Project2 report Zhili Hu

Overall, I did all the code parts independently. Therefore, this report does not include the section on collaboration with others.

Overall, I described all the functions required by interface.c in scheduler.c, and the interface.c mainly implemented the SRTF strategy and thread control. Scheduler.h include all global variable and function declarations. Next, I will introduce the overall idea and the main content of each file.

The overall idea is that I created a linked list structure as the carrier of the scheduler, and realized all requirements by manipulating the linked list structure. In general, when a thread is running, the thread is first locked. Then enter all its information (current_time, tid, remaining_time) into the list. The order in the linked list is not important, as all thread information is retrieved with each call. Then let them be suspended according to the condition. The condition I set is when the number of units in the linked list is equal to the number of threads and the current thread is a worker thread. When all threads have been added, the last thread needs to wake up the work thread and suspend itself. To wake up specific thread, I add conditional variable in linked list structure so that I can search the variable and wake the one I need. Work threads need to return the appropriate global time based on current_time. Finally unlock the thread.

First one is file interface.c. It includes init_scheduler(int thread_count), cpu_me(float current_time, int tid, int remaining_time) and end_me(int tid). Ignore io_me() because I do not finish it. Function init_scheduler() includes all the variable that we need to initialize in the whole process like global time. Then is the most important function cpu_me(). It input three information and return the time that this thread works 1 tick. The work process is like I describe above. Special situation like when remaining time is 0, it will return global time immediately and do nothing. Function end_me will remove the information in linked list then wake up the next thread that should work. It also corrects the variable of number of threads already in scheduler and should in scheduler.

Then is scheduler.c. Her are many functions to realize all the requirement above. They are int srtf_add(float current_time, int tid, int remaining_time), int choose(void), int wake(int tid), pthread_cond_t* getlock(int tid), int rem(int tid). srtf_add() is a function to add the information to the linked list. Here are three situations. When list is empty, when there are other threads in list and the thread is already in the list. When add success, the function returns 0, else return -1. Function choose(). It returns the tid that should be the work thread when the function is called. Function wake(). It is a function to wake up the given tid in the linked list. If it succeeds, returns 0, else returns -1. It is always used with function choose() when trying to find a work thread and wake it. Function *getlock(). It input a tid and returns a pointer of pthread_cond_t. It is the conditional variable of the input tid. This function normally uses when thread need to be suspended.

Finally scheduler.h. This file is a normal file that work for scheduler.c, with all the functions and global variables in scheduler.c. It also includes a linked list structure called threadControler, with related information, a pointer of conditional variable and the link pointer next.

Describe some problems and solutions I encountered in the process of implementing the code. In the linked list structure, in order to accurately wake up the target thread, a conditional variable attribute is added. But when adding new thread information into the list, it is found that all conditional variables are duplicated. The solution is to pass unique objects between functions by using pointers. Another problem is that in cpu_me, using the created function wake() doesn't seem to be able to directly wake up the suspended thread. The solution is to wake up directly after finding the conditional variable pointer

through the function. However, in the end_me function, wake() can wake up the target. The detailed process is not understood.

Regarding the srtf strategy, my implementation method is to ensure that the awakened thread is the thread with the highest priority every time a thread is awakened. So it can provide the time that global_time should adjust and return to cpu_me.