Semaphores

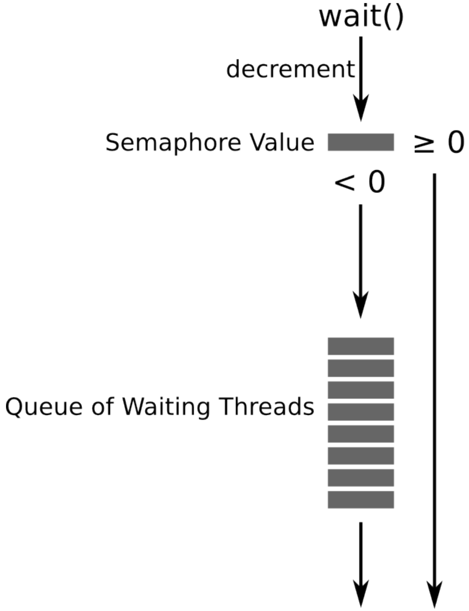
Daniel Zappala

CS 360 Internet Programming
Brigham Young University

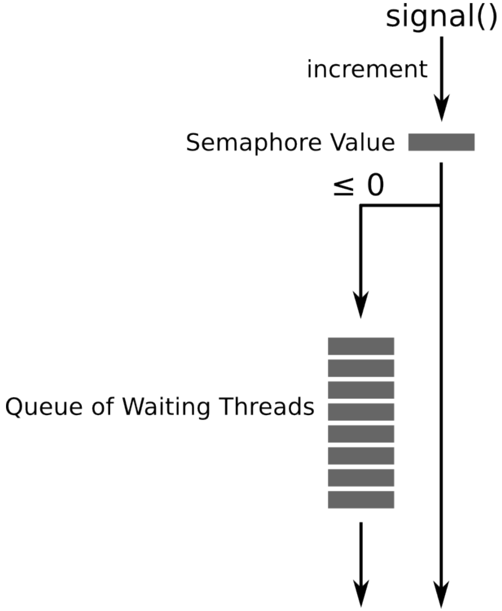
Semaphores

- semaphore is a shared variable maintained by OS
  - contains an integer and a queue
  - value initialized $>= 0$
- `wait(s)`: wait for a signal on semaphore *s*
  - decrements semaphore, blocks if value < 0
  - if blocked, process put on the queue, suspends until signal is sent
- `signal(s)`: transmit a signal to semaphore *s*
  - increments semaphore
  - if value $<= 0$ then unblock someone
- `wait()` and `signal()` are atomic operations and cannot be interrupted

wait()


wait()
decrement
Semaphore Value
≥ 0
< 0
Queue of Waiting Threads

signal()


signal()
increment
Semaphore Value
≤ 0
Queue of Waiting Threads

Types of Sempahores

- binary semaphore
  - only one process at a time may be in the critical section
- counting semaphore
  - a fixed number of processes > 0 may be in the critical section
- OS determines order that process are released from the queue, but usually FIFO in order to prevent starvation

Using Semaphores

```
semaphore s = 1;

void thread(int i) {
    while (true) {
        wait(s);
        /* critical section */
        signal(s);
       /* remainder */
    }
}
```

- semaphore protects critical section
- can set s to > 1 to let more than one process in the critical section
  - $s >= 0$: number that can enter
  - $s < 0$: number that are waiting

POSIX Semaphores

POSIX Semphores

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_wait(sem_t * sem);
int sem_trywait(sem_t * sem);
int sem_post(sem_t * sem);
```

- `sem_init()`: sets initial value of semaphore; `pshared = 0` indicates semaphore is local to the process
- `sem_wait()`: suspends process until semaphore is > 0, then decrements semaphore
- `sem_trywait()`: returns EAGAIN if semaphore count is $= 0$
- `sem_post()`: increments semaphore, may cause another thread to wake from `sem_wait()`

Example Code

- see example code `semaphore.cc`

▸ GitHub

Producer Consumer

Producer Consumer

```
sem_t lock, numItems, numSpaces;
sem_init(&lock,0,1);
sem_init(&numItems,0,0);
sem_init(&numSpaces,0,BUFFER_SIZE);
```

producer:

```
while (True) {
    produce();
    sem_wait(&numSpaces);
    sem_wait(&lock);
    append();
    sem_post(&lock);
    sem_post(&numItems);
}
```

consumer:

```
while (True) {
    sem_wait(&numItems);
    sem_wait(&lock);
    take();
    sem_post(&lock);
    sem_post(&numSpaces);
    consume();
}
```

Looking at the Code ...

1. *What is the purpose of semaphore lock?*
2. *What is the purpose of semaphore numSpaces?*
3. *What is the purpose of semaphore numItems?*
4. *Why are the semaphores initialized to different values?*
5. *Can the producer swap the signals for numItems and lock?*
6. *Can the consumer swap the waits for numItems and lock?*

Important Insights

- two purposes for semaphores
  - mutual exclusion: semaphore *lock* controls access to critical section
  - signalling: semaphore *numSpaces* coordinates the number of spaces in the buffer, so the producer waits if the buffer is full
  - signalling: semaphore *numItems* coordinates the number of items in the buffer, so the consumer waits if the buffer is empty
- avoid race conditions
  - *item* keeps a local copy of the data protected by the semaphore so that it can be accessed later
  - reduces amount of processing inside the critical section
- ordering of semaphores is important