Unix Assignment9

Team19 108072244 邱煒甯 108032053 陳凱揚 112065525 簡佩如

Implementation

- pthread_barrier

```c
typedef struct {
    pthread_mutex_t mutex;
    pthread_cond_t cond;
    int count;
    int total_threads;
} pthread_barrier_t;

int pthread_barrier_init(pthread_barrier_t *barrier, int total_threads) {
    pthread_mutex_init(&barrier->mutex, NULL);
    pthread_cond_init(&barrier->cond, NULL);
    barrier->count = 0;
    barrier->total_threads = total_threads;
    return 0;
}

int pthread_barrier_wait(pthread_barrier_t *barrier) {
    pthread_mutex_lock(&barrier->mutex);
    barrier->count++;
    if (barrier->count == barrier->total_threads) {
        barrier->count = 0; // Reset for next use
        pthread_cond_broadcast(&barrier->cond);
    } else {
        while (pthread_cond_wait(&barrier->cond, &barrier->mutex) != 0);
    }
    pthread_mutex_unlock(&barrier->mutex);
    return 0;
}

int pthread_barrier_destroy(pthread_barrier_t *barrier) {
    pthread_mutex_destroy(&barrier->mutex);
    pthread_cond_destroy(&barrier->cond);
    return 0;
}
```

We implements a custom pthread barrier `pthread_barrier_t` to allow multiple threads to wait synchronously at a specific point until all threads have reached that point before continuing execution together.

The `pthread_barrier_t` structure contains a mutex `pthread_mutex_t` and a condition variable `pthread_cond_t`. The `mutex` safeguards the internal data structure of the

barrier, while the condition variable is used to signal and notify threads waiting at the barrier.

We use the following functions for initializing, waiting, and destroying the barrier:

`pthread_barrier_init()` initializes the barrier. It initializes the mutex and condition variable, setting the total number of threads the barrier should wait for.

`pthread_barrier_wait()` is the primary waiting function. Each thread calls this function upon reaching the barrier. It uses the mutex to protect access to the barrier's internal data. When a thread reaches the barrier, it increments the counter (`count`). If the counter equals the total number of threads, indicating all threads have arrived, a broadcast (`pthread_cond_broadcast()`) is sent, allowing all waiting threads to continue. Otherwise, it waits on the condition variable until all threads have arrived.

`pthread_barrier_destroy()` is used to destroy barrier-related resources. It destroys the mutex and condition variable, releasing the associated resources.

- thread_function

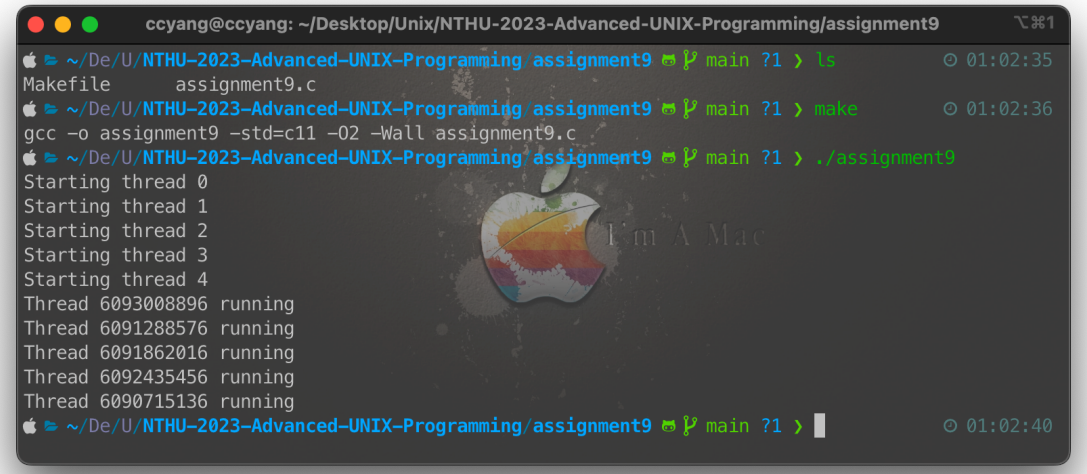
```c
#define NUM_THREADS 5

pthread_barrier_t barrier;

void *thread_function(void *id) {
    pthread_barrier_wait(&barrier);
    printf("Thread %ld running\n", (long)pthread_self());
    return NULL;
}

int main() {
    pthread_t threads[NUM_THREADS];

    pthread_barrier_init(&barrier, NUM_THREADS);

    for(int i = 0; i < NUM_THREADS; i++) {
        printf("Starting thread %d\n", i);
        int rc = pthread_create(&threads[i], NULL, thread_function, NULL);
        if (rc) {
            printf("ERROR; return code from pthread_create() is %d\n", rc);
            exit(-1);
        }
    }

    for(int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    pthread_barrier_destroy(&barrier);
```

In the `main()` function, we use the `pthread_barrier_init()` function to initialize this barrier, specifying the total number of threads to wait for, denoted by `NUM_THREADS`, which is 5.

Then we create threads. Through the `pthread_create()` function, the program creates `NUM_THREADS` threads. Each thread executes the `thread_function`. Within this function, there's a call to `pthread_barrier_wait()`, which makes the thread wait until all threads reach the barrier point. Once all threads have reached this point, they continue execution together and print their respective thread IDs.

After creating all the threads, the `main()` function uses the `pthread_join()` function to wait for all threads to finish. Finally, the `pthread_barrier_destroy()` function is used to destroy the barrier and release associated resources, effectively freeing memory.

Result


ccyang@ccyang: ~/Desktop/Unix/NTHU-2023-Advanced-UNIX-Programming/assignment9
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment9 main ?1 ❯ ls 01:02:35
Makefile assignment9.c
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment9 main ?1 ❯ make 01:02:36
gcc -o assignment9 -std=c11 -O2 -Wall assignment9.c
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment9 main ?1 ❯ ./assignment9
Starting thread 0
Starting thread 1
Starting thread 2
Starting thread 3
Starting thread 4
Thread 6093008896 running
Thread 6091288576 running
Thread 6091862016 running
Thread 6092435456 running
Thread 6090715136 running
~/De/U/NTHU-2023-Advanced-UNIX-Programming/assignment9 main ?1 ❯ 01:02:40