Synchronization (3)

David A. Eckhardt
School of Computer Science
Carnegie Mellon University

de0u@andrew.cmu.edu

Status Rendezvous

P1 Handin

- See academic.cs.15-412.announce for directions
  - (please follow them!)
- A word about the p-word

Partner selection for Project 2

- de0u+partner@andrew
  - or de0u+partners@andrew (I am learning)
- By Tuesday 2002-03-04 23:59 EST
- Only 25 as of midnight (some 2-way)

Project 2

- Out: Wednesday, February 5
- In: Wednesday, February 19

HW1???

- Don’t be surprised if it’s out Friday

Outline

More flavors of mutual exclusion

- semaphore
- monitor

A word about ***deadlock***

- (Much) more to come

Semaphore

Basic concept

- Integer: number of free instances of a resource
- Threads should not run unless they are allocated an instance

Operations

- wait()
  - aka P() aka proberen(), “wait”
  - wait until value > 0
  - decrement value
- signal()
  - aka V() aka verhogen(), “increment”
  - increment value

Just one small issue...

- wait() and signal() must be *atomic*!

Semaphore - example

“Mutex-style” semaphore

```
semaphore m = 1;

do {
 wait(m);
 ...critical section...
 signal(m);
 ...remainder section...
} while (1);
```

Semaphore - example

“Condition-style” semaphore

```
semaphore c = 0;
```

Process 1	Process 2
	wait(c);
...compute some important result...	
signal(c);	
	...consume result...

More powerful than condition variables

Process 1	Process 2
...compute some important result...	
signal(c);	
	wait(c);
	...consume result...

Semaphore vs. mutex/condition

Good news

- Semaphore is a higher-level construct
    - Integrates mutual exclusion, waiting
    - Avoids mistakes common in mutex/condition API

Bad news

- Semaphore is a higher-level construct
    - Integrates mutual exclusion, waiting
        - Some semaphores are “mutex-like”
        - Home semaphores are “condition-like”
        - How’s a poor library to know?

Semaphores - 31 Flavors

Binary semaphore

- It counts, but only from 0 to 1!
  - “Available” / “Not available”
- Consider this a hint to the implementor...
  - “Think mutex!”

Non-blocking semaphore

- wait(semaphore, timeout)

Deadlock-avoidance semaphore

- #include <deadlock.lecture>

Semaphore Wait: The Inside Story

Wait

```
wait(semaphore s) {
  ACQUIRE EXCLUSIVE ACCESS
  --s->count;
  if (s->count < 0) {
    enqueue(s->queue, my_thread());
    ATOMICALLY
      RELEASE EXCLUSIVE ACCESS
      thread_pause()
  } else {
    RELEASE EXCLUSIVE ACCESS
}
```

Semaphore Signal - The Inside Story

Wait

```
signal(semaphore s) {
 ACQUIRE EXCLUSIVE ACCESS
 ++s->count;
 if (s->count <= 0) {
  tid = dequeue(s->queue);
  thread_wakeup(tid);
 }
 RELEASE EXCLUSIVE ACCESS
```

What’s all the shouting?

- spin-waiting on an exclusion algorithm, a la mutex
- OS-assisted atomic de-scheduling

Monitor

Basic concept

- Semaphore code may have fewer errors than mutex/condition
- But there are still common errors
  - Saying “signal()” when you mean “wait()”, or reverse
  - Accidentally omitting one or the other

Monitor: higher-level abstraction

- Collection of high-level language procedures
- All access some shared state
- Compiler adds synchronization code
  - A thread running *any* procedure blocks *all* thread entries

Monitor Example

Monitor “commerce”

```
int cash_in_till[N_STORES] = { 0 };
int wallet[N_CUSTOMERS] = { 0 } ;

void buy(int cust, int store, int price) {
 cash_in_till[store] += price;
 wallet[cust] -= price;
}

boolean give(int p1, int p2, int val) {
 if (wallet[p1] >= val) {
  wallet[p1] -= val;
  wallet[p2] += val;
  return (true);
} else {
  return (false);
}
```

Monitors - More Features

Automatic mutal exclusion is nice...

- ...but it is too strong
  - Sometimes one thread needs to wait for another
  - Automatic mutual exclusion forbids this

Monitor condition variables

- Similar to mutex/condition conditions we’ve seen
- condition_wait(cvar) - needs only one parameter
  - mutex-to-drop is implicit (“the” monitor mutex)
- Policy question - which thread to run?
  - Some implementations: signalling thread
  - Others: signalled thread
  - Others: signal() side effect forces instant monitor exit

A Word About Deadlock

“Pushy trains”

- Only one train per track segment (either direction ok)
- Once on a track segment, don’t back up (*very* slow)

Ok for a while...

NY->San Franciso Train	San Francisco -> NY train
allocate(10)	allocate(1)
allocate(9)	allocate(2)
release(10)	release(1)
allocate(8)	allocate(3)
release(9)	release(2)

A Word About Deadlock

Disaster

allocate(7)	allocate(4)
release(8)	release(3)
allocate(6)	allocate(5)
release(7)	release(4)
allocate(5)	*allocate(6)*
release(6) - never!	release(5) - never!

What do we do?

- There is no easy answer
- We will talk about hard answers (soon)