This is the first “paper” homework assignment. All the work you submit should be written or printed on paper and can be handed in at the beginning of lecture on the due date. Late homework submissions will be accepted by the beginning of the next lecture but will suffer a 10% lateness penalty.

Reading: In AOS: finish chapter 4 on Processes; some of this homework is based on section (4.4). Read chapter 5 on Threads; some homework includes running and modifying a simple Java program with threads. Read chapter 6 on Scheduling up through section (6.2).

In Bovet: In Chapter 2 on Memory Addressing, read the whole section Paging in Hardware, up through the subsection on Translation Lookaside Buffers. Answer for yourself: Why is the translation lookaside buffer essential to make paging practical?

In Haviland: Read Chapter 5. Get a “leg up on” CSI402 by writing and debugging your own Unix shell fashioned after Haviland’s “smallsh” and then adding to it cool features of your own choice. (This is not an Operating Systems project because the shell is NOT part of the kernel. Some people and vendors say the shell is part of the Operating System but I and others prefer to say it belongs to the Operating Environment.) You can defer details of shell writing for a couple of weeks, but please figure out how and why Unix shells (1) scan command lines, (2) use fork() and exec family system calls to create new processes and (3) use wait() to determine when to scan again.

1 Question 1

1. On an ITS Solaris remote access system which you access via ssh or telnet to “itsunix.albany.edu”, run the command “ps -Afl | less”, then find out the significance of the one-letter process state codes by finding that info in the output of “man ps”. Then do ps -Afl again now that you know the significance of some of the columns.

2. On the Solaris machine, do “cd /proc” with your shell and explore (to the extent your user privileges allow) the /proc directory and subdirectories.

3. Go to LC-4, log on to one of the UltraSparc Solaris workstations, and repeat the above exercise. Briefly compare, in writing, your observations on the remote access system with your observations on the personal system. (“Personal” here means merely “intended to be used by one person at a time”.)

4. Repeat the above exercise on a Linux system if you have access to one. If not, see me. You can view “data” in various files under “/proc” with the cat command. Find and write about “proc” files that report information about the operating system and cpu version, and what filesystems the kernel supports. Also, explore and write some guesses about what is in “/proc/sys” and “/proc/net”.

5. On a Windows (95 to XP) system on which you have suitable access, start some application programs and press control-alt-delete. You will observe some of the process list. Similarly, run the “regedit” command which can be found in the C:\Windows directory. DO NOT change any keys or values (unless you know what you are doing), but it is safe to click on the left hand pane to expand the registry hierarchy and observe keys and their values on the right hand frame. Expand and explore the directory HKEY_DYN_DATA under “My Computer”. This hierarchy in the Windows registry is the analog of the “/proc” filesystem on modern Unix variants. Write a brief report of what you find. See me if you don’t have access to a machine you can do this on.

2 Question 2

Draw with pencil or graphics software two neat sequence diagrams like Figure 4.3 in AOS to illustrate each history:

1. One process makes a non-blocking system call to the operating system. The operating system returns directly to that process after handling the system call.
2. One process makes a blocking system call.
   The OS runs a second (ready) process.
   This second process is interrupted by the event caused by the I/O device (draw an activity line for the
   I/O device) where this event enables the blocked system call to complete. (You must show a column
   for the I/O device and indicate when the I/O device sends an interrupt that interrupts the second
   process.)
   The OS reschedules the interrupted 2nd process because it has high priority.
   After a timer (You must draw an activity line that shows the timer is active all the time.) interrupt, the
   OS makes the second process be idle (in the ready state).
   The OS immediately makes the previously blocked system call return to the original process.

I explained this on the blackboard, and your job is to draw it neatly and correct any mistakes or
ambiguities! You must supply all the labels to indicate that you understand what is happening.

3 Question 3

This is based on the Java programs quoted in AOS figures 4.9, 4.10 and 4.11 which implement a shared
memory bounded buffer solution to the Producer-Consumer problems. The full code is available from
~acsi400/Hw1/Java on itsunix, from
http://www.cs.albany.edu/~sdc/CSI400/Lectures/L06
and from links on the course home page. Its use is demonstrated by the following Java application top level
“main” method:

/**
 * Server.java
 *
 * This creates the buffer and the producer and consumer threads.
 *
 * @author Greg Gagne, Peter Galvin, Avi Silberschatz
 * @version 1.0 - July 15, 1999
 * Copyright 2000 by Greg Gagne, Peter Galvin, Avi Silberschatz
 */

public class Server
{
    public static void main(String args[]) {
        BoundedBuffer server = new BoundedBuffer();

        // now create the producer and consumer threads
        Producer producerThread = new Producer(server);
        Consumer consumerThread = new Consumer(server);

        producerThread.start();
        consumerThread.start();
    }
}

First copy the contents of ~acsi400/Hw1/Java on itsunix into an empty subdirectory, read and use
the Makefile to build and run the software. Recall that this program was partially explained in class and
diagrams of its data structures are posted on the Web. Observe that the particular sequence of statements
that computer executes depend on (1) which positive integer is picked for each calculation of a random
integer; and (2) decisions made by the scheduler for when to switch from thread to thread.
1. Suppose (for this exercise) that the Java thread scheduler is non-preemptive (see AOS section (6.1.3) on preemptive scheduling and/or our class notes). Under this assumption and the given program code, the scheduler will run another thread only when the current thread calls `sleep(n)` with a non-zero number of seconds to sleep $n$ value. Give a written explanation why, in terms of the definition of non-preemptive scheduling and what event each waiting thread will wait for. Explain why the whole application software's execution will “hang” if any thread enters any of the “// do nothing” while loops.

2. If you choose the integers that determine how many seconds to sleep instead of letting those integers be random, you can predict exactly what the program will do, assuming the scheduling is non-preemptive. Figure out a sequence of such integers which make the program behave as follows:
   - First, the buffer fills up.
   - Next, the buffer is emptied.
   - Third, one more item is put in the buffer.
   - Fourth, that 4th item is removed.
   - Finally, the program hangs with the buffer empty.
   Write a second-by-second description of what how the program executes which describe which thread does what, what integer you choose in place of each random integer generation, how the data structure variables change, what is printed, etc., to demonstrate your sequence of choices.

4 Question 4

Modify some of the Java code so it gives more informative reports of what it is doing. Print copies of your modified code and some interesting output you got from running it. (Note that the Java scheduler is preemptive, which makes understanding what it does harder.)

5 Future

Some kind of similar simulation with preemptive scheduling.