**CS518 - A0: Uthreads!**

Group Members: Jay Patil (jsp255), Aishwarya Harpale (ach149)

**OVERVIEW**

The aim of this assignment is to create a user-level thread library which resembles the Linux’s POSIX thread library. Further, as a part of the implementation, we have also created user-level mutexes to handle mutual exclusion. In order to schedule the created threads, we have implemented three scheduling policies namely Round Robin scheduling, Multilevel Feedback Queue (MLFQ) scheduling and Pre-emptive Shortest Job First (PSJF) scheduling. The result of this assignment is a user-level thread library named ‘mypthread’ which enables users to create multiple threads and mutexes just like the actual pThread library. User also has the ability to select the scheduling policy to schedule the jobs.

**API**

1. **Threads:**

* **mypthread\_create**

Thismethod is used to create a new mypthread. When invoked, it creates a new Thread Control Block (tcb) and this tcb is added to the front of the scheduling queue.

* **mypthread**\_**yield**

When this method is invoked, the current running thread stores its current context and surrenders its runtime so that other threads can run.

* **mypthread**\_**exit**

This method is used to terminate a thread and later it is also freed from memory.

* **mypthread\_join**

When this method is called it checks the current queue for the thread passed as argument. If the target thread is found and is terminated, this method returns immediately. Otherwise, it waits for the target thread to terminate.

1. **Mutexes:**

* **mypthread\_mutex\_init**

This method is used to initialize the mutex.

* **mypthread\_mutex\_lock**Whenever this method is called, the current thread tries to set the atomic flag so that it can acquire the mutex. If it is able to acquire it, the status of current thread changes to mutex lock, indicating it is holding the mutex. If it is unable to acquire the lock, it is added to the queue in a blocked state.
* **mypthread\_mutex\_unlock**

This method is used to clear the atomic flag so that the mutex is available for other threads to use. Also, the blocked threads are dequeued and added to the running queue.

* **mypthread\_mutex\_destroy**

When this method is invoked, all the blocked threads are dequeued and the memory is deallocated of each blocked thread.

1. **Scheduler:**

* **schedule**

This is the driver function to decide which scheduling policy is to be used for scheduling the threads.

* **sched**\_**RR**

This scheduling method checks for the current thread. If it has finished its task, it is pushed out of the queue and the memory for the same is deallocated. If it has not completed its task, it sends the current thread at the last of the queue and takes the next waiting thread in the queue for scheduling.

* **sched\_MLFQ**

In this method if the current thread has yielded it is sent to the back of running queue. All completed threads are removed from the running queue. All the other threads are pushed to the lower running queues. The thread at the front of running queue is scheduled then. This method also checks the time elapsed till now and if the time elapsed is greater than equal to a threshold, it calls a maintenance cycle to push all the threads of queue to front one by one. It then resets the time counter.

* **sched\_PSJF()**

In this method, the thread with shortest time remaining is scheduled. Quantum count determines which thread will have highest priority. So, the thread with lowest quantum count or highest priority will be scheduled first.

1. **Structures:**

* Thread Control Block

struct threadControlBlock {

mypthread\_t threadID; // Thread ID

int status; //READY=0, WAIT=1, JOIN=2, YIELD=3, EXIT=4, MUTEX\_LOCK=5

void \*jVal; // Joining Thread ID

void \* retVal; //Return value

ucontext\_t \*context; // Context

struct list \*joinQueue; // Queue to implement join operation

int quantumCount; // used for MLFQ scheduling

} tCB;

* Mutex Metadata

struct mypthread\_mutex\_t {

int locked; // To check if the mutex is locked

int available; // To check if the mutex is free

int holder; // Thread which currently has the lock

int initialized; // To check if it is initialized

int ll \*queue; // Mutex queue

} mypthread\_mutex\_t;

* Linked List

We have used linked list data structure to implement queue instead of an array.

struct list {

tCB \*thread; // Holds the thread ID of the thread

struct list \*next; // pointer to the next node in list

} ll;

**PERFORMANCE**

|  |  |  |
| --- | --- | --- |
| **Benchmark** | **Time taken by ‘mypthread’ library (in µs)** | **Time taken by POSIX thread library (in µs)** |
| external**\_cal** | **383** | **374** |
| vector\_multiply | **36** | **32** |
| parallel\_cal | **1909** | **1868** |

The quantum we have used to test mypthread library against the benchmark is 10µs, as the other quantum value chosen were giving poor response time.

**ANALYSIS**

The ‘mypthread’ library performance is not as good as the Linux POSIX thread library. The scheduling policy used in ‘mypthread’ library do not have impact on the performance. As the thread count increases, the response time for benchmarks improves. For the benchmark ‘external\_cal’ ‘mypthread’ outperforms the Linux thread library slightly when the number of threads is increased to 100.