Names: Craig Li, Prerak Patel
NetIDs: craigli, pjp179
Ilab Server: rm.cs.rutgers.edu

MyPthread User-Level Thread Library and Scheduler

Introduction:

For our second project in CS416 - Operating Systems, we were to implement a user level thread library that replicated the functions of the POSIX pthread library. As opposed to the pthread library, which can make usage of multiple CPUs, a user level library instead uses one CPU, and interleaves each thread with each other in execution based off of a given quantum, for which we chose 10ms. This quantum would be used in conjunction with a signal handler and timer to constantly notify our library to switch threads, giving each thread an equal opportunity to run, depending on the given scheduling algorithm. Memory and Register values could be swapped between each of these threads by using contexts to save and resume each thread exactly where they had left off. Constantly switching between each thread would mimic a truly concurrent multithreaded process.

Implementation details of API Functions and Scheduler:

1. **mypthread_create()**

   Our implementation of pthread_create() involved creating a thread control block using contexts, involving a struct containing all necessary information that a thread might have, such as a thread ID, status, context, thread IDs that this thread would be waiting to finish, number of elapsed quantums (used for PJSF), as well as the location and contents of a return value. The first call to mypthread_create() would generate a thread control block and context for the main function, as well as initialize and start a timer that would expire every quantum. Threads would also be allotted a section of memory designed as a stack for that given thread, and would have their contexts linked to the desired function. From there, any new threads that were not the main function were inserted into a priority queue, which would implement preemptive shortest job first by assigning a priority to a thread based on how much time it had already spent running. Lower times would have a higher priority in the queue, while higher run times would suffer, and the corresponding TCBs would be pushed back in the queue.

2. **mypthread_yield()**

   The intent behind mypthread_yield() was to swap out the currently running thread to the next one in the queue. Since our implementation removes the currently running thread from the queue while it is executing, we can instead call schedule(), which instantly pauses the current thread, changes its status to READY, puts it back into the queue, after which the next thread in the priority queue is dequeued and run.

3. **mypthread_exit()**

   When mypthread_exit() is called, we mark the currently running thread's status as FINISHED, after which we check for return values from the given thread to return to our main function. After marking the thread as FINISHED, we also use a helper function (removeNode()) to search our priority queue to find the thread that is finished, and deallocate its stack, and any other dynamically allocated memory. This thread is then removed completely from the queue. After it is removed, we account for any threads that may have been waiting on this thread to finish, using a helper function (notifyThreads()) to find any threads that are in our queue that were waiting for this thread to finish, which is obtained by finding any threads in WAITING status, and comparing the thread ID that in the waitingThread attribute of its thread control block to the

thread ID of the finished thread. At this point, those threads are marked READY and can be resumed when they are selected next in the queue.

4. **mypthread_join()**

   mypthread_join() first checks if the argument thread is finished through the usage of a helper function (isFinished), which checks the status of the given thread. If the given thread is finished, we can both remove it from the queue, as well as store its return value in a waiting thread's control block. If the thread is not finished, we instead change the calling thread's status to WAITING, and set the waitingThread attribute to the argument thread's ID, which allows us to be notified when that thread exits so that we can resume.

5. **mypthread_mutex_init()**

   mypthread_mutex_init() creates our mutex lock for our thread library, involving a struct with a pointer to a thread control block, and a int value to represent whether it is locked or unlocked. The pointer to the thread control blocks is a list of threads that are waiting on the mutex to be released from another thread's control. This is further discussed in mypthread_mutex_lock(). When mypthread_mutex_init() is called, it first checks if the main context and timer exists, and implements them if neither exist. Then our pointer to our waiting thread control blocks is initialized to NULL, and we initialize our lock to 0 (unlocked).

6. **mypthread_mutex_lock()**

   Our implementation of mypthread_mutex_lock() involves the usage of the atomic_flag_test_and_set function from the stdatomic.h library to test the mutex. If the mutex is not acquired, it is added to the list of thread control blocks that are waiting to acquire the mutex, and the respective thread is marked as WAITING, from there, our scheduler is called, and the WAITING node is put back into the queue, waiting until the mutex is unlocked.

7. **mypthread_mutex_unlock()**

   To unlock the mutex, we mark all of the nodes that were waiting for the lock as READY from WAITING, which signals to the scheduler that they can be run again. Our implementation then frees the dynamically allocated memory that was created for each mutex node in its linked list. Finally, our mutex is reset to have no waiting list, and to be unlocked.

8. **mypthread_mutex_destroy()**

   As we take care of dynamically allocated memory for our mutex nodes in mypthread_mutex_lock() and mypthread_mutex_unlock(), this function does not necessarily have a purpose, and as such is essentially left blank.

9. schedule()/sched_stcf()

Our scheduler was based on Preemptive Shortest Job First, which involved measuring the given amount of quantum time periods that a given thread had run, and assigning queue priority based off of that. Each thread control block has an elapsedQuantums attribute that records the number of quantums that a given thread has run. Each time schedule() is called, it redirects to shed_stcf(), where elapsedQuantums for the currently running thread is incremented, and it is placed back in the queue. The scheduler then dequeues the next node in the queue (which is already sorted by priority), and sets that thread to run for a quantum by using swapcontext() between the thread that was running, and the thread that had just been dequeued. This saves the context of the previous thread, while loading the context of the dequeued thread, allowing it to resume from where it was last interrupted.

Our scheduling algorithm is based off of a timer, which was implemented using setitimer(). Based off of a real world (ITIMER_REAL) timer that decrements from our QUANTUM value (10ms), the timer generates a SIGALRM every time our quantum elapses. To prevent program disruption, we utilized the sigaction struct to acknowledge the SIGALRM and redirect the current thread to the schedule() function. This ensures that each time our timer elapses, we are sent directly to the schedule() function, where our currently running thread is swapped out for the next one in the queue.

10. Queue Implementation

Our priority queue implementation is fairly standard, with helper functions such as enqueue(), dequeue(), getTCB(), getQueueSize(), printThreadQueue(), and removeNode(). To enqueue a thread control block, a node is created within the enqueue() function, whose elapsedQuantums are iteratively compared to each node's elapsedQuantums. The node to enqueue is then inserted based on its elapsedQuantums (lower elapsedQuantums are given higher priority). The dequeue() function merely changes the pointer to the head of the queue based on the first node found that has a READY status, which is then scheduled and run through schedule(). Nodes with other statuses such as WAITING are ignored, as they are depending on other threads to exit. Since we instantly remove nodes with a REMOVE or FINISHED status, there should only be READY or WAITING nodes in the queue.

Benchmark Results:

To test our thread library for correctness, we tested each of the given benchmark functions with a varying quantity of threads. To be exact, we tested both our thread library and the POSIX thread library (as a reference) within a set of [1, 2, 5, 10, 25, 50 ,75, 100] threads. As specified in class, we would not be tested on more than 100 threads, so we decided to test a varying amount within the window of 0 to 100. Since the major difference of our thread library and the POSIX library is that the POSIX library has true concurrency, and is able to take advantage of multiple cores to use its multithreading functions, but our library implements an interleaved system that is decided based on both our scheduling algorithm and chosen QUANTUM. The benchmarks themselves are included towards the end of this report, but the main differences are that our thread library is slower, which is to be expected given the nature of our "concurrency". Our tests for parallel_cal averaged at around 6200 μs (microseconds), while the POSIX

library generally took an average of around 2500 µs. Our library did not fare as well on the vector_multiply tests either, though the difference was much smaller due to the incredibly fast runtime of that program. Our library averaged about 70 µs across our tests, and the POSIX library averaged around 40 µs. What was most interesting is the external_cal tests, which generally were the same between both libraries at an average of around 530 µs.

Challenges in Implementation:

I will say that this project as written may be incredibly dependent on how much the programmer understands the functions of makecontext(), getcontext(), and swapcontext(). I (Craig) personally had a very hard time trying to understand their respective man pages, and implementation for those functions initially went poorly. Using the get/set/makecontext functions did not provide a positive debugging experience, as tracing memory leaks, and stack smashing errors was not fun. However, understanding the key concepts behind concurrency and threads ultimately proved to illuminate a different path towards completing the project. While the project writeup states that we can create a context for a scheduler, the thought of why we would need a constant link to the schedule only illuminates the fact that we can get by without it (which provides much easier implementation). The scheduler context is originally tied to the schedule() function via makecontext(&schedulerContext, schedule, 0). However, if we always need a context that starts from the schedule() function, why not just call the schedule() function itself? By calling the schedule() function at the end of any function that required the scheduler, the current executing context would change to the scheduler, and then resume later at the end of that function, with its context not having changed. Once we eliminated the use of these functions as much as possible (swapcontext and getcontext are definitely needed to create/change threads), the implementation became easier.

Possible Improvements in Implementation:

As discussed in the prior section, we did not implement makecontext(), getcontext(), and setcontext() as much as the instructor may have intended. While one of our TAs (Jian) specified that it was not necessary to create a scheduler context, most of the answers that were given on piazza tended to lean towards the usage of these functions. While our program works, whether this is against the spirit of the assignment is up to the instructor. As for more concrete improvements, due to the impending due date, as well as the initial difficulty of implementation of this project, while our results are satisfactory, I know that not all of our external_cal results were correct, which may indicate a flaw in our mutex design. While our test suite shows that we created the correct output, a small percentage of the time, we will obtain a different answer from the verified answer, much to our consternation.

Names: Craig Li, Prerak Patel
NetIDs: craigli, pjp179
Ilab Server: rm.cs.rutgers.edu

MyPthread User-Level Thread Library and Scheduler

Parallel_cal Test Suite (MyPthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 1
==1355993==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6282 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 2
==1356291==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6219 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 5
==1356417==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6210 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 10
==1356556==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6215 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 25
==1356725==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6238 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 50
==1357105==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6223 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 75
==1357246==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6212 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 100
==1357407==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 6214 micro-seconds
sum is: 83842816
verified sum is: 83842816
```

Parallel_cal Test Suite (POSIX pthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 1
running time: 2568 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 2
running time: 2569 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 5
running time: 2669 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 10
running time: 2573 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 25
running time: 2572 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 50
running time: 2575 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 75
running time: 2580 micro-seconds
sum is: 83842816
verified sum is: 83842816
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./parallel_cal 100
running time: 2577 micro-seconds
sum is: 83842816
verified sum is: 83842816
```

Names: Craig Li, Prerak Patel
NetIDs: craigli, pjp179
Ilab Server: rm.cs.rutgers.edu

MyPthread User-Level Thread Library and Scheduler

External_cal Test Suite (MyPthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 1
running time: 532 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 2
running time: 532 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 5
running time: 545 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 10
running time: 526 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 25
running time: 528 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 50
running time: 552 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 75
running time: 532 micro-seconds
sum is: -60089584
verified sum is: -60089584
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 100
running time: 533 micro-seconds
sum is: -60089584
verified sum is: -60089584
```

External_cal Test Suite (POSIX pthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 1
running time: 532 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 5
running time: 553 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 10
running time: 532 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 25
running time: 546 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 50
running time: 544 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 75
running time: 530 micro-seconds
sum is: -828096543
verified sum is: -828096543
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./external_cal 100
running time: 536 micro-seconds
sum is: -828096543
verified sum is: -828096543
```

Names: Craig Li, Prerak Patel
NetIDs: craigli, pjp179
Ilab Server: rm.cs.rutgers.edu

MyPthread User-Level Thread Library and Scheduler

Vector Multiply Test Suite (MyPthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 1
==1363608==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 46 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 2
==1363653==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 46 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 5
==1363674==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 48 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 10
==1363715==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 50 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 25
==1363770==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 58 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 50
==1363812==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 122 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 75
==1363884==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 108 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 100
==1363912==WARNING: ASan doesn't fully support makecontext/swapcontext functions and may produce false positives in some cases!
running time: 81 micro-seconds
res is: 631560480
verified res is: 631560480
```

Vector_multiply Test Suite (POSIX pthread library)

```
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 1
running time: 33 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 2
running time: 33 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 5
running time: 37 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 10
running time: 37 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 25
running time: 44 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 50
running time: 69 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 75
running time: 61 micro-seconds
res is: 631560480
verified res is: 631560480
craigli@rm:~/CSMaster/CS416/Projects/Project2/code/benchmarks$ ./vector_multiply 100
running time: 54 micro-seconds
res is: 631560480
verified res is: 631560480
```