Assignment #2

Pthread's Code Review

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>

const size_t NUMTHREADS = 3;

int done = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;//Mutex initialization
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;//Condition variable initialization

void* ThreadEntry( void* id )
{
        const int myid = (long)id;

        const int workloops = 5;
        int i;
        for( i=0; i<workloops; i++)
        {
                printf("thread number is %ld, Sleep i= %d\n", myid, i);
                sleep(1);
        }

        pthread_mutex_lock( &mutex );//lock thread
        printf("thread number is %ld, Locking mutex. Updating the value of done. done now=%d\n", myid, done);
        done++;
        printf("thread number is %ld, done now=%d. Sending the signal.\n", myid, done);
        pthread_cond_signal( &cond );//Unblock on condition variables
        pthread_mutex_unlock( &mutex );//unlock thread
        printf("thread number is %ld. Unlocking mutex.\n", myid);
        return NULL;
}

int main( int argc, char** argv)
{
        pthread_t threads[NUMTHREADS];//Create thread function return type

        int t;
        for( t=0; t<NUMTHREADS; t++)
        {
                printf("In main: creating thread %ld\n", t);
                pthread_create( &threads[t], NULL, ThreadEntry, (void*)(long)t );//creat thread
        }

        pthread_mutex_lock( &mutex );
        printf("%s\n","main thread lock.");
        while( done<NUMTHREADS )
        {
                printf("Main(): waited with threads. done= %d. Going into wait...\n", done);
                pthread_cond_wait( &cond, &mutex);//Blocking on condition variables
                printf("Main(): Condition signal received. done= %d\n", done);
        }

        printf("%s\n","main thread unlock.");
        pthread_mutex_unlock( &mutex );

        pthread_mutex_destroy(&mutex);
        pthread_cond_destroy(&cond);
        return 0;
}
~
~
~
-- 插入 --
```

```
[biyangf@login001 ~]$ vim Pthread.c
[biyangf@login001 ~]$ gcc Pthread.c -lpthread -o Pthread
[biyangf@login001 ~]$ ./Pthread
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
thread number is 1, Sleep i= 0
main thread lock.
Main(): waited with threads. done= 0. Going into wait...
thread number is 2, Sleep i= 0
thread number is 0, Sleep i= 0
thread number is 1, Sleep i= 1
thread number is 2, Sleep i= 1
thread number is 0, Sleep i= 1
thread number is 0, Sleep i= 2
thread number is 1, Sleep i= 2
thread number is 2, Sleep i= 2
thread number is 0, Sleep i= 3
thread number is 1, Sleep i= 3
thread number is 2, Sleep i= 3
thread number is 0, Sleep i= 4
thread number is 1, Sleep i= 4
thread number is 2, Sleep i= 4
thread number is 0, Locking mutex. Updating the value of done. done now=0
thread number is 0, done now=1. Sending the signal.
thread number is 0. Unlocking mutex.
Main(): Condition signal received. done= 1
Main(): waited with threads. done= 1. Going into wait...
thread number is 1, Locking mutex. Updating the value of done. done now=1
thread number is 1, done now=2. Sending the signal.
thread number is 1. Unlocking mutex.
Main(): Condition signal received. done= 2
Main(): waited with threads. done= 2. Going into wait...
thread number is 2, Locking mutex. Updating the value of done. done now=2
thread number is 2, done now=3. Sending the signal.
thread number is 2. Unlocking mutex.
Main(): Condition signal received. done= 3
main thread unlock.
```

When we execute the program, the main process first executes a for loop to create three child processes: thread 0, thread 1, and thread 3. Then main thread is locked and going into wait on condition variables. The 3 child processes execute a for loop to sleep 5 seconds. Thread 0 has finished the sleep loop at first. It locks the mutex, updates the value of done. After done changes from 0 to 1, it sends the signal to pthread_cond_wait(). When the wait() receives the condition signal, judge if the value of done lagers than NUMTHREADS. If not, main thread will continue to execute pthread_cond_wait() until thread 1 sends the signal and the value of done. Thread 3 also take above operations and the value of done doesn't lager than NUMTHREADS at this time. So it breaks the loop, main thread unlocks itself.