Midterm Review

ICS332

Operating Systems

Henri Casanova (henric@hawaii.edu)

What to Expect

- Closed notes/computer/phone
- Scope: All modules up to and including Synchronization
- Material to review:
  - Quiz solutions
  - Homework solutions
  - Lecture notes
  - Reading assignments in the textbook
    - Especially the examples

What Questions to Expect

- Some quiz-like
- Some short “how?” or “why?” questions
  - Answer in a few sentences
- Study for the above by going through the material pretending you’re a professor who has to come up with a bunch of quiz / short questions
- Some may have to do with what you did in the homework assignments
- They will be no “write a program” question
  - At most 1-2 lines of code to insert/fix in a given program

- Questions like the homework assignments
- There should be very few surprises

Example Short Question

- What’s the difference between an Interrupt and a Trap? Give an example of each

Example Short Question

- What’s the difference between an Interrupt and a Trap? Give an example of each
  - An interrupt is dan external event
    - Example: keyboard input, disk operation completion, mouse click, network packet arrival, etc.
  - A trap is an internal event caused by an instruction
    - Example: divide by zero, illegal memory access, system call instruction

Example Short Question

- What memory reads/writes happen when a context-switch occurs?
  - Say we context switch from process A to process B
  - A's registers are written into its PCB
  - B's registers are read from its PCB

Example Short Question

- What's a Zombie?
- What's an Orphan?

Example Short Question

- What's a Zombie?
  - A process that has terminated (i.e., called exit()), but whose parent has not acknowledged the death (i.e., not call to wait(), waitpid()).
  - It is kept around so that later the parent can find out its return value
- What's an Orphan?
  - A process whose parent has died
  - In Linux it is adopted by process PID=1 (meaning that its PPID=1)

Example Short Questions

- What is priority inversion?
- Why should context-switching overhead be low?
- What happens if at the end of a time quantum of a running process the ready queue is empty?
- A large time quantum value in a round-robin scheduler will be preferred by what kind of processes?
- And so on….

Longer Questions

- Longer questions will ressemble homework and in-class exercises / examples
  - Given a program with calls to fork(), exec(), wait(), dup(), close(), etc. what is wrong / what does it do?
  - Given a schedule what can you say about the scheduling algorithm?
  - Given a set of jobs with CPU and I/O burst times, what do various scheduling algorithms do?
  - Given a multi-threaded programs, what does it do? What's wrong with it?
    - How would you add locks?
    - Is there a deadlock?

Synchronization Questions

- We did not have a homework in the Synchronization module
- So let’s review this right now
- Main concepts
  - Race conditions
  - Locks
    - Spinning vs. Blocking
  - Deadlocks

Race Condition

- Two threads, one global variable a = 0;
- Thread #1 does: a+=2;
- Thread #2 does: a-=1;
- What are the possible final values of a?

Race Condition

- Two threads, one global variable a = 0;
- Thread #1 does: a+=2;
- Thread #2 does: a-=1;
- What are the possible final values of a?

- The “clean” execution is a = 1
- Each thread could have a lost update, which would lead to -1 or 2
- Answer: {-1, 1, 2}

Locks

- Consider the following code fragment:

```
int a = 0;

void f(int x) {
  x++;
  a  = a + x;
}
```

- Where would you add lock() and unlock() calls so that multiple threads can safely call this function “at the same time”, while making the critical section as short as possible?

Locks

- Consider the following code fragment:

```
int a = 0;

void f(int x) {
  x++;
  lock();
  a  = a + x;
  unlock();
}
```

- Where would you add lock() and unlock() calls so that multiple threads can safely call this function “at the same time”, while making the critical section as short as possible?

Lock: Spinning vs. Blocking

- Spinning:
  - burn CPU cycles checking the lock continuously, which is wasteful
  - but as soon as the lock is released by whoever had it you grab it, which is good
- Blocking lock:
  - Go to "sleep" asking for somebody to wake you up when the key's ready, which avoids being a useless CPU hog
  - But this requires much more work as the OS is now involved to put you to sleep and wake you up
    - Moving your PCB from various queues, etc.
- Rules of thumb:
  - Spinning for a long time is wasteful (wasted CPU)
  - Blocking for a short time is wasteful (high overhead)

Deadlocks

- Make sure you understand resource allocation graphs
  - If the ”boxes” have more than ”one dot”: if there is a cycle, there may be a deadlock
  - If the ”gray boxes” have only ”one dot”: if there is a cycle, there is a deadlock
- Should we do an example?

- Remember the in-class exercise with 9 locks and 2 threads?

Questions?

- Any past homework assignments or in-class examples that we should look at?
  - Perhaps examples in the IPC lecture notes for processes (output redirection, dup(), etc.)
  - The scheduling assignment?

- I have yet-another-fork example after this slide that we can look at…

Another fork() Example

```
int a = 10;
int p1, p2;

p1 = fork();
if (p1 != 0) {
    a++;
    sleep(200);
    p2 = fork();
    if (p2 == 0) {
      sleep(300);
      a++;
    }
    printf("%d\n",a);
} else {
    printf("%d\n",a);
}
```

- What does this code print?

Another fork() Example

```
int a = 10;
int p1, p2;

p1 = fork();
if (p1 != 0) {
    a++;
    sleep(200);
    p2 = fork();
    if (p2 == 0) {
      sleep(300);
      a++;
    }
    printf("%d\n",a);
} else {
    printf("%d\n",a);
}
```

- The output is 10, 11, 12