Operating Systems: Midterm Review

CSC-4320/6320 –Summer 2014

Dustin Kempton (dkempton1@cs.gsu.edu)

What to Study?

- Slides, notes, and reading assignments up to and including “Deadlocks”
- Past homework assignments
- Past programming assignments
- There should be very few surprises
- Type of Questions:
  - general “what do you think?” questions
  - homework-like problems
- Exam will be available for 24 hours starting 6:00 p.m. Thursday.
  - You are expected to work alone
  - You may use your book, notes, google
    - I am more interested in seeing that you can find the answers and have an understanding of the topic than I am in your ability to memorize.

Sample General Questions

- What is the difference between an Interrupt and an Exception/Trap? Give two examples of each

Sample General Questions

- What is the difference between an Interrupt and an Exception/Trap? Give two examples of each
  - Both are events that the OS must react to
  - An Interrupt is due to some external event
    - e.g., keyboard input
    - e.g., disk operation completion
  - A trap/exception is caused by an instruction
    - e.g., illegal memory access
    - e.g., system call instruction

Sample General Question

- What are the advantages/disadvantages of User vs. Kernel Threads?

Sample General Question

- What are the advantages/disadvantages of User vs. Kernel Threads?
  - User-level
    - good: very low overhead for creation/maintenance because the kernel’s not involved
    - bad: cannot take advantage of multi-core architectures
    - bad: if one thread blocks, they all block
  - Kernel-level
    - bad: high overhead because the kernel is involved
    - removes the two “bad” of user threads

Sample General Question

- Explain what happens during a context-switch

Sample General Question

- Explain what happens during a context-switch
  - Assume A is running and we are switching to B because A's time quantum has expired
  - Save A's "state" (registers) into its PCB structure in the kernel
  - Move A's PCB to the "Ready" queue
  - Restore B's "state" (registers) into the CPU by reading it from B's PCB
  - Move B's PCB to the "Running" queue
  - Tell the CPU to start a new fetch-decode-execute cycle
    - The PC register contains the address of the next instruction that B must execute

Sample General Questions

- What is a zombie?
- What is an orphan?

Sample General Questions

- What is a zombie?
  - A process that has terminated (i.e., called “exit()”), but whose parent has not acknowledged the death (i.e., not called wait() or waitpid()).
  - It is kept around so that later the parent can find out its return value (e.g., exit(42));
- What is an orphan?
  - A process whose parent has died
  - It is adopted by process PID=1 (meaning that its PPID=1)

Sample General Question

- 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?
- etc. etc.

Sample General Questions

- There may be a general question about the programming assignments
- Just to make sure that you've done them and understood concepts in there
- Example: What system program is used to trace system calls on a Linux system? strace
- Example: What can be done to force the init process to adopt a child process you create? Create a child process that creates a child and then calls exit, thus the grandchild will be adopted by init.

Preparing for General Questions

- Most of the lectures consist of topics that beg for a “how does this work?” question
- Each such topic corresponds to an OS concept
  - There are not that many such concepts in each lecture
- Go through general questions in homework assignments, questions in the chapter practice exercises may be a good place to review your understanding of the chapter

Problems

- All problems will resemble homework problems with maybe a few more creative questions added
  - Given a program with calls to fork(), what happens?
  - Given a set of jobs with arrival time, burst time, and priority, what do various scheduling algorithms do?
  - Given a multi-threaded program, what are the possible outputs?
  - Given a multi-threaded program, how would you add locks?
  - Given some scenario, is there a possible deadlock?

Problems

- Some problems will be: “look at that code, and tell me what it does”
  - either pseudo-code or C++/Java code

Concurrency

- Since we didn’t have a concurrency Homework assignment, we can do a few “similar” concurrency examples..
- Spinning locks vs. blocking locks
- Interleaved output
- Locks
- Deadlocks

Spinning vs. Blocking Locks

- 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 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
  - Blocking if the lock is never taken for a long time is wasteful

Interleaved Output

- Two threads, two global variables a=1 and b=1
- Thread #1: b++; a=b-a
- Thread #2: a++
- What are the possible outputs?
- 3 “clean” interleavings:
  - a++; b++; a=b-a (0,2)
  - b++; a++; a=b-a (0,2)
  - b++; a=b-a; a++ (2,2)
- There can also be bad interleaving of the non-atomic “a++” and “a=b-a” “instructions” since they both write to the same variable
- These can happen only after b++ has executed and then we’re left with both “a++” and “a=b-a” to execute

Interleaved Output

- What are the “bad” interleavings for “a++” and “a=b-a”?
  - Remember that at this point b=2
- **Case #1**:
  - “a++” load the value of a, which is 1, computes2, and is about to write back to a, but there is a context switch
  - “a=b-a” loads the value of b (2) and of a(1), computes b-a=1, and is about to write back to a, but there is a context switch
  - “a++” resumes and writes back value 2 to a
  - “a=b-a” resumes and overwrites value 2 by value 1
  - So in the end we have (1,2)

Interleaved Output

- What are the “bad” interleavings for “a++” and “a=b-a”?
  - Remember that at this point b=2
- Case #2
  - “a=b-a” loads the value of b(2) and of a (1), computes b-a=1, and is about to write back to a, but there is a context switch
  - “a++” loads the value of a, which is 1, computes 2, and is about to write back to a, but there is a context switch
  - “a=b-a” resumes and writes back value 1 to a
  - “a++” resumes and overwrites value 1 by value 2
  - So in the end we have (2,2)

Interleaved Output

- In the end we found these possible outputs:
  - Clean #1: (0,2)
  - Clean #2: (0,2)
  - Clean #3: (2,2)
  - Unclean #1: (1,2)
  - Unclean #2: (2,2)
- So, in total, we have 3 possible outputs: (0,2), (2,2), and (1,2)

Locks

- When writing concurrent code in which multiple threads update the same (global) variables, one must make sure to avoid all “unclean” interleaving
- For this purpose we create critical sections
- Each critical section (which can have multiple zones in the code!) is defined by a lock, with a lock(A)/unlock(A) pair of statments

Locks

- Say our code has the following statements and that many threads will execute these statements

  ```
  ….
  x++;
  …
  y++;
  x++;
  …
  z=2;
  …
  ```

Locks

- Say our code has the following statements and that many threads will execute these statements

```
….
x++;
…
y++;
x++;
…
z=2;
…
```

- A brute force approach is to use one lock

```
…
lock(A);
x++;
unlock(A);
…
lock(A);
y++;
x++;
unlock(A);
…
lock(A);
z=2;
unlock(A);
…
```

Locks

- Say our code has the following statements and that many threads will execute these statements

```
….
x++;
…
y++;
x++;
…
z=2;
…
```

- A brute force approach is to use one lock

```
…
lock(A);
x++;
unlock(A);
…
lock(A);
z=2;
unlock(A);
…
```

NOT good because a thread cannot do x++ while another one is doing y++.
Loss of concurrency, and thus performance.
If you want everything to be sequential, why use threads in the first place

Locks

- Say our code has the following statements and that many threads will execute these statements

```
....
x++;
…
y++;
x++;
…
z=2;
…
```

- First Question to answer, how many locks?
- Should there be a lock for protecting updates of x?
- Should there be a lock for protecting updates of y?
- Should there be a lock for protecting updates of z?

Locks

- Say our code has the following statements and that many threads will execute these statements
  ```
  ….
  x++;
  …
  y++;
  x++;
  …
  z=2;
  …
  ```

- Solution:
  ```
  …
  lock(A);
  x++;
  unlock(A);
  …
  lock(B);
  y++;
  unlock(B);
  lock(A);
  x++;
  unlock(A);
  …
  lock(A);
  z=2;
  unlock(A);
  …
  ```

Deadlocks

- We haven’t seen much on this topic
- Make sure you understand the resource allocation graphs
  - If the “gray 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