Duygu Tümer 24842

PA-2 REPORT

In this programming assignment 2, I choose Fetch-and-add Ticket Lock because I know that it is fair, so threads do not starve. Also, this method automatically transfers the control between threads with a given order, so it works one by one with an order.

My locking algorithm as a pseudocode is:

Create a new struct as “ticket_lock_t” which has these variable inside it:

- pthread_cond_t as cond
- pthread_mutex_t as mutex
- unsigned long as queue_head and queue_tail

void ticket_lock(ticket_lock_t *ticket) // create a func for ticket lock

- unsigned long queue
- pthread_mutex_lock(&ticket->mutex) // for concurrency
- queue = ticket->queue_tail++;
- while queue is not equal to ticket->queue_head
    - pthread_cond_wait(&ticket->cond, &ticket->mutex) // it has to wait
- pthread_mutex_unlock(&ticket->mutex) // now open lock

void ticket_unlock(ticket_lock_t *ticket)

- pthread_mutex_lock(&ticket->mutex)
- ticket->queue_head++;
- pthread_cond_broadcast(&ticket->cond)
- pthread_mutex_unlock(&ticket->mutex)

create a new m as ticket_lock_t to use new locking algorithm

Since my locking algorithm is done, I can use it for my algorithm. In main thread, I initialized two child threads as “childX” and “childO”. I create my matrix in the heap and initialize all elements as ‘ ’. After then I created my threads using “pthread_create” but I put ticket lock before and after both threads. So, they work concurrently, and this provides that “x” will always be the first thread and after “x”, “o” will be the second thread. When I create them, they go to “mythread” function to occupy empty places randomly in the created matrix. In this function there is a for loop to continue to do their jobs with a given condition. When a thread enters the for loop, it should lock the loop’s inside with “ticket_lock” function so that the other thread cannot enter and waits the other to do their jobs one by one. There are some conditions to write ‘x’ or ‘o’ in the matrix so in here the thread inside looks at these conditions and after then it writes. After it writes ‘x’ or ‘o’, it must also check “win conditions” because if it wins, both threads should terminate. For this reason, after “ticket_lock” and “ticket_unclock” conditions, there is a statement which is about stopping threads (if (STOP_THREADS) pthread_exit(NULL);). This statement is important because when a thread terminates after unclocking, normally threads should stop. However, since a thread locks, the other thread waits before lock statement, so that after it unclocks, it can directly go in. That is why, to stop this problem we have to put this statement both after lock and unclock. In main thread, they have to continue while the matrix is not full or none wins. After this, “pthread_join” statement is used both “childX” and “childO” (but first for “childX” because “childO” has to wait to “childX” to finishes.). At the end, matrix is printed and result is given.