C++ with Pthreads

Ryan M. Swanstrom

December 27, 2008

Abstract

This paper will demonstrate the use of Pthreads in C++. It is set up as lab exercise.

Contents

1 Creating and Joining Threads

```
#include
#include
#include
using namespace std;

void *thread_a(void* param);    // thread function

// This is a demo of using pthreads
int main(int argc, char** argv) {
    pthread_t t_id_a;       //declare the thread

    cout << "main():before thread is called." << endl;
    pthread_create(&t_id_a, NULL, thread_a, NULL);
    // try commenting and uncommenting the next line
//    pthread_join(t_id_a, NULL);         // the main() function does not go
                                        //beyond this line until t_id_a is done
    cout << "main(): after thread is called" << endl;

    // Without the join up above, the following loop is necessary to give
    // thread_a time to finish. Otherwise, thread_a is just killed when the
    // main() method returns
//    for (long i = 0; i < 9999999; i++) {
//        // simulate some work
//        i*i;
//    }
    cout << "main(): is exiting" << endl;
    return (EXIT_SUCCESS);
}

void *thread_a(void* param) {
    cout << "thread_a(): before work" << endl;
    for (long i = 0; i < 9999989; i++) {
        // simulate some work
        i*i;
    }
    cout << "thread_a(): after work" << endl;
    pthread_exit(0);
}
```

1.1 Directions

1. Compile and run the supplied code.

2. Why doesn't "(): after work" get printed out?

3. Uncomment the for loop in the main() function.
4. Rerun the code.
5. Does "thread_a(): after work" get printed out? Why/Why not?
6. Comment out the for loop in the main() function.
7. Uncomment the pthread_join method

2 Mutex Variables

```
#include
#include
#include
#include
using namespace std;

vector shared_vec;
int MAX_SIZE = 1;
//pthread_mutex_t mutex;          // declare a mutex
void *thread_a(void* param);    // thread function
void *thread_b(void* param);    // thread function

// This is a demo of using pthreads
int main(int argc, char** argv) {

//    pthread_mutex_init(&mutex, NULL);     // initialize the mutex

    pthread_t t_id_a;       //declare the thread for A
    pthread_t t_id_b;       //declare the thread for B

    pthread_create(&t_id_a, NULL, thread_a, NULL);
    pthread_create(&t_id_b, NULL, thread_b, NULL);
    pthread_join(t_id_a, NULL);
    pthread_join(t_id_b, NULL);

//    pthread_mutex_destroy(&mutex);

    return (EXIT_SUCCESS);
}

void *thread_a(void* param) {
//    pthread_mutex_lock(&mutex);
    for (int i = 0; i < 5; i++ ) {
        if (shared_vec.size() < MAX_SIZE) {
            // do some work
            cout << "thread_a(): A doing work, " << i << endl;
            for (int j = 0; j < 9999999; j++) {
                i*j*i*j*j*-i *j/i+i+j;
            }
            shared_vec.push_back(rand());
        }
        if (shared_vec.size() > MAX_SIZE) {
            cout << "ERROR: thread_a(): MAX_SIZE exceeded with " <<
shared_vec.size() << endl;
        }
    }
//    pthread_mutex_unlock(&mutex);
    pthread_exit(0);
}

void *thread_b(void* param) {
//    pthread_mutex_lock(&mutex);
    for (int i = 0; i < 5; i++ ) {
        if (shared_vec.size() < MAX_SIZE) {
            // do some different work
            cout << "thread_b(): B doing work, " << i << endl;
            for (int j = 0; j < 10000; j++) {
                i*j+j;
            }
            shared_vec.push_back(rand());
        }
        if (shared_vec.size() > MAX_SIZE) {
            cout << "ERROR: thread_b(): MAX_SIZE exceeded with " <<
shared_vec.size() << endl;
        }
    }
//    pthread_mutex_unlock(&mutex);
    pthread_exit(0);
}
```

2.1 Directions

1. Run the above code
2. Why does an error occur?
3. How could this erroneous behavior be avoided?
4. Why doesn't thread_b() print out an Error message?
5. Uncomment the mutex global variable and the mutex code in the main() method and in thread_a(). Leave the mutex commented out in thread_b();

6. Run the program again.
7. Why does the error still occur?
8. Uncomment the mutex code in thread_b();
9. Run the program again.
10. Does it work correctly?
11. Raise the MAX_SIZE to 7.
12. Run again. Note the output.
13. Move the mutex_lock and mutex_unlock inside the for loop in both thread methods.
14. Run again.
15. Is the output different? Why?

3 Condition Variables

3.1 Simple Example

3.1.1 C++ Code

```
#include <iostream>
#include <pthread.h>
using namespace std;

pthread_mutex_t mutex;          // declare a mutex
pthread_cond_t cond_a;          // declare a condition variable
void *thread_a(void* param);    // thread function
void *thread_b(void* param);    // thread function

// This is a demo of using pthreads with condition variables
int main(int argc, char** argv) {

    pthread_mutex_init(&mutex, NULL);       // initialize the mutex
    pthread_cond_init(&cond_a, NULL);       // initialize the condition variable

    pthread_t t_id_a;       //declare the thread for A
    pthread_t t_id_b;       //declare the thread for B

    pthread_create(&t_id_a, NULL, thread_a, NULL);
    pthread_create(&t_id_b, NULL, thread_b, NULL);
    pthread_join(t_id_a, NULL);
    pthread_join(t_id_b, NULL);
    pthread_cond_destroy(&cond_a);      // destroy and cleanup
    pthread_mutex_destroy(&mutex);      // destroy and cleanup

    return (EXIT_SUCCESS);
}

void *thread_a(void* param) {
    pthread_mutex_lock(&mutex);
    cout << "thread_a() is waiting for a signal" << endl;
    pthread_cond_wait(&cond_a, &mutex);
    cout << "thread_a(): A has received signal"<< endl;
    pthread_mutex_unlock(&mutex);
    pthread_exit(0);
}

void *thread_b(void* param) {
    pthread_mutex_lock(&mutex);
    cout << "thread_b(): sending signal to thread_a() " << endl;
    pthread_cond_signal(&cond_a);
    pthread_mutex_unlock(&mutex);
    pthread_exit(0);
}
```

3.1.2 Directions

1. Run the code numerous times.
2. Is the output always the same?

3.2 Complex Thread Coordination Example

Now a for loop is added so that thread A will wait for thread A to signal and visa versa. Notice the presence of two condition variables. One condition variable is for thread A; the other is for thread B.

The signal will be sent. If no thread is waiting for the signal, it will be stored until a thread actually issues a wait on that signal. At that point the signal will be delivered.

3.2.1 C++ Code

```
#include <stdlib.h>
#include <iostream>
#include <pthread.h>
using namespace std;

pthread_mutex_t mutex;          // declare a mutex
```

```
pthread_cond_t cond_a;          // declare a condition variable
pthread_cond_t cond_b;          // declare another condition variable
void *thread_a(void* param);    // thread function
void *thread_b(void* param);    // thread function

// This is a demo of using pthreads with condition variables
int main(int argc, char** argv) {

    pthread_mutex_init(&mutex, NULL);       // initialize the mutex
    pthread_cond_init(&cond_a, NULL);       // initialize the condition variable
    pthread_cond_init(&cond_b, NULL);       // initialize the condition variable

    pthread_t t_id_a;       //declare the thread for A
    pthread_t t_id_b;       //declare the thread for B

    pthread_create(&t_id_a, NULL, thread_a, NULL);
    pthread_create(&t_id_b, NULL, thread_b, NULL);
    pthread_join(t_id_a, NULL);
    pthread_join(t_id_b, NULL);

    pthread_cond_destroy(&cond_a);      // destroy and cleanup
    pthread_cond_destroy(&cond_b);      // destroy and cleanup
    pthread_mutex_destroy(&mutex);      // destroy and cleanup the mutex

    return (EXIT_SUCCESS);
}

void *thread_a(void* param) {
    for (int i = 0; i < 5; i++) {
        pthread_mutex_lock(&mutex);
        cout << "A: thread_a() waiting" << endl;
        pthread_cond_wait(&cond_a, &mutex);
        pthread_cond_signal(&cond_b);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(0);
}

void *thread_b(void* param) {
    for (int i = 0; i < 5; i++) {
        pthread_mutex_lock(&mutex);
        pthread_cond_signal(&cond_a);
        cout << "B: thread_b() waiting" << endl;
        pthread_cond_wait(&cond_b, &mutex);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(0);
}
```

3.2.2 Directions

1. Run the following code numerous times.
2. Is the output the same?
3. Try moving the pthread_cond_wait statement from thread_b to after the mutex unlock.
4. Run the program.
5. Is the output the same? Why/Why not?

A Setup Netbeans

1. Create a new project
2. Add the pthread library
    - Right-click on project name and go to Properties
    - Linker ⇒ Libraries
    - Click on the Libraries input
    - Select Add Standard Library
    - Select Posix Threads