COMS2001: Operating Systems
Tutorial 2: Threads

September 7, 2017

Contents

1 Terminology

- **thread** - A thread of execution is a unit that can be scheduled for execution by the operating system. A thread is made up of sequential instructions. Multiple threads can share the same address space, global data, and the heap, but each thread has its own program counter and its own stack.

- **pthreads** - POSIX threads, a POSIX-compliant (standard specified by IEEE) implementation of threads. In C, they are provided in the pthread.h header file.

- **pthread_create** - Creates and starts a child thread running in the *same address space* as the thread that spawned it. The child executes starting from the function specified. Internally, this is implemented by calling the clone syscall.

- **pthread_join** - Waits for a specific thread to terminate, similar to the waitpid function for processes.

- **race condition** - A race condition occurs when the timing or ordering of events affects the correctness of a program.

- **data race** - A data race is a type of race condition. It occurs when two concurrent operations target the same memory location. This can lead to unexpected or corrupted data values when multiple threads read and write to a shared variable without some form of synchronisation.

- **atomic** - An operation is deemed to be atomic if it can be executed without interruption or interference from other threads/processes.

- **critical section** - A section of code that accesses a shared resource that must not be concurrently accessed by more than a single thread. This can be achieved by thread synchronisation, such as through semaphores or mutex locks.

- **semaphore** - A synchronization primitive used to protect a shared resource by restricting the maximum number of simultaneous accesses. Semaphores contain an integer value and generally support two atomic operations: increment and decrement.
  e.g. Say the initial value of a semaphore is 2, when a thread requests access to the resouce (by decrementing the semaphore), the value becomes 1. When another thread accesses it, the value is decremented to 0. At this point, other threads requesting access will be denied and have to wait. When the threads using the resource are finished, they increment the semaphore, allowing other threads access.

- **mutex lock** - Mutex ('mutual exclusion') variables, are another way of providing thread synchronisation for shared resources. Only one thread can lock (or own) a mutex variable at any given time. So when multiple threads try to acquire the lock only one will succeed and proceed, the others have to wait until the lock is released.

2 Compilation

Using pthreads in C requires two things:

1. Include the pthreads.h header file

   ```
   #include <pthreads.h>
   ```

2. Link the pthread library during compilation (be sure to include the library **after** your source files in the compile statement):

   ```
   gcc main.c -o main.exe -lpthread
   ```

3 Questions

3.1 Threads

What output is produced by the following code? Assume the PID of the parent process is 26713.

```
void*
hello() {
    pid_t pid = getpid();
    printf("Hello world %d\n", pid);
    pthread_exit( NULL );
}

int
main( void ) {
    pthread_t thread;
    pthread_create(&thread, NULL, hello, NULL);
    pthread_join(thread, NULL);
    hello();
}
```

3.2 Join

1. List all possible outputs of the following code.
   (Hint: The program may exit before the new thread is run.)
2. Alter the program so that it always outputs '1' then '2'.

```
void* helper() {
    printf( "1\n" );
    pthread_exit( NULL );
}

int main( void ) {
    pthread_t thread;
    pthread_create( &thread, NULL, helper, NULL );
    printf( "2\n" );
}
```

3.3 Stack Allocation

What is the output of the following code?

```
void* helper( void *arg ) {
    int *num = (int*) arg;
    *num = 2;
    pthread_exit( NULL );
}

void main( void ) {
    int i = 0;
    pthread_t thread;
    pthread_create( &thread, NULL, helper, &i );
    pthread_join( thread, NULL );
    printf( "i is %d\n", i );
}
```

3.4 Heap Allocation

What is the output of the following code?

```
void* helper( void *arg ) {
    char *message = (char *) arg;
    strcpy( message, "I know how to use threads" );
    pthread_exit( NULL );
}

int main( void ) {
    char *message = malloc(26);
    strcpy(message, "I can't use threads");
    pthread_t thread;
    pthread_create( &thread, NULL, helper, message );
    pthread_join( thread, NULL );
    printf( "%s\n", message );
}
```

Why would spawning processes rather than threads produce different output in the previous two questions?

3.5 Atomic Operations

Given the declarations:

```
int x = 0;
void* data;
```

Identify all atomic operations in the following lines of code.

```
++x;
x = 10;
int y = x;
printf("x is %d\n", x);
data = malloc(8);
```

3.6 Synchronization

Explain why the following code can produce incorrect results.

```
void* helper( void *arg ) {
    int *num = (int*) arg;
    *num = (*num) + 1;
    pthread_exit(NULL);
}

int main( void ) {
    pthread_t threads[5];
    int data = 0;

    int i = 0;
    for(i = 0; i < 5; i++)
        pthread_create( &threads[i], NULL, helper, &data );
    for(i = 0; i < 5; i++)
        pthread_join( threads[i], NULL );

    printf("Data is %d\n", data);
}
```

To see the problem in action, you could try executing the program multiple times in a bash for-loop. Say the executable is 'a.out':

```
for i in `seq 1 1000`; do ./a.out; done | grep -v "Data is 5"
```

Note that the punctuation surrounding the 'seq' statement is a grave accent, the other symbol on the tilde key. It may not copy correctly in some pdf readers, so you should type it manually.

3.6.1 Semaphores

Fill in the specified lines of code to use a semaphore for synchronisation. If you want to run the code, you need to include the semaphore.h header.

```
sem_t lock;

void* helper( void *arg ) {
    _________;
    int *num = (int*) arg;
    *num = (*num) + 1;
    _________;
    pthread_exit(NULL);
}

int main( void ) {
    pthread_t threads[5];
    int data = 0;
    int i = 0;

    _________;
    for(i = 0; i < 5; i++)
        pthread_create( &threads[i], NULL, helper, &data );

    for(i = 0; i < 5; i++)
        pthread_join( threads[i], NULL );

    sem_destroy(&lock);

    printf("Data is %d\n", data);
}
```

3.6.2 Mutex Locks

Fill in the specified lines of code to use a mutex lock for synchronisation.

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void* helper( void *arg ) {
    _________;
    int *num = (int*) arg;
    *num = (*num) + 1;
    _________;
    pthread_exit(NULL);
}
```