Synchronization, Part 2: Counting Semaphores

Alex Kizer edited this page on Mar 8 · 4 revisions

What is a counting semaphore?

A counting semaphore contains a value and supports two operations "wait" and "post". Post increments the semaphore and immediately returns. "wait" will wait if the count is zero. If the count is non-zero the semaphore decrements the count and immediately returns.

An analogy is a count of the cookies in a cookie jar (or gold coins in the treasure chest). Before taking a cookie, call 'wait'. If there are no cookies left then `wait` will not return: It will `wait` until another thread increments the semaphore by calling post.

In short, `post` increments and immediately returns whereas `wait` will wait if the count is zero. Before returning it will decrement count.

How do I create a semaphore?

This page introduces unnamed semaphores. Unfortunately Mac OS X does not support these yet.

First decide if the initial value should be zero or some other value (e.g. the number of remaining spaces in an array). Unlike pthread mutex there are not shortcuts to creating a semaphore - use `sem_init`

```
#include <semaphore.h>

sem_t s;
int main() {
  sem_init(&s, 0, 10); // returns -1 (=FAILED) on OS X
  sem_wait(&s); // Could do this 10 times without blocking
  sem_post(&s); // Announce that we've finished (and one more resource item is av
  sem_destroy(&s); // release resources of the semaphore
}
```

Can I call wait and post from different threads?

Yes! Unlike a mutex, the increment and decrement can be from different threads.

Can I use a semaphore instead of a mutex?

Yes - though the overhead of a semaphore is greater. To use a semaphore:

Pages 51

Find a Page…

Home
#Example Markdown
#Informal Glossary
#Piazza: When And How to Ask For Help
C Programming, Part 1: Introduction
C Programming, Part 2: Text Input And Output
C Programming, Part 3: Common Gotchas
C Programming, Part 4: Debugging
Deadlock, Part 1: Resource Allocation Graph
Deadlock, Part 2: Deadlock Conditions
File System, Part 1: Introduction
File System, Part 2: Files are inodes (everything else is just data...)
File System, Part 3: Permissions
File System, Part 4: Working with directories
File System, Part 5: Virtual file systems
Show 36 more pages…

- Initialize the semaphore with a count of one.
- Replace `...lock` with `sem_wait`
- Replace `...unlock` with `sem_post`

Can I use sem_post inside a signal handler?

Yes! `sem_post` is one of a handful of functions that can be correctly used inside a signal handler. This means we can release a waiting thread which can now make all of the calls that we were not allowed to call inside the signal handler itself (e.g. `printf`).

```
#include <stdio.h>
#include <pthread.h>
#include <signal.h>
#include <semaphore.h>
#include <unistd.h>

sem_t s;

void handler(int signal)
{
    sem_post(&s); /* Release the Kraken! */
}

void *singsong(void *param)
{
    sem_wait(&s);
    printf("I had to wait until your signal released me!\n");
}

int main()
{
    int ok = sem_init(&s, 0, 0 /* Initial value of zero*/);
    if (ok == -1) {
       perror("Could not create unnamed semaphore");
       return 1;
    }
    signal(SIGINT, handler); // Too simple! See note below

    pthread_t tid;
    pthread_create(&tid, NULL, singsong, NULL);
    pthread_exit(NULL); /* Process will exit when there are no more threads */
}
```

Note robust programs do not use `signal()` in a multi-threaded program ("The effects of signal() in a multithreaded process are unspecified." - the signal man page); a more correct program will need to use `sigaction`.

How do I find out more?

Play using a real linux system! (9/19/14: Linux-In-the-Browser project is missing semaphore.h - this will be fixed in the next update). Read the man pages:

- sem_init
- sem_wait
- sem_post

- sem_destroy