Andreas Landgrebe

Computer Science 440

1. A clear description of the term semaphore, with comments about its historical origins.

A semaphore is a variable of abstract data type that is used for controlling access. In this laboratory assignment, these semaphores are used to make sure that one thread is being ran at once. If it were ran at the same time, then it would throw an exception. Outside the world of computers, a semaphore is either a visual apparatus with flags, lights or mechanically moving arms, as one used on a railroad, or is is a visual system for sending information by means of two flags that are held one in each hand. are two types of semaphores that exist, counting and binary. Binary is the idea to controlling access only using 0 and 1s. The counting semaphores is the idea to use multiple numbers for a variable or abstract data type.

The idea to first use semaphores was first created by this famous Dutch Computer Scientist named Edward Dijkstra. He had written a paper in 1968, called “THE Multiprogramming System.” This paper address the different layers on a operating system in the context of multiprogramming and address how the semaphores are being used in layer 0 to use a variable to blocked or unblocked threads from running and to check to see if these threads are available or unavailable to run.

The interesting idea of semaphores is being able to use 2 different methods and be able to use these two methods together. The advantage to these two methods is that they are loosely coupled so one does not have a dependency on another to be able to run. In other words, if you make changes to one of the methods, it would not change what is going on in the other method.

It think of this abstracting, if you think of semaphores as train tracks. When it comes train tracks that are going through a road, when a train is about to go through, there is going to be a signal for cars to stop and not pass the road while the train is passing. The idea of this is that the signal acts as a semaphores to signal the cars to go or to halt when the train is in motion. When it comes to threads, semaphores will let one thread go at a time while the rest of the threads are at a halt until the one thread being run is being done or stopped. This idea is known as the atomic action. Once the semaphore operating has been completed, no other thread can be started until the current thread is either finished or stopped.

2. A revised and extended implementation of the producer-consumer model

**From the parameters.java:**

import com.beust.jcommander.Parameter;

public class parameters{

@Parameter (names = {"-totalConsumerNumber","--tcn"}, required = true, description = "total number of consumers")

Integer totalConsumerNumber;

//call the total number of consumers

@Parameter(names = "--debug", description = "Debug mode")

//call whether debug should be turned on or off

boolean debug = false;

}

**From the ProducerConsumer.java:**

import java.util.Random;

import com.beust.jcommander.JCommander;

import java.lang.Thread;

class Producer extends Thread { //extend the thread instead of implementing the runnable to properly run the thread

BoundedBuffer b = null;

Boolean debug;

public Producer(BoundedBuffer initb, Boolean d) {

b = initb;

debug = d; //be able to show debug information about the producer

new Thread(this).start();

}

public void run() {

double item;

Random r = new Random();

while(true){

item = r.nextDouble();

if(debug) System.out.println("produced item " + item + " " + this.getName()); //be able to get the name of the thread

b.deposit(item);

Util.mySleep(200);

}

}

}

class Consumer extends Thread { //extend the thread instead of implementing the runnable to properly run the thread

BoundedBuffer b = null;

Boolean debug;

public Consumer(BoundedBuffer initb, Boolean d) {

b = initb;

debug = d; //be able to show debug information about the consumer

new Thread(this).start();

}

public void run() {

double item;

while (true) {

item = b.fetch();

if(debug) System.out.println("fetched item " + item + " " + this.getName()); //be able to get the name of the thread

Util.mySleep(50);

}

}

}

class ProducerConsumer {

public static void main(String[] args) {

parameters param = new parameters(); //call the object from parameters

new JCommander(param, args); //have JCommnder parse the command line for the parameters and arguments

BoundedBuffer buffer = new BoundedBuffer();

Producer producer = new Producer(buffer, param.debug); //call the debug parameter to show debug information about the producer

for (int i = 0; i<param.totalConsumerNumber; i++ ) { //for loop to go through all of the consumers that have been parse through JCommander

Consumer consumer = new Consumer(buffer, param.debug); //call the debug parameter to show debug information about the consumer

}

}

}

3. A comprehensive analysis of the output of each defective multi-threaded Java program

In the first step, where it is told to comment out the first two calls to P in both the deposit and fetch methods, this caused a defect in this multi-threaded Java program. In this java program, when these four lines of code is commented out, the P will now not check if the the buffer is full and it will also not check to ensure mutual exclusion. Since the P process is not taking effect, then the mutation exclusion and the isFull variable will not be checked and the V method will be incrementing the mutex and the isFull variable by 1 each time so it the producer and consumer will run threads for the remainder of the time until the program is stopped.

In the second step, it is told to comment out the mutex, isFull, and the isEmpty variable. Due to this, the defect will cause the value variable on the counting semaphore to not increase by 1. Due to this, the waiting process will come into effect. Since the size starts at a specific size, as default, it is set to 10. The program will run normally for the first steps but since the value is only decreasing, once it reach zero and the consumer will be waiting for the rest of the time. Due to this, the waiting process will run until the program is halted. This program could just run forever by waiting and nothing would happen since nothing is being added to the variable. Due to this, the consumer will not get a value to be able to run.

4. A reflection of the characteristics of the two different producer-consumer models

There are many different characteristics of the two different producer-consumer models that have been studied. One of the characteristics of the producer-consumer model in Laboratory Assignment 3 was that the program was able to be complete without having the to halt the program when in progress. Due to this, you were also able to compute the amount of time it took for the computer to run a program such as this. Another characteristic of the producer-consumer model in Laboratory Assignment 3 was one was able to specify the total number of data items that was going to be gone through. You were able to specify through the command line how many data items and number of consumers to go through. The more data items that is being called through the command line, the longer it will take to complete this task. Another characteristic one these model is that it is less source code from the model that was just studied for these laboratory assignment. This will give you the idea that it is easier to understand the source code compared to the producer-consumer model that was just studied.

One of the characteristics of this model was from the way the program was set up, the program would run forever unless it was halted. Due to this, you could not see how long it would take to run this program since it would run forever. Another characteristic of this model is the idea to use semaphores. While using semaphores, you are able to use a variable or abstract data type to be able to signal when to use or halt a specific method. In the previous model, this was not the case. Another characteristic of this model is that there is more source code to study, and change compared to the past model which causes this model to be more difficult to understand.

A characteristic that both of these model share is the idea that have two separate methods to be loosely coupled. If you were to make changes to one of the methods, then the other method would not be effected. This is one of the advantages to having this model.

A characteristics that causes this model to be created is a problem for both of these producer consumer model is making sure that the producer does not try to add data into the buffer if it is full and the consumer tries not to remove data is it is empty. If order for this not to happen, the two models that were studied and changed in these 2 laboratory assignment exhibited to idea to use semaphores for the current model and being able to experiment with the synchronization from the last laboratory assignment. The last assignment in laboratory assignment 3 was a stepping stone to understand more complex producer consumer models such as the model that was studied for this laboratory assignment.