COLBY COLLEGE
Department of Mathematics
and Computer Science

CS357 **Allen B. Downey**
Spring 1998

Assignment 6: Paging and thrashing

Due: Tuesday 28 April

The purpose of this assignment is to test out the paging mechanism on the irises.

Reading

Read Silberschatz and Galvin, Chapters 8 and 9, with emphasis on Section 8.5 and 9.3.

Warm-up exercises

The real purpose of these exercises is to make sure you get the basic mechanisms we are studying. You can do them in groups along with the experimental part of the assignment, but take special care that you are not riding a partner's coat-tails. These are exactly the sort of questions I am likely to put on the final exam.

Also, don't you think we are overdue for a surprise, unannounced pop quiz?

1. Write up a solution to Exercises 8.8 and 8.10 on page 285. In the second part of 8.10, be aware that by "associative registers," they mean the TLB. They have a point, I suppose; TLB is a stupid name.
2. Write up a solution to Exercise 9.5 on page 331. Once again, be aware that the "registers" in the second sentence are in the TLB. Also, do enough of Exercise 9.11 to convince yourself that you know how page replacement policies work. You don't need to turn anything in, though.

Check out the IRIX pager

1. Modify the cache-testing program from Assignment 1 so that it tests the paging mechanism. Try to find out how much memory you can allocate before you start paging and how big the pages are.

   You will have to be careful in your design of this experiment, since it will be difficult to strike a balance between using enough memory to exercise the paging mechanism and using so much memory that the program takes forever to complete.
2. When you finish running the page-testing program, what do you notice about the behavior of the system? Can you explain your observations?

Thrash

1. Experiment with multiple processes and try to infer whether IRIX is enforcing a limit on the amount of memory a single process can allocate, or whether processes can steal frames from each other without restriction.

2. Run two programs at the same time and try to get them to thrash. Either program alone should terminate in a reasonable amount of time, and use at least a little CPU (like 10% utilization). When you run the two programs at the same time, though, they should both run *much* longer and their CPU utilizations should go down.

   In order to get this part to work, you might have to fiddle with priorities. If you are having trouble, let me know and I will make some suggestions.

Usual rant about reports omitted.