Project 1

Draft Due: 6/29/15

CSE 150- Operating Systems

Landon Taylor

Lenny Khoden

Isaac Flores

Andrew Yfantis

1. Implement KThread.join(). A KThread can be used to execute Nachos kernel code. Kthread.join() is a function that checks the current thread to see if it is completed. Kthread must not be the current thread.

It would make sense to use a condition variable to ensure the system keeps track of available resources.

/\* Waits for this thread to finish. If this thread is already finished, return immediately. This method must only be called once; the second call is not guaranteed to return. This thread must not be the current thread. \*/

**public** **void** join() {

if(currentThread is completed)

{

Return;

}

Lib.debug(dbgThread, "Joining to thread: " + toString());

Lib.assertTrue(this != currentThread);

save currentTCB

disable interrupts

put current thread into wait queue

Re-enable interrupts

}

1. Implement condition variables directly, without semaphores. This is to be done in Condition2. This function disables interrupts for synchronization.

Public void sleep()

{

Disable interrupts;

While(current thread exists)

{

Wait to be current thread;

}

//do things until done

//when done go to waiting

Signal ( waiting to be ready. )

Enable interrupts;

}

Public void wake(){

Disable interrupts;

If( something is sleeping)

{

Wake one up; ////put in ready queue=wake

}

Enable interrupts;

}

Public void wake(){

Disable interrupts;

If( anything is sleeping)

{

Wake all up; //put in ready queue=wake

}

Enable interrupts;

}

1. Complete the implementation of the Alarm class by the waitUntil(long x). This method puts the current thread to sleep for x ticks, and wakes it in the timer interrupt handler.

waitUntil( long x)

{

wakeTime = currentTime + x;

while(wakeTime > currentTime)

{

Disable interrupts

Save current TCB

Current thread put into waitQueue

Restore interrupts

}

}

timerInterrupt()

{

kThread.currentThread().yield();

disable interrupts;

//put current into readyQueue

restore interrupts;

}

1. Implement synchronous send and receive of one word messages, using **condition variables**. Implement the Communicator class with operations:void speak (int word) and int listen().

Int reading=0;

Int writing=0;

Public void speak(int word){

Disable interrupts

While(speaking | writing| ){sleep}

Writing++

Enable interrupts

//write stuff

Disable interrupts

writing--

Signal() ; //give up turn

Enable interrupts

}

Public int listen(){

Disable interrupts

While( speaker speaking ){sleep;}

//reading condition variable++

Enable interrupts

//reading pseudocode;

Disable interrupts

Reading condition variable--

Signal() ; //give up turn

Enable interrupts

}

1. Implement priority scheduling in Nachos by completing PriorityScheduler class. It is necessary to change nachos.conf that specifies the scheduler class to use.
2. The boat problem. Many people are stuck on the island of Oahu (at least two children), and need to go to the island Molokai on a single boat. This boat can only hold up to two children, or one adult. Example 3 people waiting:
3. Two children row to Molokai
4. One child rows back to Oahu
5. One adult rows to Molokai
6. The child that was left rows back to Oahu
7. Both children go to Molokai (done)

We can see there needs to be a loop where if an adult crosses, there must be a child waiting at Molokai to bring the boat back, if the adult was not the last person on Oahu.