Operating Systems

CSC468H

Tutorial Week 5

October 9, 2ooo

Outline

Discussion on a concurrency problem

Discussion about newly-arriving threads barging in front of previously waiting threads, all trying to get into a monitor

Small deadlock problem (time-permitting)

Question on Assignment 1

General Information

⇨ Office hours for assignment 1:
Today (Tuesday, October 10th) at 1 p.m. in VC 101 and
at 3 p.m. in SS 2130

⇨ Silent running begins on Tuesday, October 10th at 4p.m.

⇨ Due electronically by noon Thursday, October 12

⇨ Next time: assignment 2 and previous midterms

Concurrency Problem

Consider a system consisting of processes P_1, P_2, …, P_n, each of which has a unique priority number. Write a monitor that allocates three identical line printers to these processes, using the priority numbers for deciding the order of allocation.

Solution:

PrinterMonitor.java

```
/**
* class that implements Printer Monitor with priority access
*/
public class PrinterMonitor
{

private static final int NUM_PRINTERS = 3;

/**
* array of booleans denoting whether a particular printer is in
use
*/
private static boolean[] pLock = new boolean[NUM_PRINTERS];

/**
* current maximum priority among processes waiting to acquire
*/
private static maxPriority = 0;

/**
* current priority sufficient to attempt to acquire resource
*/
private static sufficientPriority = 0;
```

```
/**
* public constructor. Initializes pLock's to false
*/
public PrinterMonitor()
{
for ( int i = 0; i < NUM_PRINTERS; i ++ )
{
pLock[i] = false;
}
}

/**
* acquires the resource
* @param id process priority
* @returns id of the allocated printer
*/
public static int acquire( int id )
{
//
// in this block we wait until we have the highest current
priority
// before we proceed to allocate the resource.
//
while (true)
```

```
if (id > maxPriority )
synchronized (this.class())
{
if (id > sufficientPriority)
//
// we are off out of this block. make sure no one else
// follows us into the next block before we're done.
// set suufficient priority to MAX_INT to enforce that
//
maxPriority = sufficientPriority;
sufficientPriority = MAX_INT;
break;
if (id > maxPriority)
//
// didn't make it to sufficient priority. well, set the
// maxPriority to our id to preserve our rights to come next
//
maxPriority = id;
}
wait();
}

// only one process with current highest priority gets here
// all others are forced to wait in the previous block
int printerId;
```

```
boolean allLocked = true;
while (true)
{
// check if there are available printers
for (int i = 0; i < NUM_PRINTERS; i++ )
{
if (!pLock[i])
{
allLocked = false;
printerId = i;
break;
}
} // for
if (allLocked)
// no available printers yet. wait.
wait();
else
{
// aha! got one. mark it used
pLock[printerId] = true;
// allow next process get out of the previous block
sufficientPriority = maxPriority;
// reset maxPriority, so that awaken processes will
// reconstruct true maxPriority
maxPriority = 0;
```

```
// now it is time to wake up others.
notifyAll();
return printerId;
} // else
} // while
} // acquire()

/**
* release the printer resource
* @param printerId the id of the printer to be released
*/
public static synchronized void release( int printerId )
{
pLock[printerId] = false;
notifyAll();
}
}
```

PrinterMonitor.java

niced version, no comments

```
public class PrinterMonitor
{

    private static final int NUM_PRINTERS = 3;

    private static boolean[] pLock = new boolean[NUM_PRINTERS];

    private static maxPriority = 0;

    private static sufficientPriority = 0;

    public PrinterMonitor()
    {
        for ( int i = 0; i < NUM_PRINTERS; i ++ )
        {
            pLock[i] = false;
        }
    }

    public static int acquire( int id )
    {
        while (true)
        {
```

```
    if (id > maxPriority )

        synchronized (this.class())
        {
            if (id > sufficientPriority)
            {
                maxPriority = sufficientPriority;
                sufficientPriority = MAX_INT;
                break;
            }
        if (id > maxPriority)
        maxPriority = id;
        }
    wait();

}

int printerId;
boolean allLocked = true;
while (true)
{
    for (int i = 0; i < NUM_PRINTERS; i++ )
    {
        if (!pLock[i])
```

```
                allLocked = false;
                printerId = i;
                break;

        }
        if (allLocked)
            wait();
        else
        {
            pLock[printerId] = true;
            sufficientPriority = maxPriority;
            maxPriority = 0;
            notifyAll();
            return printerId;
        }   // else
    }       // while
}           // acquire

public static synchronized void release( int printerId )
{
    pLock[printerId] = false;
    notifyAll();
}
                                }
```

Signal and Continue

The signal-and-continue discipline does not require the signaling thread to leave the monitor after a signal on a condition variable. Nor does the signaled thread have priority to proceed in the monitor over other threads waiting to enter through a service method call.

We cannot guarantee that the condition leading to the signal is still true when the signaled thread reenters the monitor: before leaving the monitor, the signaling thread could change data fields and alter the state of the monitor after generating the signal. Also threads trying to enter the monitor through a service method can "barge" ahead of signaled threads that were waiting on condition variables, resulting in an unbounded waiting time on a condition variable, a form of starvation.

Binary and Counting Semaphore Monitors

Note: JDK doesn't include either a binary or counting semaphore class.
However, they are easily implemented as Java monitors with P and V methods:

Semaphore2.java

```
public abstract class Semaphore {
    protected int value = 0;
    protected Semaphore() {value = 0;}    // constructors
    protected Semaphore(int initial) {value = initial;}
    public synchronized void P() {
        value--;
        if (value < 0)
            try {wait();}
            catch(InterruptedException e) {}
    }
    public synchronized void V() {
        value++;
        if (value <= 0)  notify();
                                }
                                }
```

Assignment 1

1. Question:

I don't understand - one of the paramateres to the program is simulation time. How am I supposed to stop the simulation?

Answer:
The simplest way to do it is to call exit(0); anywhere in your program. Make sure you have printed all the required output before that.

2. Question:

I thought that once water molecule is made, the atoms that made it are removed from the pool (in my case the pool is a Vector instance) and when all H and O combined, the simulation ends. Am I wrong?

Answer:
'Pool of atoms' means 'pool of ready atoms' in the last line of Q1 description. This is suggested by the words: "ready atoms are: " on the same line.
When a water is made, its components (H and O) are not ready anymore, and thus not part of the ready pool.

3. Question:

Could I have an additional thread that would "wake up" (using a condition variable and signal) when the conditions are right for making water? This thread would loop in the makeWater method and wait until it recieves the signal.

Answer:
This is okay, but not required. For example, You can call makeWater from within [HO]Ready methods in the atom thread, which is simpler.

4. Question:
How to interpret the last output string in q2. What does "print out pool of messages" mean? I was thinking that that meant to print out the contents of each consumers message container, to make sure they are in order. Is this correct?

Answer:
It simply means that you have to print out the content of the buffer. Of course you can alternatively print out the messages at each consumer, but I think this will be much more of a hassle to do (i.e. you have to keep track of all processed messages).

5. Question:
When exactly should we consider the start of the simulation. As soon as main has started or after main but before thread creation, or after thread creation etc...

Answer:
It does not matter. However, the start of simulation should be the first timestamp that you get, because all the other timestamps will be relative to it. So i would recommend first option, as soon as main has started.

(see reply to Timestamps post for clarification of timestamp format)

6. Question:
Output format…

Answer:
Print out atoms with their ready timestamps (when they entered pool). No format enforced for this printout. Do what's easier according to your implementation. For example, you could print a line for each atom, or you can just print out timestamps for H and for O. Only print out the ready atoms. Dont print those that have not become ready.

Refer to the following links:

http://www.cs.toronto.edu/~gsg/468

http://www.cs.toronto.edu/~anna/468h/468_t4.ps

and don’t forget the http://java.sun.com!