Synchronization

and Proxylab testing

15-213: Introduction to Computer Systems
Recitation 14: November 26, 2012

Praveen Kumar Ramakrishnan (praveenr)

Section E

Topics

- News
- Thread safety
- Synchronization
- Proxylab Testing

News

- Proxylab due on Sunday, Dec 2
- Last date to handin is Wednesday, Dec 5
- Each group gets 2 grace days

Thread Safety

Race condition

- You might have experienced race conditions in shell lab!

- A race occurs when the correctness of a program depends on one thread reaching point x in its control flow before another thread reaches point y.
  - Access to shared variables and data structures
  - Threads dependent on a condition

Race condition - Example

- global++;

- Think of it as:
  1. Load value of global into register
  2. Add one to register
  3. Store new value in address of global

- We don't want threads to interleave
  - 1-2-3-1-2-3

- But they might…
  - 1-2-1-2-3-3

Unsafe multi-threading

```
#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;
}
```

Output:
??

Unsafe multi-threading

```
#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;
}
```

Output:
Can print any integer from 2 to 200!

Synchronization

Synchronization

- Need to synchronize threads so that any critical region has at most one thread in it

- Ways to do synchronization:
  1. **Semaphore**
     - Restricts the number of threads that can access a shared resource
  2. **Mutex**
     - Special case of semaphore that restricts access to **one** thread
  3. **Reader/Writer locks**
     - Multiple readers allowed
     - Single writer allowed
     - No readers allowed when writer is present

Semaphore

- Classic solution: Dijkstra's P and V operations on semaphores

- Semaphore: non-negative integer synchronization variable.
  - P(s): [while (s == 0) wait(); s--;]
  - V(s): [s++;]
  - OS guarantees that operations between brackets [ ] are executed indivisibly.
  - Only one P or V operation at a time can modify s.
  - Semaphore invariant: (s >= 0)
  - Initialize s to the number of simultaneous threads allowed

Reader/Writer locks

- Many concurrent readers
- Only one writer

- Good for data-structures that are read often
  - Like caches!

- Ask for either “read” or “write” permission, and the lock will wake you up when it's your turn.

POSIX synchronization functions

- Semaphores
  - sem_init
  - sem_wait
  - sem_post
- Mutex
  - pthread_mutex_init
  - pthread_mutex_lock
  - pthread_mutex_unlock
- Read-write locks
  - pthread_rwlock_init
  - pthread_rwlock_rdlock
  - Pthread_rwlock_wrlock

Safe multi-threading

```
#include “csapp.h”

static volatile int global = 0;
static sem_t sem;

int main(void) {
    pthread_t tid1, tid2;
    sem_init(&sem, 0, 1);
    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++) {
        sem_wait(&sem);
        global++;
        sem_post(&sem);
    }
    return NULL;
}
```

Output:
??

Safe multi-threading

```
#include “csapp.h”

static volatile int global = 0;
static sem_t sem;

int main(void) {
    pthread_t tid1, tid2;
    sem_init(&sem, 0, 1);
    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++) {
        sem_wait(&sem);
        global++;
        sem_post(&sem);
    }
    return NULL;
}
```

Output:
Always prints 200

Proxylab Testing

Proxylab Testing

- Document all design decisions
- No driver program to evaluate correctness
- You will have to come up with your own tests. TAs will be interested in your test cases.
- Test simple pages at the beginning and more complicated ones as your proxy improves
- Not all pages will work!
  - Only need to handle GET requests. Other requests are optional.
  - Https pages (Facebook, gmail, etc.)

Testing Tools

- Use the following tools to test and debug your proxy:
  - Netcat
  - Curl
  - Thttpd
  - See writeup for more details
- Make sure you test all code paths and edge cases
- You can even come up with your own test harness

Web browser

- Configure your browser to use an HTTP proxy
  - Exciting and fun!
- Suggested sites:
  - http://www.cs.cmu.edu/213/
  - http://www.cs.cmu.edu/
  - http://www.nytimes.com/
  - http://www.cnn.com/
  - http://www.youtube.com/
- Use your proxy and test the websites you normally visit
- Your proxy should not crash!
  - Handle error conditions gracefully

Testing proxy caching

- Find a website that changes frequently to test your caching
- Modern web browsers have caches of their own
  - Disable browser caching before attempting to test your proxy's cache

Questions?