**Write-up for Project 3**

In this project, we are supposed to implement a user thread library, which supports multithreads and related synchronization mechanism.

1. **Basic idea:**

In a multithreads system, threads are given CPU time sequently. But there may be some critical resources which are shared by several threads. Thus synchronization is important for multithread system. Generally speaking, there are two approach to implement synchronization. Mutex lock and Semaphores.

In this project, I implemented Mutex, and some thread manipulate operations, such as thread creating, thread exiting, thread joining and so on. As for scheduling, in this project, there are 10 priority levels, each are divided into 10 sub-levels to make up a priority queue. We’ll execute threads according to the sequence of these queues.

1. **Design:**

Basic data types:

|  |  |
| --- | --- |
| typedef struct Thread\_t  {  uthread\_t tid;  uthread\_t joining;  int state;  int pri;  int arg;  int\* retval;  ucontext\_t\* ucp;  } thread\_t; | typedef struct {  steque\_node\_t\* front;  steque\_node\_t\* back;  int N;  }steque\_t; |

1. In the thread\_t structure, it contains several parameters recording the statues of one thread. To point out, the ucp is a context structure, used to record thread’s context, so that it can be switched between different threads.

2. The steque\_t structure records a priority queue’s first and last item. So we can access the first and the last thread quickly.

**Functions:**

**Priority queue:** As mentioned above, the priority queue is used to classify different threads’ priority, by giving them a priority number ranging from 0 to 100. The system will pick the highest level ready threads to execute first. When the last thread in a priority queue finished, the system will check the following queues and load related tasks.

**Mutex**: I built mutex lock in this project, using another queue and the signal mechanism. The queue is used to store different mutex requests, and the signal is for synchronization purpose.

**Thread join and thread exit**: This is a difficult part. As talked above, the system should pick the highest level’s ready-thread to execute first. Thus it comes with context switch. The scheduler should switch between threads, to make threads run parallely. I use gititimer() and setitimer() to monitor a thread’s running time. When it uses up the time slice, the scheduler will switch it to the ready queue, to wait for next time slice.

1. **Evaluation:**

At this time, the functions I mentioned above should work well. However, I failed to implement the semaphore mechanism. For later work, I’ll catch up with semaphore, and make the system more stable.