Net ID: yp315 | Yash Patel , kks107 | Kev Sharma
CS 416 - Undergraduate OS
Tested on man.cs.rutgers.edu

Report

Authored by Yash Patel

1. Worker_create: Initializes all the necessary variables and data structures when it is first called i.e. queues and list of TCBs also, create context for scheduler. For the first time that the worker creates we need to get the context of the main as it will be a thread that will run when the scheduler selects. We also create new context for a thread that the user creates and for each new worker_create call that the user makes in the future we will create a new thread/tcb for them and put them in the arrival queue.

   Worker_yield: Original logic was just to swap context to the scheduler but we also added time logic as it holds the amount of time that a thread used while it was running as in the future it can be run for how much time that is left for that thread. All time is set at microseconds so every benchmark and timer usage is the same.

   Worker_exit: When worker exit is called by the thread function and it has a non null return value we set the tcb return value to that so if the user in the join asks for the return value we are able to return the exit return value to the user.

   Worker_join: We had to keep track of tcbs that already ended as if the main joins to the thread that is already ended it essentially does nothing so we hold tcbs that ended and if we are joining to a tcb that ended we would just return return_val if the user ask for it. Otherwise if it is a valid running thread we block this thread as it is waiting for the other thread to end running so we can be unblock so we can run again.

   Worker_mutex_init: This function creates mutex and adds it to the mutex list for it to be used by a thread in the future. Stuff that the mutex holds is that mutex id and holder tid. This will be assigned to no thread number initially.

Worker_mutex_lock: checks if the mutex id is held by another thread if it is we are in a while until it is unlocked and in that while loop we swap out to scheduler context for it to select a thread that is not blocked by either join or mutex being lock.

Worker_mutex_unlock: Checks if the right thread is unlocking its thread as another thread should not be allowed to unlock a mutex that it does not hold. If the thread holds the mutex it unlocks it and tells others that are waiting for the thread that it is unlocked so when scheduler selects a thread every thread has a fair chance of getting selected and holding the mutex as they are not blocked.

Worker_mutex_destroy: Checks that the mutex is not held by a thread and if it is using it so we should not try to destroy it. It also checks if the mutex is released then it will release the mutex and free all the allocated memory for the mutex struct.

Scheduler PSJF: Data structure used : Sorted List by tcb->elapsed .The scheduler runs continuously. Initially it will add all the threads in the arrival queue into the sorted list. Then we get the first thread that is unblocked for the list and can be the first node/tcb and schedule that. When we return to the scheduler we add that node back to the sorted list and check for another node again and this continues until all threads are done. Timers are done as a one shot timer so we know that if the timer goes off when a thread uses its full time quantum. We also use the logic in the yield so we give each thread how much time that it has left and our previous statement is correct.

Scheduler MLFQ: Data structure used: Queues. We initialize an array of queues and it is decided by the variable so it can be 4 or 8 the default is 4. We add all threads that are in the arrival queue into the top most priority queue(queue[0]) as when a thread arrives it is at the top most priority. We check that if we are coming back from a thread that ran if true we can add it back to the right priority queue level if a thread used all of its time quantum it will be assigned to the next priority queue level than its previous one. Then after the queue has all the threads we go through priority queue levels to find a thread that is unblocked and if we find one we schedule it. In the scheduler we check if the time elapsed is greater than some constant, which is S, if it has we boost all the threads up to top priority so they can run as the top priority.

2. When we run our thread library with a different number of threads the runtime, response time and turnaround time increases. As this might be the case due to the increase of threads adding more time as there might be some time that a single thread will need to complete its function. If we increase the threads we are increasing the time that each total thread will take so our runtime increases and turnaround time. Our response time increase are due to data structures adding time as it gets larger and searching and looking at the list this increases time as we add more nodes/tcb

3. The default pthread library runs way faster than our library one thing that we can think about is that the default pthread library is using the thread of the cpu efficiently as it can run more threads at the same time than our library that runs a single thread and swaps it out to a different thread when the time is up. Another thing to also note that the data structure and logic that we are using in our library might be adding time to how much time that it takes to finish. So with optimization we can reduce some time from our library.