CSCI 5103 Project 1 Writeup

Anlei Chen & Shuo Jiang

1. For the grader:

The project can be compiled using `make` or `make all` in the top-level submission directory. These commands will compile the library (TCB.cpp, uthread.cpp), pi.cpp, and test.cpp (our testing file). Two executables will be created: pi and test, which can be run using `./pi arg1 arg2` and `./test`, respectively.

2. Assumptions:

For each uthread function and the TCB constructor, any assumptions or behaviors that differ from (within the scope of the project) the corresponding POSIX pthread implementation will be listed here under their respective functions:

TCB::TCB:

- The main thread will not have a stack.
- The provided join entry and finish entry data structures were merged into the TCB class.
- By default, _retval, _join_id, and _quantum are set to NULL, -1, and 0, respectively.

uthread_init:

- The main thread will have a tid of 0.
- The interrupt timer will be started.

uthread_create:

- If the total number of threads in the library exceeds the maximum (defined in uthread.h), a new thread will not be created, and an error will be returned.
- The thread ID will be assigned arbitrarily based on the total number of threads created during the library's lifetime.
- Otherwise, the behavior follows the POSIX pthread_create implementation.

uthread_join:

- A thread cannot join with itself, and attempting to do so will return an error.
- If the library fails to switch to another thread, the current thread will resume and return an error.
- Otherwise, the behavior follows the POSIX pthread_join implementation.

uthread_yield:

- The calling thread will automatically be added to the ready queue.
- On error (get/setcontext fails), the calling thread will be resumed (i.e., chosen to run again).
- The quantum is increased before the actual thread switch.
- Otherwise, the behavior follows the POSIX pthread_yield implementation.

uthread_exit:

- If the calling thread is the main thread, it will be deleted, and the program will terminate. No other cleanup steps will be performed.
- If another thread is waiting to join the calling thread, it will be unblocked.
- Otherwise, the behavior follows the POSIX pthread_exit implementation.

uthread_suspend:

- The calling thread may suspend itself. Doing so will move the thread to the block queue, and another thread will be scheduled to run.
- If the target thread is already in the block queue, the function does nothing.
- Any thread may suspend another thread.

uthread_resume:

- This function searches the block queue for the specified thread ID.
- Any thread may resume another thread.

uthread_self:

- The behavior follows the POSIX pthread_self implementation.

uthread_once:

- The init_routine will run under the library's signal mask.
- Otherwise, the behavior follows the POSIX pthread_once implementation.

uthread_get_total_quantums:

- Quantums represent the number of times a thread was scheduled to run, including voluntary yields.
- A thread that was created, ran, and exited within the same quantum will have a quantum count of 1 (note: the TCB will have been deleted).
- The total quantum count includes the sum for all threads, including those that have finished, been joined, or been deleted.

uthread_get_quantums:

- This function searches each queue for the specified thread ID. If a thread has been successfully joined, the function will fail.

3. Additional APIs:

No additional APIs are made available to the user. However, several library functions were added to simplify the library code and reduce code duplication. The library maintains three queues: a READY queue, a BLOCK queue, and a FINISH queue. Each queue is a double-ended queue of TCB pointers. A given thread TCB can exist in only one queue at a time. The READY queue contains threads to be scheduled, the BLOCK queue contains threads that have been suspended or are waiting to join, and the FINISH queue contains threads that have finished execution (via uthread_exit). As a result, the queue management functions were adjusted to work with any arbitrary queue. The library also includes a function, threadSwitch(), which handles performing context switches. uthread_yield() calls this function to perform a context switch. Apart from these changes, no other additions were made.

4. Thread entry:

On thread creation, the calling thread's context will be saved into the new TCB using getcontext(). A stack will be allocated on the heap using posix_memalign() with a 16-byte alignment and a size defined in uthread.h. makecontext() will be called, starting at the thread stub with the start routine and arguments provided as parameters to the stub. When the stub is called (i.e., the thread has been chosen to run), interrupts will be enabled to ensure the user code doesn't run with the library's signal mask. The thread will enter the start routine, and if the thread returns from the start routine, uthread_exit() will be called for the thread with the return value. During the creation of the thread, makecontext() is responsible for setting up the execution context for the newly created thread. When thread_create() is called, makecontext() allocates the stack for the new thread and modifies the context to start at the stub.

5. Time slice and performance:

The length of the time slice (quantum_usecs) directly affects the responsiveness and efficiency of the program. Running a program with a short time slice increases the frequency of context switches, allowing every thread to make some progress. However, as a trade-off for this concurrency, the overhead of switching contexts can slow the overall performance of the program. The CPU will spend more time on bookkeeping and transitioning between threads instead of executing thread work. That being said, under our testing with the Pi program, changing the quantum had very little effect on the execution time.

6. Critical Sections:

All library functions, except uthread_self() and uthread_get_total_quantums(), will operate with SIGVTALRM blocked. This is to ensure the expected behavior of the uthread functions, especially for critical functions such as uthread_create(), uthread_exit(), uthread_join(), and uthread_yield(). Other functions contain critical sections that may not be strictly necessary but are nice to have to ensure expected behavior.

7. Test Cases:

We created eight test cases, testing every function at least once. The main purpose of these test cases is to provide a program that can be manually observed step-by-step to confirm proper library behavior. As a result, there are no predefined correct results to test against. All testing programs (test.cpp and pi.cpp) were tested with Valgrind and stepped through using GDB when necessary. The final submission does not include the debug print statements used during testing, in order to improve readability. The version of the library with the debug statements can be found on our GitHub in the project1 directory.

8. Source code:

The library code, Makefile, and test cases have been submitted with the write-up. In case any files are missing, they can be found on our GitHub. The submission directory contains all the necessary files for submission, with all debug statements removed.