# Project Milestone Report

Michael Xu

Yao Xu

April 2023

Project repo: https://github.com/michaelxuuu/CMU-1561-Final-Project
Project website: https://michaelxuuu.github.io/CMU-1561-Final-Project/

# 1 Detailed schedule

In the project proposal, we were expecting to read material and decide API exposed to users from 4/1/2023 to 4/6/2023, and implement all APIs with distributed work queue from 4/7/2023 - 4/14/2023. We have completed the tasks previously stipulated to be completed during these two periods. Here is a more detailed schedule from now to the end of the semester:

Apr 19 - Apr 22 implement lock and malloc
Apr 23 - Apr 25 implement test cases
Apr 26 - Apr 29 collect test data
Apr 29 - May 3 Write final report and prepare poster

# 2 Status quo and future

Most of our work is contained in uthread.c and uthread.h files. Interfaces exposed to users are functions called uthread_create() and uthread_john() which will create and rape a user level thread for users. In uthread_create(), run time data structures including some worker threads (pthreads) and task queue for each pthread will be initialized. Also, a system timer is set up which will be used to send SIGALARM to notify the main thread a scheduling period.

After that, a user level context including a stack and registers will allocated on the heap and be added to a task queue. In those registers, the RIP register

will be set to the function address that current user thread will execute later, the RSP register will be set to the new stack allocated on the heap. This new user level context will be added into a lock free worker queue by the task adding algorithm for later scheduling. The task queue is a lock free doubly linked list with some constraints including task can only be added from the head of the list and deletion can only happen starting from the second element in the list.

The user level context maintenance is achieved by using uthread_context structure defined by us. It is used to keep track id, stack address, user thread state and return address. Also, a ucontext_t structure defined in #include ¡sys/ucontext.h¿ is used to maintain status of all registers. We encountered a problem in maintain registers when context switching which is changing RIP register causes segmentation fault. After doing relevant research, we found that on Linux, is we do not update CS register when context switching, the CS register will not be maintained by the OS automatically and will lead to segmentation fault. However, on MacOS, the CS and SS register will be updated automatically by the OS.

When the SIGALARM arrived, the scheduler which is basically a signal handler, will pick the next user thread to run. In the scheduler, we change the context in the scheduler's stack which is pushed on to the stack before entering the scheduler to the context of next scheduled user thread. After return from the scheduler, the new context is runnig on CPU. uthread_join() is also implemented to collect resources of user threads in the joined state.

We believe the goal set in the proposal can be achieved. For now, this library can support 200000 user threads running concurrently calculating Fibonacci 30. In the next two weeks, we will focus on implementing lock, malloc function and

test cases.

In the poster session, we plan to show a poster describing the implementation and also do a demo. People interested in this project may also try to code with is library since this is designed for programmers so user experience is also important in demo.