Project 2: MyPthread User-level Thread Library and Scheduler: Understanding Scheduling & Synchronization Complexity Design

Paul Kim
NetID: obawolo
CS416 OPERATING SYSTEM 01

December 18th 2020: tested from 'cp.csrutgers.edu'

Introduction: Thread

Program under execution via main memory is process. In a process, threads are used for its cpu utilization. E.g.) 1 thread = 1 processor usage. 2 threads = 2 processors usage. And so on. Associated with a program, multiple process exists. And a process can have multiple threads executing in it. Each Thread is comprises of Thread ID, Program Counter (PC), a register set, Stack. Within a same process, threads share code, data (static and global variables), and other operating system resources, such as open files and signals. Through multiple threads, a process can execute multiple tasks at a time. Thus, more throughput.

Resources that I've used for my understanding of thread and pthread:

1. https://www.geeksforgeeks.org/thread-in-operating-system/
2. https://www.geeksforgeeks.org/multithreading-c-2/
3. https://www.youtube.com/watch?v=ynCc-v0K-do
4. https://www.youtube.com/watch?v=1ks-oMotUjc
5. https://www.youtube.com/watch?v=GXXE42bkqQk

FUNCTIONS IMPLEMENTATION

1. enqueue
   FIFO implementation function that adds 'insert' to existing list 'q'. If queue isn't created, then queue is created with 'malloc' function with size of list.

2. dequeue
   FIFO implementation function that removes the front element of the queue by using 'free' function.

3. l-insert
   linked list implementation function that inserts new node 'jThread' to front of the linked list 'q'. If a linked list doesn't exist then a new linked list is created.

4. l-remove
   linked list implementation function that removes the first node of the list 'q'.

5. thread-search
   Thread search function that searches a thread with given ThreadID 'tid' via 'allThreads' list table where all threads found with TCB. If found, it returns the index number of 'allThreads'. If not, then it returns 'NULL'.

6. initializeQueues
   Initialization function that default starts 'runningQueue' and 'allThreads' lists. All elements of Both 'runningQueue' and 'allThreads' are initially set to 'NULL'.

7. maintenance
   'runningQueue' list management function that inverts all threads inside the 'runningQueue' status set to READY (0) state.

8. garbage-collection
   Management function that handles exiting thread: supports invoked/non-invoked pthread-exit call.

9. initializeMainContext
   All threads Context startup. All threads connects and starts from a thread called 'mainThread'.

10. initializeGarbageContext
    Thread cleaner startup. Sets all elements of 'runningQueue' to NULL. Initialize the garbage collector.

11. scheduler
    Signal handler to reschedule upon VIRTUAL ALARM signal. Current running thread is not in the scheduler. It is in the 'runningQueue', but it's always in the 'allThreads'. Once the timer finishes, the value of it-value.tv-usec will reset to the interval time.
    New currentThread is fetched from dequeue'ing from the 'runningQueue' and executes following depending on the status of 'currentThread':

(a) READY(0): signifies that the current thread is in the running queue

(b) YIELD(1): signifies pthread yield was called. The priority doesn't updates.

(c) WAIT(2): do nothing

(d) EXIT(3): exit the currentThread and frees exited thread's thread control block and ucontext control. Reset the timer.

(e) JOIN(4): corresponds with a call to pthread-join.

(f) MUTEX-WAIT(5): corresponds with a thread waiting for a mutex lock.

12. mypthread-create
Creates a new thread. The new thread has all Thread Control Block struct members into appropriate values: priority = 0, context = function argument, status = READY, etc.

13. mypthread-yield
Simply updates the 'currentThread' status to YIELD(1).

14. mypthread-exit
Calls the garbage collection.

15. mypthread-join
Searches and fetch the given TCB argument via 'thread-search'. The 'currentThread' is insert to the TCB's thread stack which makes the 'currentThread' status updates to Join(4).

16. mypthread-mutex-init
Initializes the given mutex argument for the mutex lock. The mutex's variable 'holder' represents the tid of the thread that is currently holding the mutex.

17. int mypthread-mutex-lock
Initially, assume that mutex is initialized. Then operates 'the atomic-test-and-set'. The reason why I reset notFinished to 1 is that when coming back from a swapcontext, notFinished may be 0 and I can't let the operations in the loop be interrupted. notFinished need to be 0 before going to scheduler. The 'currentThread' priority gets inversed.

18. mypthread-mutex-unlock
Errors can occur: unlocking an open mutex, unlocking mutex not owned by calling thread, or accessing unitialized mutex. Initially, assume that mutex-¿available will be initialized to 0 by default without calling init. Available in this case means that mutex has been initialized or destroyed (state variable). 'muThread' gets 'dequeued'

BENCHMARK OUTPUT:

1. ./external-cal 6

```
running time: 420 micro-seconds
sum is: 504180096
verified sum is: 504180096
```

2. ./external-cal 60

```
running time: 428 micro-seconds
sum is: 504180096
```

3. ./vector-multiply 6

```
running time: 47 micro-seconds
res is: 631560480
verified res is: 631560480
```

4. ./vector-multiply 60

```
running time: 95 micro-seconds
res is: 631560480
verified res is: 631560480
```

DIFFICULTIES

Book keeping the timer for the time quantum of each thread in schedule function was very confusing. I originally was going to use 3 states for the thread status implementation but that got me confused and I had to started over and implemented with 6 states: READY(0), YIELD(1), WAIT(2), EXIT(3), JOIN(4) MUTEX-WAIT(5).

IMPROVEMENT

My parallel-cal testing gives me error.

./parallel-cal 6

```
Killed
```

My program must have some limitation to calculate for that specific calculation which I cannot figure it out due to my inexperienced knowledge of C and time limitation.