Midterm exam
Thursday, October 28, 2010  5:36 PM

Midterm review
  scope -- lectures, assignments, exercises, examples
  strategies -- two pass process, my limits, etc.
  samples

Some caveats
  My exams are difficult.
  Always curved.
  Professionals (you know who you are) are not included in calculating curves.
Structure of midterm
Tuesday, November 4, 2014  7:32 PM

Looks like a loooong exercise.
10 answers, 10 points per answer, points do not indicate difficulty.
Roughly 1/3 relatively straightforward questions.
Roughly 1/3 intermediate difficulty questions.
Roughly 1/3 "unfair:)" questions.
2-pass process

Do problems you understand first.
Then do harder ones.
Budget time to check your answers for whether I am being sneaky.
Some strategic hints for open book exams

Index your notes -- with postits -- so you can find things you need quickly. Review the examples -- I will not veer far away from them.
It's important to outline

Tuesday, November 4, 2014    7:43 PM

I suggest that you outline the answer first, and then write it nicely if you have more time. You will definitely get partial credit for the outline, but no credit for a blank.
The key to dup
a) close the descriptor that you need to duplicate into.
b) dup an existing descriptor into that.
   a. a pipe
   b. a file
   c. a socket
c) watch out for open and unused descriptors.

int fd = open("file.txt", O_RDONLY);
close(fileno(stdin));
// does not close stdin, just fileno(stdin)
dup(fd); // copies fd into stdin.
// after this, stdin is a buffer for fd.
close(fd); // does not close stdin.

Generalizing:
   Dup occurs before a fork()/exec()
   The exec inherits the open file descriptors,
   but not the file pointers.

Example: error duping.

Suppose we want to point both stderr and stdout to a file "foo.txt".

int fd = open("foo.txt", O_WRONLY);
close(fileno(stdout));
close(fileno(stderr));
dup(fd); // dup stdout
dup(fd); // dup stderr

printf("ho there"); before dup: keep in buffer.
printf("ho there\n"); before dup: buffer is flushed before the dup -> not in foo.txt.
Producer-consumer

a) the bounded buffer algorithm
   a. block on read (when queue empty)
   b. block on write (when queue full)
   c. implementation: ring queue of characters. with mutexes.
   d. can use semaphores to track sizes of buffer.
   e. can use semop to update semaphores.

Note that the basic bounded buffer algorithm uses two mutexes that determine whether the queue is empty or full
   can_read: unlocked if there is data, locked if no data.
   can_write: locked if queue is full, unlocked if there is space.

If you try to enqueue, and can_write is locked, you block. --> you go out of the run queue, you are queued for re-running after can_write is unlocked, and you wait for can_write to be unlocked.
Assumptions:
processes ask for what they need.
If they get what they need, they end and release resources.
processes do not collude or lie about requirements.

A situation is safe if there is a completion schedule whereby all processes complete according to the above assumptions. Unsafe otherwise.

Banker's tableau
121 * 2 0
P2 2 3
P3 3 2

\[
\begin{array}{cc}
0 & 1 0 \\
1 & 1 \\
3 & 2 \\
\end{array}
\]

\[
\text{TOTAL}
\]

\[
\begin{array}{cc}
R_1 & R_2 \\
S & Y \\
\end{array}
\]

Suppose P3 asks for \( \frac{1}{4} \) more. So? R2
Difference between semop and sem_wait

The main difference is that semop can modify multiple semaphores in parallel and atomically:

a) either all semaphores are modified, or none of them are.
b) semop can combine wait and post steps.
c) sem_wait and sem_post are separate and uncoupled. You must synthesize semop behavior with logic.
The 5-state model of a process
Running and runnable processes and the run queue.
Basic execution model: segments, etc.
Isolation: processes seem to run in private memory spaces.
Kernel mode: system calls require a context switch to a different memory map.