More concurrency and deadlock

2/17/25

Administrivia

- HW 6 (finishing cache simulator) due Friday night

Concurrency so far

- Race conditions vs deadlock
- Producer-consumer problem
- Locks and semaphores

High-level primitive: Monitors

- Group of functions such that only one can run at a time

- Example: Java's synchronized methods
  - acquires lock on object before entering method

Another concurrency problem: Readers and writers

- Processes share a common database
- Some want read access (readers) while others want ability to write (writers)
- Readers should be able to share the database, but all other processes must block if a writer gets access

Solving readers and writers

```
semaphore mutex = 1;    //control access to database

void read() {
    down(mutex);
    //perform read
    up(mutex);
}
```

```
void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes.
B. No. It allows deadlock
C. No. It creates some other problem
D. What are up and down again?

Solving readers and writers

```
semaphore mutex = 1;    //control access to database

void read() {
    down(mutex);
    //perform read
    up(mutex);
}

void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes.
B. No. It allows deadlock
C. No. It creates some other problem (doesn't allow more than 1 reader)
D. What are up and down again?

Solving readers and writers

```
semaphore mutex = 1;    //control access to database
int numR = 0;           //number of active readers

void read() {
    numR++;
    if(numR == 1) down(mutex);
    //perform read
    numR--;
    if(numR == 0) up(mutex);
}
```

```
void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I can't think this hard anymore

Solving readers and writers

```
semaphore mutex = 1;    //control access to database
int numR = 0;           //number of active readers

void read() {
    numR++;
    if(numR == 1) down(mutex);
    //perform read
    numR--;
    if(numR == 0) up(mutex);
}

void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I can't think this hard anymore

Solving readers and writers

```
semaphore mutex = 1, num_mutex = 1;    //mutex protects database, num_mutex protects numR
int numR = 0;                          //number of active readers

void read() {
    down(num_mutex);
    numR++;
    if(numR == 1) down(mutex);
    up(num_mutex);
    //perform read
    down(num_mutex);
    numR--;
    if(numR == 0) up(mutex);
    up(num_mutex);
}
```

```
void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I can't think this hard anymore

Solving readers and writers

```
semaphore mutex = 1, num_mutex = 1;    //mutex protects database, num_mutex protects numR
int numR = 0;                          //number of active readers

void read() {
    down(num_mutex);
    numR++;
    if(numR == 1) down(mutex);
    up(num_mutex);
    //perform read
    down(num_mutex);
    numR--;
    if(numR == 0) up(mutex);
    up(num_mutex);
}
```

```
void write() {
    down(mutex);
    //perform write
    up(mutex);
}
```

Does this successfully implement readers and writers?

A. Yes
B. I sure hope so
C. No. It allows deadlock
D. No. It creates some other problem
E. I can't think this hard anymore

(but does privilege readers since they never have to give up the database)