Lecture 06—More on Threads

ECE 459: Programming for Performance

Patrick Lam

University of Waterloo

January 16, 2015

Mutual Exclusion

Mutexes are the most basic type of synchronization.

- Only one thread can access code protected by a mutex at a time.
- All other threads must wait until the mutex is free before they can execute the protected code.

Live Coding Example: Mutual Exclusion

Creating Mutexes—Pthreads Example

```
pthread_mutex_t m1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t m2;

pthread_mutex_init(&m2, NULL);
...
pthread_mutex_destroy(&m1);
pthread_mutex_destroy(&m2);
```

- Two ways to initialize mutexes: statically and dynamically
- If you want to include attributes, you need to use the dynamic version

Creating Mutexes—C++11 Example

```
#include <mutex>

std::mutex m1, m2;
```

Resources released when objects go out of scope.

Pthreads Mutex Attributes

- **Protocol**: specifies the protocol used to prevent priority inversions for a mutex
- **Prioceiling**: specifies the priority ceiling of a mutex
- **Process-shared**: specifies the process sharing of a mutex

You can specify a mutex as *process shared* so that you can access it between processes. In that case, you need to use shared memory and `mmap`, which we won't get into.

Using Pthreads Mutexes: Example

```
// code
pthread_mutex_lock(&m1);
// protected code
pthread_mutex_unlock(&m1);
// more code
```

- Everything within the `lock` and `unlock` is protected.
- Be careful to avoid deadlocks if you are using multiple mutexes.
- Also you can use `pthread_mutex_trylock`, if needed.

Using C++11 Mutexes: Example

```
// code
m1.lock()
// protected code
m1.unlock()
// more code
```

- Everything within the `lock` and `unlock` is protected.
- Be careful to avoid deadlocks if you are using multiple mutexes.
- Also you can use `mutex::trylock()`, if needed.

Data Race Example

Recall that dataraces occur when two concurrent actions access the same variable and at least one of them is a **write**

```
...
static int counter = 0;

void* run(void* arg) {
    for (int i = 0; i < 100; ++i) {
        ++counter;
    }
}

int main(int argc, char *argv[])
{
    // Create 8 threads
    // Join 8 threads
    printf("counter = %i\n", counter);
}
```

Is there a datarace in this example? If so, how would we fix it?

Example Problem Solution

```
...
static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
static int counter = 0;

void* run(void* arg) {
    for (int i = 0; i < 100; ++i) {
        pthread_mutex_lock(&mutex);
        ++counter;
        pthread_mutex_unlock(&mutex);
    }
}

int main(int argc, char *argv[])
{
    // Create 8 threads
    // Join 8 threads
    pthread_mutex_destroy(&mutex);
    printf("counter = %i\n", counter);
}
```