CSE312
HOMEWORK 1
REPORT

EMRE SEZER
1901042640

First, I created Thread and ThreadManager classes. These classes are very similar to the Task and TaskManager classes at the tutorials. In my design, there are Tasks which represents processes. Each Task has Threads and a ThreadManager. Whenever an interrupt occurs, current Thread will switch to another Thread at the threads array. With that approach I provide multithreading.

I created an enum which represents Thread States.
enum ThreadState { READY, RUNNING, BLOCKED };

Thread Class:

There is an int variable typeID for representing if Thread is Producer Thread or Consumer Thread.
If typeID == 0, Producer Thread;
if typeID == 1, Consumer Thread.

There is an threadState variable which is an enum for representing Thread State.

There is a threadID variable which is an int for representing Thread's ID.

createThread function: Makes Thread's threadState READY. Prints threadID and informs that Thread is created.

joinThread function: If Thread's threadState is READY, turns it to RUNNING. Prints threadID and informs that Thread is joined.

yieldThread function: If Thread's threadState is RUNNING, turns it to READY. Prints threadID and informs that Thread is yielded.

apply function: Applies consumer-producer fashion and Peterson's algorithm. If threadID is 0, increases the num by input amount, If threadID is 1 decreases the num by input amount. Also, applies the Peterson's algorithm by using enterCriticalRegion and leaveCriticalRegion functions from the slides of the course. Prints threadID and manipulated value of the input to the screen.

applyX function: Applies consumer-producer fashion.

If threadID is 0, increases the num by input amount, If threadID is 1 decreases the num by input amount. It calls enterCriticalRegion function. But, doesn't call leaveCriticalRegion function. applyX function is for showing that when one of the Threads doesn't leave the critical region, other Thread can't enter the critical region. Expected result is other Thread prints "WAITING" infinite times.

ThreadManager Class:

Each Task has a ThreadManager and each ThreadManager has a Thread array. Keeps currentThread int which represents current Thread index at the threads array, numThreads which represents total number of threads kept in threads array.

Schedule function: Just like TaskManager, ThreadManager has a Schedule function for scheduling Threads. At each interrupt ThreadManager increases currentThread by 1. With that approach, at each interrupt thread switch will happen. Current Thread's yieldThread function is call in the end of this funciton. Also, this function is called whenever TaskManager's Schedule function is called.

AddThread function: Adds input Thread to the threads array.

NOTES:

* Each Task has an uint8_t called counter. This counter is being manipulated by the Threads. Each Thread increases or decreases this variable.

* Program prints infinite many times when counter is manipulated. So, you can comment out the prints inside the apply function and see the print results of the createThread, joinThread, yieldThread.

* You can type "make mykernel.iso" at the command line. Then, use this iso file at the Virtual Machine.

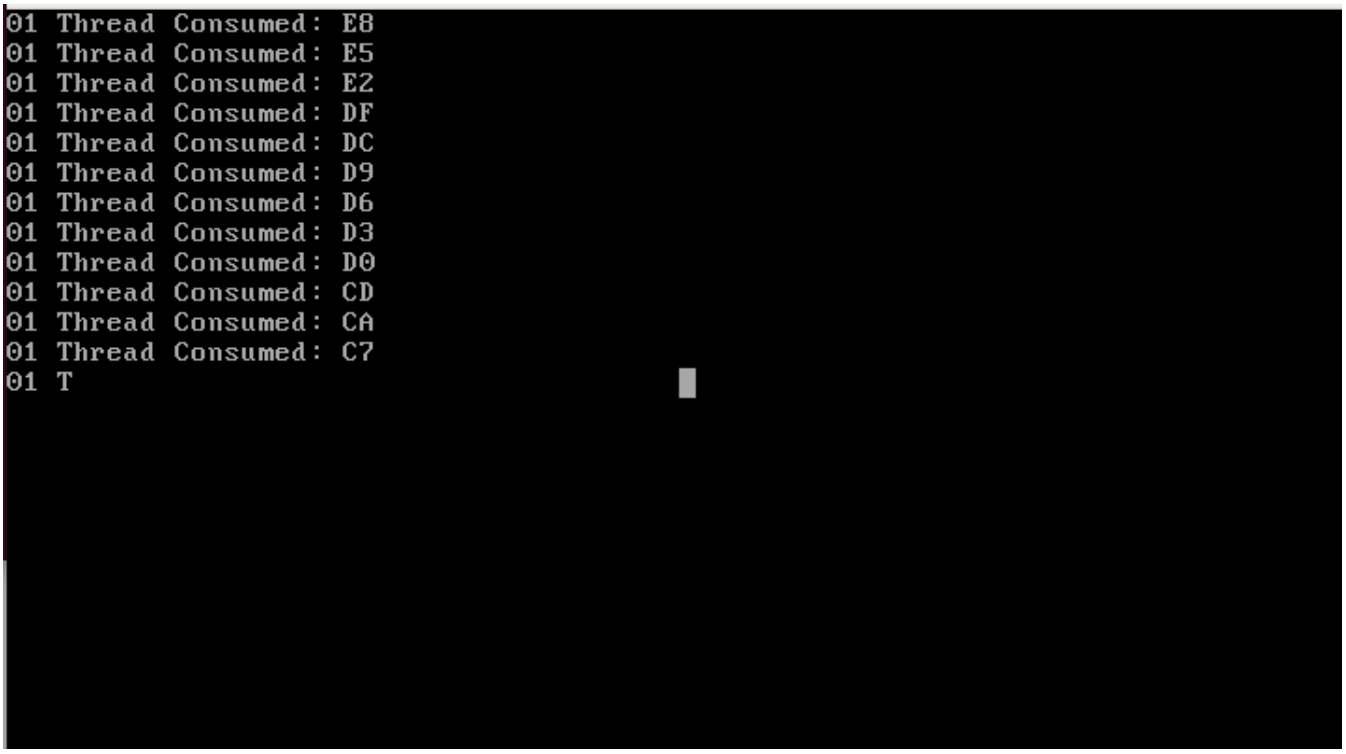
* I thought that I can comment out some of the code and try other functions at the demo and show you some of the different features of the program.

* My program doesn't take any input from the user at the run-time.

* I attached mykernel.iso file at the folder. You can try it directly if you want.

Screenshots:

With apply Function: Means with Peterson's algorithm



With applyX Function: Means one of the threads doesn’t exit critical region and other thread is wating the first thread. You can see at the next screenshot.

You can test my homework and see the same results yourself