CSE 312 / 504 Operating Systems HW 2 Report

Yusuf Fatih Şişman

171044017

Thread Table Structure

I used Thread table as a vector of Thread structure. Thread structure holds ;

id: Id of the thread.

state: It can be READY, RUNNING, BLOCK or TERMINATED.

joinID: Id of another thread which this threads join.

mutexID: Id of mutex that locks this thread.

thread memory datas: PC, registers, stack etc.

Mutexes

Mutexes are hold as a vector of Mutex structure. Mutex structure holds;

id: Id of the mutex.

value: It can be 1 or 0.

lock: It is used for hold how many thread is locked by this mutex.

Interrupt Handling / Changing Thread

When timer interrupt occurs or system calls like join, thread_exit or mutex locks, context switch happens if there are any available thread. Thread choosing making acording to Round-Robin scheduling. Before context switch happens, program saves the datas to the current thread. Then program loads the datas of the choosen thread and continue to running.

System Calls

1-) init: This syscall is used for load assembly files with names and create initial thread. It takes filename as parameter. This syscall is called in GTU_SPIMOS_X.s for load MergeSort.s or ProducerConsumer.s

2-) create_thread: This syscall is used for create new thread. It takes start address as parameter. This syscall return thread id after thread is created.

3-) join_thread: This syscall is used for wait another thread to complete. It takes thread id as parameter. After this syscall is called, state of current thread is changed to BLOCK and context switch occurs. This thread is remain blocked untill the waited thread is finished and notify this thread.

4-) thread_exit: This syscall is used for exit from thread. It sets state of current thread to TERMINATED. All other threads that wait for this thread to finish(join) is notified and context switch occurs.

5-) mutex_init: This syscall is used for create new mutex. It returns id of the created mutex.

6-) mutex_lock: This syscall is used for mutex lock. It takes id of mutex as parameter. If value of mutex is 1, it value is setted to 0. If value of mutex is 0, state of current thread is changed to BLOCK, mutexID value of the current thread is setted to this mutex id, lock value of this mutex is increased by 1 and context switch occurs.

7-) mutex_unlock: This syscall is used for mutex unlock. It takes id of mutex as parameter.

If value of mutex is 1, nothing happens.

If value of mutex is 0 and mutex locked any thread, value of mutex remains 0 and state of locked thread is changed to READY and lock value of this mutex is decreased by 1.

If value of mutex is 0 and mutex did not lock any thread, value of mutex is setted to 1.

In some implementations, a thread can not unlock the mutexes which other threads lock. But for suitability to the producer and consumer problem I

assumed that you dont want us to do this and I did not implement it in this way.

8-) process_exit: This syscall is used for proper exit from program. After this syscall called, state of current thread is changed to BLOCK and context switch happens. And PC is decreased 4. When this process is notified from joined thread, this syscall called again. This syscall called untill all thread is terminated. **Note: This syscall was not implemented like busy waiting. It only called more than one times.**

Merge Sort

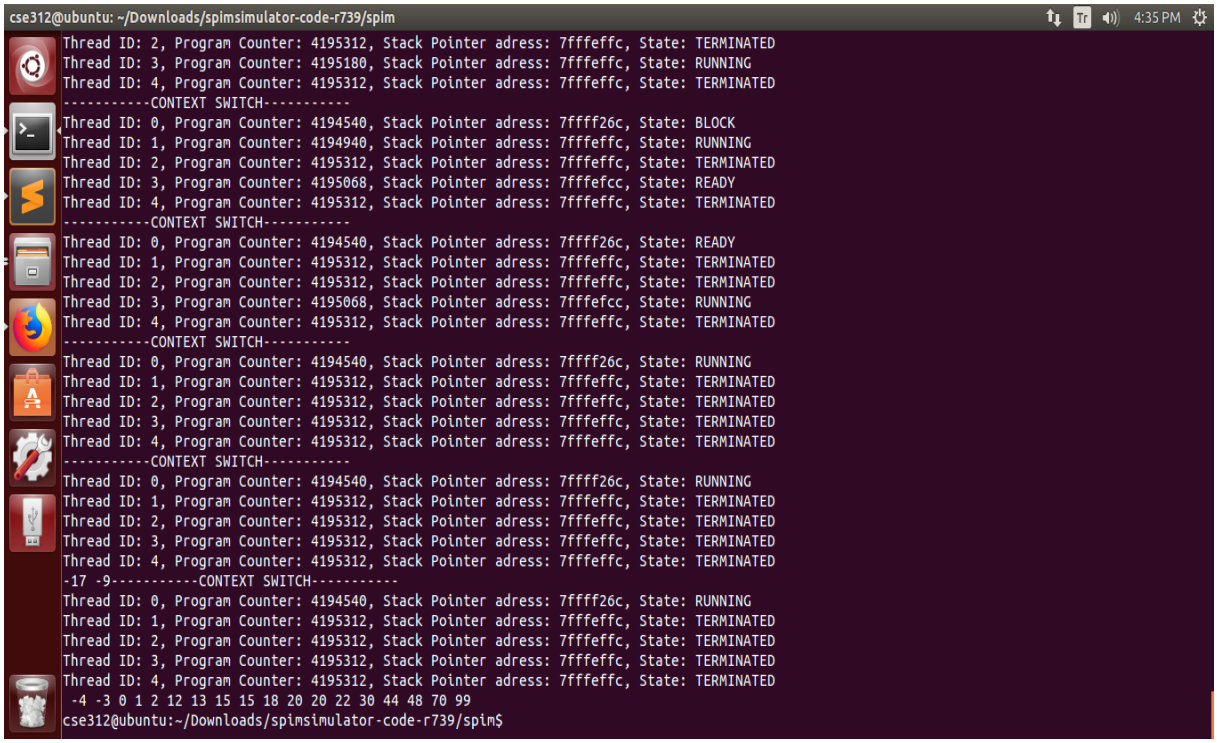
I converted the c code at the geeksforgeeks to the mips assembly.

Producer Consumer

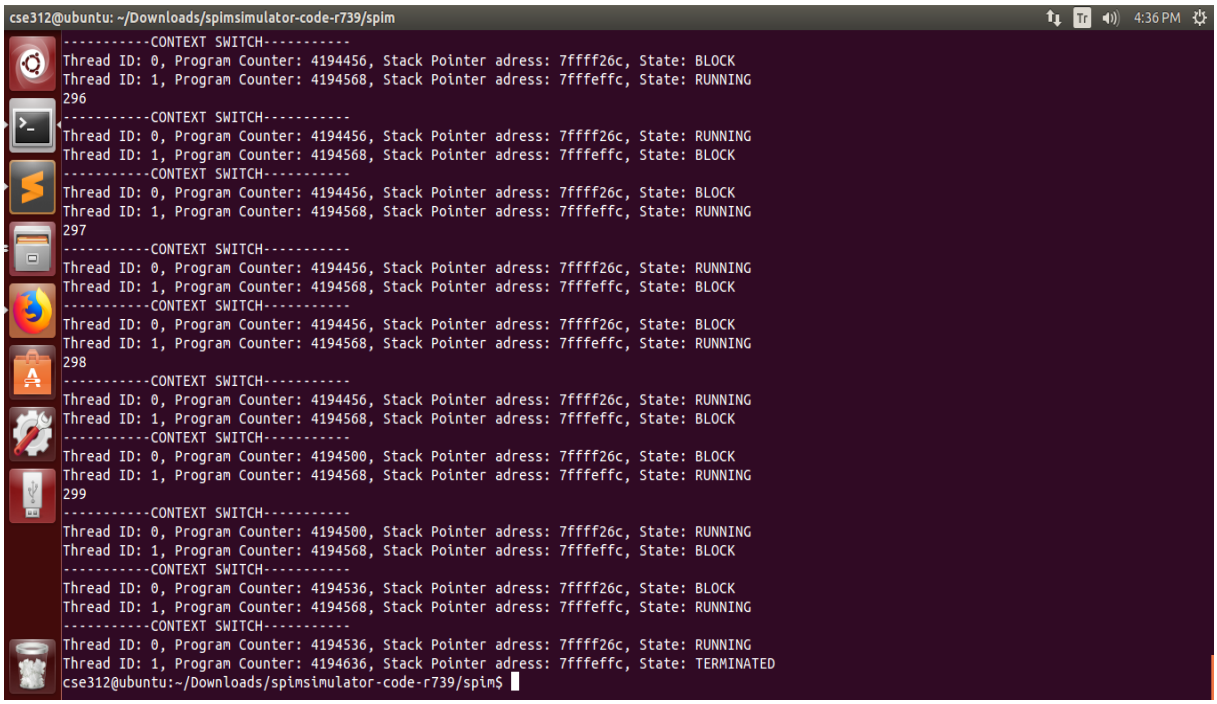
I implemented this problem with 2 mutex. One of them represents empty, the other one represents full. Since the buffer size is 1, there was no need to extra critical section mutex for buffer.

Results

1-) Merge Sort



2-) Producer Consumer With Mutexes



3-) Producer Consumer Without Mutexes