Semaphores

• Learning Objectives
  • Explain the pt61_sem implementation.
  • Solve synchronization problems using semaphores.
Primitives of Primitives

• It turns out that once you have a good synchronization primitive, you can often build other primitives out of them.

• Example:
  • Processors give you a few atomic instructions and you build a collection of primitives on top of those instructions.
  • Pthreads gives us condition variables and mutexes (and read-write locks), but no semaphores, so today we’re going to build them!
  • We could build read/write locks ourselves too if we wanted to.
Semaphore API

• We’ll define a new set of routines (pthread + cs61 semaphores):

• Declarations are in pt61_sem.h:
  
  typedef struct _sem *pt61_sem;

  int pt61_sem_create(pt61_sem *, int);
  void pt61_sem_destroy(pt61_sem);

  void pt61_sem_P(pt61_sem);
  void pt61_sem_V(pt61_sem);
One can implement different synchronization primitives in terms of other primitives (suggesting that perhaps the term primitive is inappropriate). In this example, we'll use pthread mutexes and CVs to implement semaphores. There are a number of ways we could do this; this is perhaps the simplest.

```c
#include <assert.h>
#include <stdlib.h>
#include "pt61_sem.h"

struct sem {
    // XXX What fields do we need in a semaphore structure?
    int x; // Place holder because you can't have a 0-size structure
};

int pt61_sem_create(pt61_sem *sp, int n)
{
    pt61_sem sem;

    if ((sem = (pt61_sem)malloc(sizeof(struct _sem))) == NULL)
        return (-1);

    // How do we initialize the fields of the semaphore structure?
```
semaphores [8] pwd
/home/ubuntu/cs61/cs61-videos/semaphores
semaphores [9] ls
Makefile  check.awk  pingpong2-sem.c  semaphores-soln.c
README.txt  pingpong2-sem-soln.c  pt61_sem.h  semaphores.c
semaphores [10] vi semaphores.c
clang -g -00 -Wall -I. -c pingpong2-sem.c
clang -g -00 -Wall -I. -c semaphores.c
semaphores.c:34:1: warning: unused label 'err' [-Wunused-label]
err:
...
semaphores.c:33:1: warning: unused label 'err2' [-Wunused-label]
err2:
...
2 warnings generated.
clang -g -00 -Wall -I. -o pingpong2-sem pingpong2-sem.o -lpthread semaphores.o
semaphores [12] vi semaphores.c
semaphores [13] make
clang -g -00 -Wall -I. -c semaphores.c
clang -g -00 -Wall -I. -o pingpong2-sem pingpong2-sem.o -lpthread semaphores.o
clang -g -00 -Wall -I. -c pingpong2-sem-soln.c
clang -g -00 -Wall -I. -c semaphores-soln.c
clang -g -00 -Wall -I. -o pingpong2-sem-soln pingpong2-sem-soln.o -lpthread semaphores-soln.o
semaphores [14]
Ping Pong Returns

• We’ve solved the pingpong problem using:
  • Pipes
  • Locks
  • Condition Variables

• Why not solve it using CVs?
  • We starting here with a slightly fancier version that takes a value N and spawns that many pings and that many pongs.
  • All the infrastructure is set up for you.
  • All we need to do is:
    • Define the structure we pass to the threads
    • Write the code to initialize those structures
    • Write the thread function!
These examples build on/mimic the multi-process ping pong program from lecture 18 and the select video. The challenge this time is to synchronize two pthreads who need to alternate printing out pings and pongs to the console.

The main program creates N threads of each type. We demonstrate the use of semaphores.

```c
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "pt61_sem.h"
```

How many pings and pongs each thread will try to print
`#define TOTAL_MSGS 50`

Structure that we'll use to transmit information to the threads.

```c
struct pp_thread_info {
    // What do I need here to synchronize?
```

`<ores/pingpong2-sem.c` CWD: `/home/ubuntu/cs61/cs61-videos/semaphores Line: 1`
"pingpong2-sem.c" 112L, 2702C"
```c
void *pp_thread(void *arg)
{
    struct pp_thread_info *infop;
    int i;

    infop = arg;
    for (i = 0; i < TOTAL_MSGS; i++)
    {
        // How do I know when it's OK to run?

        // Once you get here, you know you're OK to run.
        printf("%s\n", infop->msg);

        // What do I have to do after I run?
    }

    return (NULL);
}

// Usage: pingpong2-sem N
// N is the number of threads of each type to spawn
```
Wrapping Up

• You can implement one synchronization primitive in terms of another.
• Writing properly synchronized code requires two pieces:
  • Think carefully about what you’re trying to accomplish.
  • Pick the right primitive.