ADVANCED UNIX PROGRAMMING ASSIGNMENT REPORT

ASSIGNMENT 9

TEAM 9 — 林禾堃、馬毓昇、陳曦

Dec 2023

1. Code Implementation

First, we define pthread_barrier_t_ to simulate pthread_barrier_t:

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

// we define our own pthread_barrier_t
// we only care about count and max_count
// in order to differentiate this from real one, we add a single _ at the end
typedef struct {
        pthread_mutex_t mutex;
        pthread_cond_t cond;
        int count;
        int max_count;
} pthread_barrier_t_;
```

Then, three relevant functions are implemented as follows:

1.1 pthread_barrier_init_

```c
// initiate barrier, include mutex, condition, count and max_count
// we ignore attr and some error handling here
int pthread_barrier_init_(pthread_barrier_t_ *restrict barrier,
                        const pthread_barrierattr_t *attr,
                        unsigned count){
        barrier->count = 0;
        barrier->max_count = count;
        pthread_mutex_init(&barrier->mutex, NULL);
        pthread_cond_init(&barrier->cond, NULL);
        return 0;
}
```

The pthread_barrier_init_ function handles the initialization of the barrier's count, max_count, mutex, and condition.

1.2 pthread_barrier_wait_

```c
// wait until the count reach max_count
int pthread_barrier_wait_(pthread_barrier_t_ *barrier){
        pthread_mutex_lock(&barrier->mutex);
        barrier->count++;
        if(barrier->count >= barrier->max_count){
                barrier->count = 0;
                // using cond broadcast to notify other threads to move on
                pthread_cond_broadcast(&barrier->cond);
                pthread_mutex_unlock(&barrier->mutex);
                // according to the man page, one should return PTHREAD_BARRIER_SERIAL_
                return PTHREAD_BARRIER_SERIAL_THREAD;
        }
        else{
                pthread_cond_wait(&barrier->cond, &barrier->mutex);
                pthread_mutex_unlock(&barrier->mutex);
                // while others will return 0
                return 0;
        }
}
```

In pthread_barrier_wait_, the mutex of the barrier will be locked, and the thread will wait for the condition to be satisfied, i.e., count hasn't reach max_count. When it does, all the mutex will be unlocked after a condition broadcast.

1.3 pthread_barrier_destroy_

```c
// basically just call destroy function provided by mutex and cond
int pthread_barrier_destroy_(pthread_barrier_t_ *barrier){
        pthread_mutex_destroy(&barrier->mutex);
        pthread_cond_destroy(&barrier->cond);
        return 0;
}
```

Destroys the mutex and conditional variable of the barrier.

After having these three functions we declared a barrier,

```
static pthread_barrier_t_ barrier;

// a worker function, wait until barrier count to 6, which means all thread are set
// then output the text message
void *worker(void *arg){
        // arg is just dummy argument
        pthread_barrier_wait_(&barrier);
        printf("Thread %lld running\n", pthread_self());
        return NULL;
}
```

and implemented the worker function, which waits for the barrier to resolve and prints the required output.

Finally, we test the functions in our main function:

```
int main(){
        // first create 5 threads and initiate barrier
        pthread_t tid[5];
        pthread_barrier_init_(&barrier, NULL, 6);
        // then before create the thread, output the text as spec needs
        for(int i=0 ; i<5 ; i++){
                printf("Starting thread %d\n", i);
                pthread_create(&tid[i], NULL, worker, NULL);
        }
        // this wait is for notify that all threads are already been created
        pthread_barrier_wait_(&barrier);
        // we should end main process until all threads are done with their jobs
        for(int i=0 ; i<5 ; i++){
                pthread_join(tid[i], NULL);
        }
        // then we can destroy everything!
        pthread_barrier_destroy_(&barrier);
        return 0;
}
```

First, after creating the barrier, the worker threads are created with messages indicating it.
Second, the pthread_barrier_wait_ we implemented will let the workers print the messages together after the barrier resolved.
Finally, we detach the threads and destroy the barrier.

2. Result

```
root@genet0:~/Advanced-UNIX-Programming/HW9 # ./assignment9
Starting thread 0
Starting thread 1
Starting thread 2
Starting thread 3
Starting thread 4
Thread 2233846784 running
Thread 2233843200 running
Thread 2233844992 running
Thread 2233848576 running
Thread 2233841408 running
```