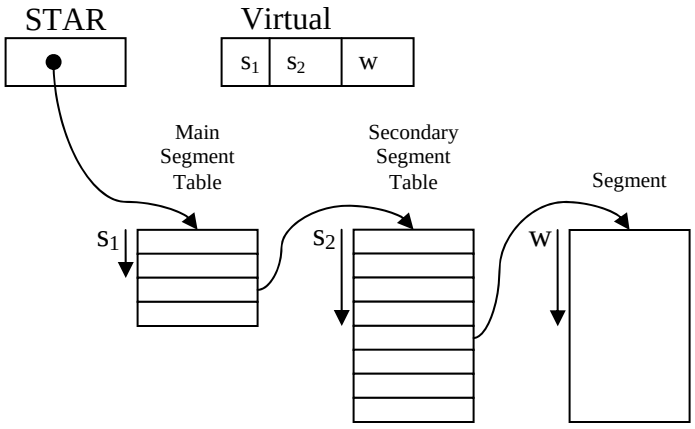
a). What is the maximum number of large segments that can be created with this setup? What is their maximum size?
Four segments of size 2^{14} = 16384 B = 16 KB

b). What is the maximum number of small segments that can be created? What is their maximum size?
256 segments of size 256 B.

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?

128 small segments.

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.]

With local frame allocation, each process is granted a fixed number of frames by the operating system. If that number of frames is insufficient to hold the working set of the process, then that process will thrash, but other processes will be unaffected. Thus thrashing may be contained to a subset of the processes running. To prevent it, the operating system must actively watch for thrashing processes, and include mechanisms to increase their allotment of frames when thrashing occurs. With global frame allocation, processes can steal frames from one another. Thus an individual process that needs more frames for its working set can take them from another process, automatically correcting the deficiency. Thrashing will only occur if the total memory available is insufficient to store the working sets of all running processes. If this occurs, then all processes will thrash at once. The solution is to temporarily block some processes from running, until the total size of the working sets of the remaining processes will fit in memory.

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 for such a context switch requires 10 ms of system overhead before the new process can be run. 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).

Time (ms)	**Activity**
0	Context Switch
10	Process 0
70	Idle
100	Context Switch
110	Process 1
210	Process 1
310	Context Switch
320	Process 3
340	Context Switch
350	Process 2
440	Context Switch
450	Process 0
550	Context Switch
560	Process 1
660	Context Switch
670	Process 5
770	Context Switch
780	Process 4
820	Context Switch

780	Process 4
820	Context Switch

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