CSE 121: Operating Systems - Architecture and Implementation
Homework 3
Fall 2003

Due: Wednesday, November 26

1 Swapping and Paging [10 points]

a) Give 2 reasons why having fixed-sized memory partitions isn't good for UNIX.

b) In Figure 1 of *Babaoglu and Joy's* paper [**BJ81**], give a specific reason why the page faults (pages requested) occur in bursts (seen as spikes in graph). Also, rather than try and change the bursty nature of page faults, Babaoglu and Joy decided to implement a mechanism to tolerate this behavior. What was this mechanism?

2 Global Memory Management [10 points]

a) What is the purpose of the *MinAge* parameter in *GMS* [**FMP+95**]?

b) In *GMS*, the initiator selects the node with the most idle pages to be the initiator in the following epoch. Explain why.

c) Microprocessors provide reference bits to help Operating Systems approximate LRU. Assuming there are enough reference bits available to calculate a per-host LRU, is this enough information to determine a global LRU? If yes, explain the how (i.e. how the *initiator node* uses this information) or why it cannot.

3 Prefetching and Caching [10 points]

Below is a file access stream for a system using a file cache, and an optimal page replacement algorithm with no prefetching, and a file access stream for a similar system that uses an aggressive prefetching policy in addition to a file cache and an optimal page replacement algorithm. The systems have 2 cache frames and each memory access for a page in the cache takes one time unit, and each fetch for a page not in the cache takes three time units.

The file access stream for the reference string $ABCB$ for the system without prefetching requires 7 time units to complete:

No Prefetching

time	1	2	3	4	5	6	7
access	A	B				C	B
fetch			C	*	*		
cache Frame1	A	A	A	X	X	C	C
Frame2	B	B	B	B	B	B	B

The same reference string using an aggressive prefetching policy (assuming the reference string is known ahead of time) requires only 6 time units to complete:

Aggressive Prefetching

time	1	2	3	4	5	6
access	A	B			C	B
fetch		C	*	*		
cache Frame1	A	A	X	X	C	C
Frame2	B	B	B	B	B	B

a) Still assuming optimal replacement, how many time units will the reference string *ABCA* require to complete without prefetching?

No Prefetching

time	1	2	...
access	A	B	...
fetch			...
cache Frame1	A	A	A...
Frame2	B	B	B...

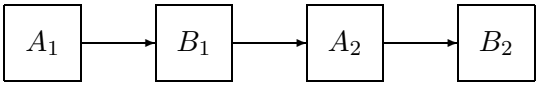
b) How many time units will the reference string *ABCA* require to complete with aggressive prefetching? Optimal replacement is still in use, but we cannot replacement a cache frame entry that is being accessed in the same time period that the prefetching begins (in the above example of aggressive prefetching, we would not have been able to replace cache frame 2 between time 2 and 3 because frame 2 contained B, which was being referenced at time 2.

Aggressive Prefetching

time	1	2	...
access	A	B	...
fetch		?	...
cache Frame1	A	A	X...
Frame2	B	B	B...

c) Does the aggressive prefetching policy violate the controlled-aggressive prefetching policy in the paper **[CFKL96]**? Explain.

d) The LRU-list of a kernel is depicted below, where the frames A_i belong to process A, and the frames B_i belong to process B. (Note: The left end of the list corresponds to the item that has been the least recently used)



Suppose process A chooses to use its own page replacement policy of most-recently used (MRU, the opposite of LRU), whereas process B will use the system's default policy of LRU. For the purposes of this question, when a page is referenced and isn't in memory, and it isn't in a placeholder, it is appended to the (right) end of the list. When it is referenced and it is is memory, it is also moved to the right end of the list.

- Illustrate the state of the kernel's LRU-list (including placeholders) using the LRU-SP policy mentioned in the paper at the end of **each** of the following page references for the page reference string A_3, B_1, A_4, B_1, A_2, B_2. Also indicate how many page faults there will be for process A and B.

- Repeat the above without using any placeholders (i.e. do not create any placeholders).

4 Microkernels [10 points]

Any demand paging system will, at some point, use a driver to access the paging device (like a disk). In later versions of Mach, paging from a local disk references a kernel-loaded disk driver rather than a user-level disk driver. L4, however, has all drivers run at user level.

Suppose that the disk driver is buggy.

a) What kind of memory corruption would be prevented by running the driver as a L4 user-level driver as compared to an kernel-loaded driver?

b) What kind of memory corruption could not be prevented by running the driver as a L4 user-level driver?

c) What is the extra overhead that is levied by running the driver at user level rather than in the kernel?