Analysis of semaphore implementations

by Alexei Finski

The following implementation is provided at https://sites.cs.ucsb.edu/~rich/class/cs170/notes/Semaphores/index.html:

```
void P(sema *s){
  pthread_mutex_lock(&s->lock);
  s->value--;
  while (s->value < 0){
    if (s->waiters < -1 * s->value){
      s->waiters++;
      pthread_cond_wait(&s->wait, &s->lock);
      s->waiters--;
    }else{
      break;
    }
  }
  pthread_mutex_unlock(&s->lock);
  return;
}

void V(sema *s){
  pthread_mutex_lock(&s->lock);
  s->value++;
  if (s->value <= 0){
    pthread_cond_signal(&s->wait);
  }
  pthread_mutex_unlock(&s->lock);
}
```

s->waiters is initialized to 0 and is non-negative because for every decrement operation there is a preceding increment operation. Thus, for any *s->value* >= 0 *s->waiters < -1 * s->value* is false. Based on this observation, the above implementation simplifies to the following implementation of P, which will be considered.

```
void P(sema *s){
  pthread_mutex_lock(&s->lock);
  s->value--;
  while (s->waiters < -1 * s->value){
    s->waiters++;
    pthread_cond_wait(&s->wait,&s->lock);
    s->waiters--;
  }
  pthread_mutex_unlock(&s->lock);
}
```

Wlog, let *s->value* == -2 and *s->waiters* == 2. Consider the following example:
1) Thread A calls V, increments *s->value* to -1, and signals with *pthread_cond_signal*.
2)Thread A continues running, calls P, decrements *s->value* to -2, and does not call *pthread_cond_wait*.
3)Thread B is awakened by the signal from *pthread_cond_signal* called by thread A, reacquires mutex, and decrements *s->waiters* to 1. Because *s->value* == -2 and *s->waiters* == 1, thread B does not exit the while loop, increments *s->waiters* to 2 and calls *pthread_cond_wait*.

Thus, thread A “received” its own signal and avoided being blocked, whereas thread B continues being blocked. Therefore, the above implementation can lead to thread starvation.

The problem is addressed by guaranteeing a call to *pthread_cond_wait (cond_wait)* with a do...while loop for each thread that calls P (sem_wait) when the value of a semaphore is <= 0 after mutex acquisition in P (sem_wait), as provided in The Little Book of Semaphores by Allen B. Downey (Version 2.2.1):

```
void sem_wait(Semaphore *semaphore){
  mutex_lock(semaphore->mutex);
  semaphore->value--;
  if (semaphore->value < 0){
    do{
      cond_wait(semaphore->cond, semaphore->mutex);
    }while (semaphore->wakeups < 1);
    semaphore->wakeups--;
  }
  mutex_unlock(semaphore->mutex);
}

void sem_signal(Semaphore *semaphore){
  mutex_lock(semaphore->mutex);
  semaphore->value++;
  if (semaphore->value <= 0){
    semaphore->wakeups++;
    cond_signal(semaphore->cond);
  }
  mutex_unlock(semaphore->mutex);
}
```