Report for project1

Overall design

scheduler.c can be broken into three parts: queue as well as other global bookkeeping variables related functions, scheduler functions and P/V operations.

For queues, we are using a statically defined array to store data instead of dynamic allocated linked list, only for simple implementation consideration. To be detailed, a large one-dimensional array is defined. Two pointers: *head* points to first data and *free* points the first not occupied place, i.e. one position behind the last data. Each push operation will store an element to where *free* points. Each pop will return an element pointed by head. And need to increase *free* and *head* by one in a circular manner, i.e. if reaching the end of the array then need to wrap around to the first position of the array.

For schedulers and P/V since these functions are called concurrently by multiple threads. A pthread mutex is used to protect scheduler related variables and another pthread mutex for P/V related variables. To implement the block behavior in thread not to be scheduled and thread pending on a P() operation, we are using pthread semaphores. We were formerly considering to use a switch-case segment to implement each thread’s state transition. But found it were quite complicated since this should consider many factors like blocked. Then the designed of the scheduler is a little tedious, just mapping the cases like the first time the scheduler is called, how another thread calls and found it should be block in the next time quantum.

Challenges

There are no detailed descriptions on codes within project1.c. The behavior of functions like worker\_thread and do\_worker is not quite clear to us in the beginning, which will affect how the scheme\_me is called and how the global time is progressed. This problem is only solved by debugging the built-in test cases in a time-consuming manner.

Also, there are quite limited number of test cases. We have additionally added the following one:

0.0 1 5 100

1.1 2 4 60

This is mainly to test how MLFQ handles the threads in a long run. In this test case, the Gantt chart can clearly show how two threads are moved gradually from high priority queues to the lowest ones. Although there are many corner cases for schedulers, we don’t test more due to limited time.

Trade-offs

The trade-offs are many folds. The framework provided by project1.c is simulated ones. In other words, some realistic conditions are manually simplified. For example, no two threads arrive at the same time. By saying the time, time in this program only progresses on either discrete event (like the arrival of a thread) or 1ms boundaries. Therefore, we are then coding to let it merely work, e.g. in one decision time there will be only one thread to be scheduled.

Partition of work between team members

Zihang Xu: queues operations & FCFS scheduler function & some facility functions, testing and debugging

Xiangyu Ren: all other scheduler functions & P/V functions & other facility functions, testing and debugging

Any specifics/quirks

None.