15-213 Recitation 13

Introduction to Computer Systems

Fahim Dalvi

28 November, 2013

Proxy Lab

- Due 4th December, 2013
- No late days for the assignment!
- We will have interviews with each of you
  - NOT a group based assignment
  - Interviews are on the Reading Day, the schedule will be out soon!

Proxy lab

- Three main steps
  - Step 1: Implement a sequential proxy
  - Step 2: Make the proxy concurrent
  - Step 3: Implement a web-cache

Step 1: Implement a sequential proxy

- Check out last recitation!
- Hopefully you are done with this, or at least started and are close

Step 2: Make the proxy concurrent

- You have to use the Pthreads library
- Why not just use 'fork()' and avoid all the hassle?

Step 2: Make the proxy concurrent

- You have to use the Pthreads library
- Why not just use 'fork()' and avoid all the hassle?
  - Overhead of creating a new process, your proxy should be fast!
  - Eventually you will also need to 'cache' web objects, sharing stuff between processes is much harder!

Step 2: Make the proxy concurrent

- You have to use the Pthreads library
- Why not just use 'fork()' and avoid all the hassle?
  - Overhead of creating a new process, your proxy should be fast!
  - Eventually you will also need to 'cache' web objects, sharing stuff between processes is much harder!

Step 2: Make the proxy concurrent

- So you can share data very easily between threads
- With great power, comes great responsibility (and headaches)
  - Possibility of race conditions

Lets look at some code!

```
#include “csapp.h”

static volatile int global = 0;

int main(void) {
 pthread_t tid1, tid2;
 pthread_create(&tid1, NULL,
       thread, NULL);
 pthread_create(&tid2, NULL,
       thread, NULL);
 pthread_join(tid1, NULL);
 pthread_join(tid2, NULL);
 printf(“%d”, global);
 return 0;
}
```

```
void *thread(void *vargp) {
 int i;
 for (i = 0; i < 100; i++) {
    global++;
 }
 return NULL;
}
```

What will be the output?

Lets look at some code!

```
#include “csapp.h”

static volatile int global = 0;

int main(void) {
 pthread_t tid1, tid2;
 pthread_create(&tid1, NULL,
       thread, NULL);
 pthread_create(&tid2, NULL,
       thread, NULL);
 pthread_join(tid1, NULL);
 pthread_join(tid2, NULL);
 printf(“%d”, global);
 return 0;
}
```

```
void *thread(void *vargp) {
 int i;
 for (i = 0; i < 100; i++) {
    global++;
 }
 return NULL;
}
```

Any value from 2-200!

Lets look at some code!

- Shared variable is `global`
- `global++`
  - Can we divided into three atomic operations
    1)Read the variable into a register
    2)Increment
    3)Store the variable back to memory

Normal Order of Operations

R → I → S

Lets look at some code!

- Lets look at some possibilities
  - 200 → This is simple, if one thread finishes completely first, and then the second thread starts
  - 100 → A little tricky, access pattern must be R R I S I S, where red is thread 1, and blue is thread 2.
  - 2 → The trickiest of them all!
    - Imagine, R R I S R I S ..... I S (Thread 2 run's 99 times)
    - I S R I S R I S ..... I S (Thread 1 run's its remaining 99 times)

A peculiar example

- Thread's can access each other's stacks. Consider the following example:

```
#include<stdio.h>
#include<pthread.h>

void *thread(void *vargp)
{
     int myid = *(int *)vargp;
     while(1) {
          *(int *)vargp = *(int *)vargp +1;
     }
}

int main()
{
     int i;
     pthread_t tid;
     pthread_create(&tid, NULL, thread, (void *)&i);

     while(1)
          printf("%d\n",i);
     pthread_exit(NULL);
}
```