

Our code cannot be compiled with “make SCHED=MLFQ”, please change the scheduler in the source code. We post this problem on piazza but it is still can’t get fixed.

#### API Logic:

There are several global variables in our pthread library. For example, head1 means the thread queue with highest priority, and main function context and scheduler context. We have a pointer “currentRunning” that points to the current running thread and another pointer “currentRLL” points to the thread queue head which current running belongs to. For instance, if thread#5 is running, and its location is at head3, currentRLL would points to head3. We also have a timer for switch context if a thread has used all its time quantum.

#### Pthread\_create:

If this is the first time main invokes pthread\_create, we would initialize everything to make sure every queue can be accessed, every context is stored, and allocate a new threadControlBlock node and then insert it into the thread pool.

#### Pthread\_yield:

Stop the timer, mark the currentRunning’s status to be yielded, and then swap the context from currentRunning to scheduler context.

Pthread\_exit:

Stop timer first.

Check if main called join on currentRunning thread before, then we would find main thread from the blocked thread queue. (we mark a thread to be blocked if it calls join on other thread, but other thread has not exited. We save this information as an attribute named “so\_wait” in TCB). If main called join on currentRunning thread before, and we found main thread, we insert it back to thread pool, so that it can be scheduled in the future. At last we modify the currentRunning’s TCB “so\_wait” to -1 (for join function). Now we save the parameter value passed by caller, and change currentRunning thread’s status.

At last we put currentRunning thread to the end of exited thread queue (we store all threads that exited here). Set context to scheduler context.

Pthread\_join:

When we just get into join, we would stop the timer using our own helper function “stopTimer” because it is not reasonable to have the timer run in this function.

Then, we will first check whether there is at least one thread that was exited (exited list is not empty). If there is one, we will traverse the whole exited list to check whether the desired thread has exited. If it has exited, we start the timer because it would not be context switched into scheduler (would

not go into the while loop in “else” because it does not need to wait) and would not get the timer re-invoked. Else, we first set current running thread’s status to blocked, delink it from the job queue and add it into the blocked list. Then, we would then traverse the job queue to find the thread that is being waited. Once we found that thread, we set its `so_wait` attribute (by default -1) to the id of the thread that is waiting for it. Then, we go into the while loop that would not go out until the thread that being waited called exit and exit function change it `so_wait` attribute back to -1. Then, we can do the rest of the job such as delink, free, set return value.etc

Thread Synchronization:

We did not do this part, but our code works for most benchmark cases.

Scheduler:

We periodically invoke scheduler by the timer function (set to be 5 usec).

PSJF:

We only use one thread queue for this scheduler.

If `currentRunning` thread is exited, we just schedule a new thread to be `currentRunning` thread (`head1->next` in this scheduler), change its status to running, start timer count, and setcontext to be `currentRunning` thread’s context.

If not, increase `currentRunning` thread’s number of time slice used by 1.

This means currentRunning thread has run out a slice of time quantum. Reinsert it to the proper index in the head1 thread queue based on its number of time slice used (more time slices used means lower priority). Schedule new currentRunning thread (head1->next in this scheduler), change its status and set setcontext to be currentRunning thread's context.

MLFQ:

We have four thread queues for MLFQ, head1, head2, head3 and head4.

If currentRunning thread is exited, we just schedule a new thread to be currentRunning thread (choose the thread with lowest priority from four queues), change its status to running, start timer count, and setcontext to be currentRunning thread's context. If all threads are in the same level, run as Round Robin.

If thread is yield, we don't demote this thread's priority.

If currentRunning thread is not blocked, insert currentRunning to the proper location based on its number of time slices used.

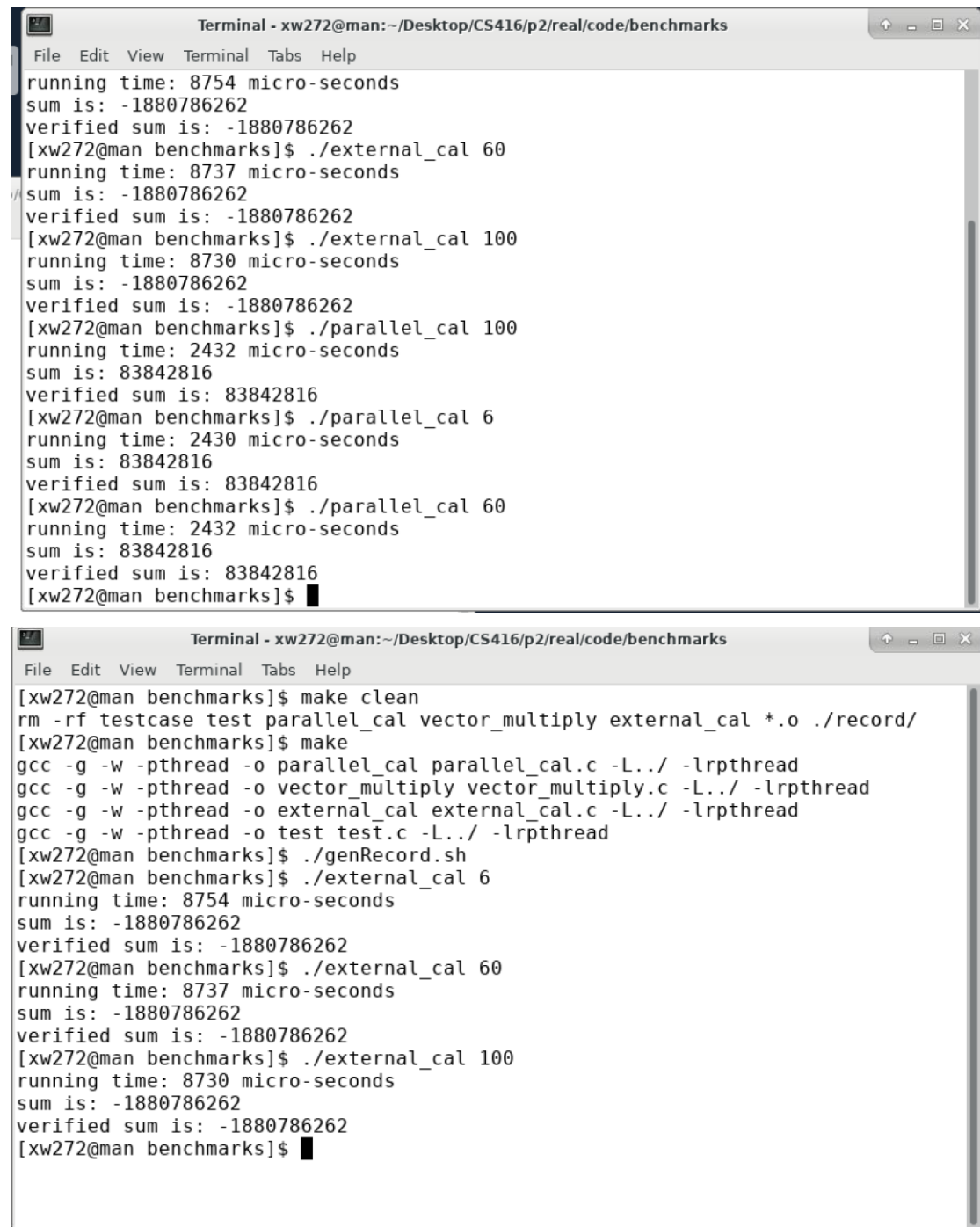
At last, schedule a new job, change its status, start the timer, and setcontext to new job's context.

## MLFQ Result:

```
[jd1216@ilab2 benchmarks]$ ./external_cal 6
running time: 9292 micro-seconds
sum is: -296024502
verified sum is: -296024502
[jd1216@ilab2 benchmarks]$ ./parallel_cal 6
running time: 3138 micro-seconds
sum is: 83842816
verified sum is: 83842816
[jd1216@ilab2 benchmarks]$ ./vector_multiply 6
running time: 13 micro-seconds
res is: 631560480
verified res is: 631560480
[jd1216@ilab2 benchmarks]$
```

```
[jd1216@ilab2 benchmarks]$ ./genRecord.sh
[jd1216@ilab2 benchmarks]$ ./external_cal 100
running time: 9176 micro-seconds
sum is: -689621421
verified sum is: -296024502
[jd1216@ilab2 benchmarks]$ ./external_cal 100
running time: 9183 micro-seconds
sum is: -296024502
verified sum is: -296024502
[jd1216@ilab2 benchmarks]$ ./parallel_cal 100
running time: 3135 micro-seconds
sum is: 83842816
verified sum is: 83842816
[jd1216@ilab2 benchmarks]$ ./vector_multiply 100
running time: 35 micro-seconds
res is: 631560480
verified res is: 631560480
[jd1216@ilab2 benchmarks]$
```

PSJF:



The image contains two terminal window screenshots. The top window shows the execution of three benchmark programs: `external_cal` with values 60 and 100, and `parallel_cal` with values 100 and 60. Each execution displays the running time in microseconds, the calculated sum, and a verified sum. The bottom window shows the compilation of these programs using `gcc` with `-w`, `-pthread`, and `-lrpthead` flags, followed by running a `genRecord.sh` script and then the same three benchmark programs as the top window.

```
Terminal - xw272@man:~/Desktop/CS416/p2/real/code/benchmarks
File Edit View Terminal Tabs Help
running time: 8754 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ ./external_cal 60
running time: 8737 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ ./external_cal 100
running time: 8730 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ ./parallel_cal 100
running time: 2432 micro-seconds
sum is: 83842816
verified sum is: 83842816
[xw272@man benchmarks]$ ./parallel_cal 6
running time: 2430 micro-seconds
sum is: 83842816
verified sum is: 83842816
[xw272@man benchmarks]$ ./parallel_cal 60
running time: 2432 micro-seconds
sum is: 83842816
verified sum is: 83842816
[xw272@man benchmarks]$ █

Terminal - xw272@man:~/Desktop/CS416/p2/real/code/benchmarks
File Edit View Terminal Tabs Help
[xw272@man benchmarks]$ make clean
rm -rf testcase test parallel_cal vector_multiply external_cal *.o ./record/
[xw272@man benchmarks]$ make
gcc -g -w -pthread -o parallel_cal parallel_cal.c -L../ -lrpthead
gcc -g -w -pthread -o vector_multiply vector_multiply.c -L../ -lrpthead
gcc -g -w -pthread -o external_cal external_cal.c -L../ -lrpthead
gcc -g -w -pthread -o test test.c -L../ -lrpthead
[xw272@man benchmarks]$ ./genRecord.sh
[xw272@man benchmarks]$ ./external_cal 6
running time: 8754 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ ./external_cal 60
running time: 8737 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ ./external_cal 100
running time: 8730 micro-seconds
sum is: -1880786262
verified sum is: -1880786262
[xw272@man benchmarks]$ █
```

```
Terminal - xw272@man:~/Desktop/CS416/p2/real/code/benchmarks
File Edit View Terminal Tabs Help
verified sum is: 83842816
[xw272@man benchmarks]$ ./parallel_cal 6
running time: 2430 micro-seconds
sum is: 83842816
verified sum is: 83842816
[xw272@man benchmarks]$ ./parallel_cal 60
running time: 2432 micro-seconds
sum is: 83842816
verified sum is: 83842816
[xw272@man benchmarks]$ ./vector_mul 6
bash: ./vector_mul: No such file or directory
[xw272@man benchmarks]$ ./vector_multiply 6
running time: 12 micro-seconds
res is: 631560480
verified res is: 631560480
[xw272@man benchmarks]$ ./vector_multiply 60
running time: 44 micro-seconds
res is: 631560480
verified res is: 631560480
[xw272@man benchmarks]$ ./vector_multiply 100
running time: 37 micro-seconds
res is: 631560480
verified res is: 631560480
[xw272@man benchmarks]$
```

### Kernal vs User Thread Analyze:

User level MLFQ thread: external 100: 9176ms; parallel 100: 3135ms;

vector 100: 35ms

User level STCF thread: external 100: 8730ms; parallel 100: 2432ms;

vector 100: 37ms;

Kernel level thread: external 100: 3582ms; parallel 100: 513ms; vector 100:

439ms;