CSc 360
Operating Systems
Semaphores

Jianping Pan

Summer 2006

Review: synchronization

- Peterson's solution
  - software-based solution

```
do {
        flag [i]:= true;
        turn = j;
        while (flag [j] and turn == j) ;
        /* critical section */
        flag [i] = false;
        /* remainder section */
} while (1);
```

  - assumption? limitation?

"Test-and-set"

- Test and set value atomically

```
boolean TestAndSet(boolean &target) {
       boolean rv = target;
       target = true;
       return rv;
}
boolean lock = false; /* shared variable */
do {
       while (TestAndSet(lock)) ;
       /* critical section */
       lock = false;
       /* remainder section */
}
```

- Any problem?

Semaphores

- Semaphore API
  - Semaphore *S* – integer variable
    - binary semaphore
    - counting semaphore
  - two indivisible (atomic) operations
    - also known as P() and V()

```
wait (S):
       while S≤ 0 do no-op;
        S--;

signal (S):
       S++;
```

Using semaphores

- Mutual exclusion
  - binary semaphore
  - shared data

```
semaphore mutex; //initially mutex = 1
```

  - process *Pi*

```
do {
    wait(mutex);
    /* critical section */
    signal(mutex);
    /* remainder section */
} while (1);
```

- Resource access
  - counting semaphore
  - initially, the number of resource instances

Semaphore implementation

- Semaphores without busy waiting
  - block(): block the caller process
  - wakeup(): wakeup another process

```
wait(S):
        S.value--;
        if (S.value < 0) {
                add this process to S.L;
                block();
        }

signal(S):
        S.value++;
        if (S.value <= 0) {
                remove a process P from S.L;
                wakeup(P);
        }
```

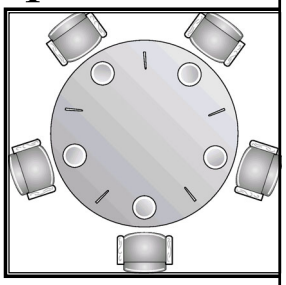
More on using semaphores

- Ordered execution
  - initially, flag = 0;
  - P1: …; *do_me_first*; signal (flag);
  - P2: …; wait (flag); *then_follow_on*;
- Caution
  - deadlock
    - wait (A); wait (B); …; signal (A); signal (B);
    - wait (B); wait (A); …; signal (B); signal (A);
  - starvation

Example: dining philosophers

- Shared data
  - Initially all values are 1

```
semaphore chopstick[5];
```



- Philosopher *i*:

```
do {
  wait(chopstick[i])
  wait(chopstick[(i+1) % 5])
     …
    eat
     …
  signal(chopstick[i]);
  signal(chopstick[(i+1) % 5]);
     …
    think
     …
  } while (1);
```

This lecture

- Hardware-assisted synchronization
  - test-and-set and *swap*
- Semaphores
  - with(out) busy waiting
- Properties
  - mutual exclusion
  - making process
  - bounded waiting

Next lecture

- More on synchronization
  - read OSC7Ch6