Administrivia

Slide 1

- Reminder: Homework 2 due today. Please turn in hardcopy for written problems if possible.
- Midterm *next* Wednesday.
- Homework 3 will be on the Web earlyish tomorrow. (I will send mail.) Due next Monday. Written part not accepted late.

Minute Essay From Last Lecture

Slide 2

- Barriers? (Next slide.)
- Which method is best / easiest to implement? (Guess what the answer's going to be.) Is there one that all o/s's implement? (I'm not sure, but my guess is semaphores.)
- Why in the ring-of-servers version of mutual exclusion with message passing do you need servers? why not just do this in the client?

Slide 3

Yet Another Synchronization Mechanism — Barriers

- Sort of what the name sounds like — a *barrier* is something that enforces the rule "no process (in a group) can go beyond this point until all processes have arrived".
- May not be enough by itself to solve all/most interesting problems. Typically included as part of a more-inclusive mechanism (e.g., as part of the MPI message-passing library).

Slide 4

Classical IPC Problems

- Literature (and textbooks) on operating systems talk about "classical problems" of interprocess communication.
- Idea — each is an abstract/simplified version of problems o/s designers actually need to solve. Also a good way to compare ease-of-use of various synchronization mechanisms.
- Examples so far — mutual exclusion, bounded buffer.
- Other examples sometimes described in silly anthropomorphic terms, but underlying problem is a simplified version of something "real".

Slide 5

Dining Philosophers Problem

- Scenario (originally proposed by Dijkstra, 1972):
  - Five philosophers sitting around a table, each alternating between thinking and eating.
  - Between every pair of philosophers, a fork; philosopher must have two forks to eat.
  - So, neighbors can't eat at the same time, but non-neighbors can.
- Why is this interesting or important? It's a simple example of something more complex than mutual exclusion — multiple shared resources (forks), processes (philosophers) must obtain two resources together. (Why five? smallest number that's "interesting".)

Slide 6

Dining Philosophers — Naive Solution

- Naive approach — we have five mutual-exclusion problems to solve (one per fork), so just solve them.
- Does this work? No — deadlock possible.

Slide 7

Dining Philosophers — Simple Solution

- Another approach — just use a solution to the mutual exclusion problem to let only one philosopher at a time eat.
- Does this work? Well, it “works” w.r.t. meeting safety condition and no deadlock, but it’s too restrictive.

Slide 8

Dining Philosophers — Dijkstra Solution

- Another approach — use shared variables to track state of philosophers and semaphores to synchronize.
- I.e., variables are
  - Array of five `state` variables (`states[5]`), possible values `thinking`, `hungry`, `eating`. Initially all `thinking`.
  - Semaphore `mutex`, initial value 1, to enforce one-at-a-time access to `states`.
  - Array of five semaphores `self[5]`, initial values 0, to allow us to make philosophers wait.
- And then the code is somewhat complex . . .

Slide 9

Dining Philosophers — Code

- Shared variables as on previous slide.

Pseudocode for philosopher i:

```
while (true) {
    think();
    down(mutex);
    state[i] = hungry;
    test(i);
    up(mutex);
    down(self[i]);
    eat();
    down(mutex);
    state[i] = thinking;
    test(right(i));
    test(left(i));
    up(mutex);
}
```

Pseudocode for function:

```
void test(i)
{
    if ((state[left(i)] != eating) &&
            (state[right(i)] != eating) &&
            (state[i] == hungry))
    {
        state[i] = eating;
        up(self[i]);
    }
}
```

Slide 10

Dining Philosophers — Dijkstra Solution Works?

- Could there be problems with access to shared `state` variables?
- Do we guarantee that neighbors don't eat at the same time?
- Do we allow non-neighbors to eat at the same time?
- Could we deadlock?
- Does a hungry philosopher always get to eat eventually?
- (To be continued . . .)

Minute Essay

- None — quiz.

Slide 11