EGRE 531 Multicore and Multithread Programming

Laboratory Number 5

Date: 04/13/18

Luis Barquero

PLEDGE: __________Luis Barquero____________________

"On my honor, I have neither given nor received unauthorized aid on this assignment"

VCU | Virginia Commonwealth University

Introduction:

The purpose of this lab is to use threads to simulate two people playing a stone game, where both players between 1 to 4 number of stones from the pile, until there is only one stone left; at that point, whoever picks up the last stone loses.

Lab Content:

To simulate the game, there are two main implementations going on: the first one is both players can pick up 1 to 4 stones and Jerry is always trying to win the game. In order to implement both, Tom's number is assigned a random number (between 1 and 4), while Jerry will use an algorithm to determine how many number of stones to pick in order to leave 1 stone in the pile and force Tom to pick it up. The algorithm used is that Jerry has to leave 1 + 5n number of stones left on the pile, because through this, there will always be 6, 11, 16, 21, …, stones left, so that no matter what Tom picks up, Jerry can leave 1 stone left.

For example, if there are 6 stones left and it's Tom's turn, no matter what he picks, there will be at least 2 stones left in the pile, which means Jerry can force 1 stone left. As an example, if Tom picks 3 stones from the pile, then there will be 6 – 3 = 3 stones left, and Jerry can then pick 2 stones, thus leaving 1 stone left.

To implement this on software, the modulus operation is performed on the current number of stones, and according to the remainder, Jerry will decide how many stones to take, according to the following:

> **Mod 5 = 0:** If the number of stones is perfectly divisible by 5, meaning the remainder is 0, then Jerry will pick 4 stones. The reason for this is if there are 5 stones in the pile, then Jerry should pick 4, so that there is only 1 stone left, forcing Tom to pick it up. If there are 10 stones left, then if Tom picks up 4, that will leave 6 stones left, and no matter what Tom

picks, Jerry force it so there is only 1 stone left, since at the most, Tom can pick 4, which leaves 2 stones, at which point Jerry picks 1, and Tom has to then pick up the last stone.

Mod 5 = 1: If the remainder left after the mod operation is 1, then Jerry will pick 5 – T stones, where T is the number of stones previously picked by Tom. For example, if there are 6 stones left and Tom picked 2, then the remaining number of stones will be 6 – 2 = 4. Jerry will then have pick 5 – 2 = 3 stones, which in return will leave 4 – 3= 1 stone left, thus forcing Tom to win the game. If instead Tom picks 3 stones, then Jerry will pick **5 –** 3 = 2, which will mean the total number of stones is 6 – 3 – 2 = 1, thus forcing Tom to pick the last stone.

Mod 5 = 2: If the remainder of stones left after the mod operation is 2, then Jerry must pick 1 stone if he wants to win game. The reason for this is if there are 7 stones left in the pile, then Jerry picking one stone will leave 7 – 1 = 6 stones left. Tom is then at a loss because no matter what he picks, Jerry can always leave 1 stone left in the pile, thus winning the game. Even if Tom picks the highest number possible, 4, there is still 2 stones left, and Jerry can pick 1 stone, thus leaving 1 stone and winning the game.

Mode 5 = 3: If the remainder of stones left after the mod operation is 3, then Jerry must pick 2 stones in order to win the game. The reason why is because if there are 8 stones left on the pile and Jerry picks 2, then that will leave 8 – 2 = 6 stones left, which means that no matter what Tom picks, Jerry will always leave 1 stone in the pile, thus winning the game.

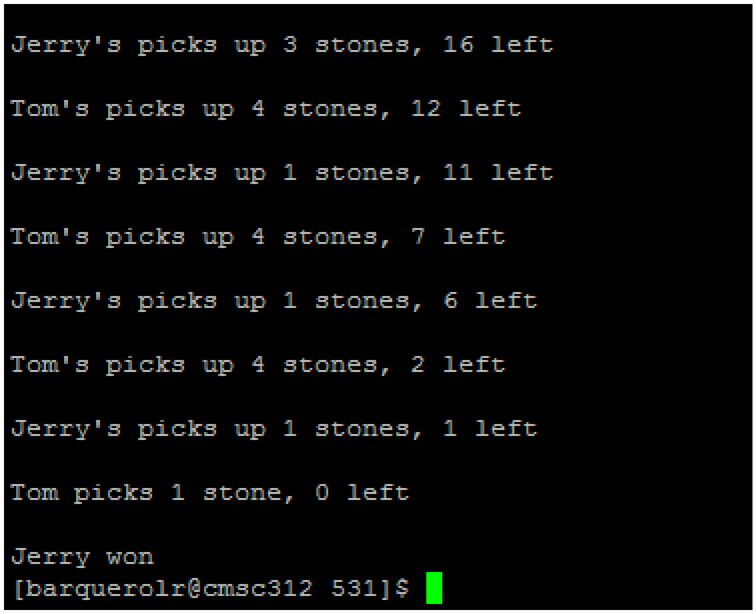
Mod 5 = 4: If the remainder of stones left after the mod operation is 4, then Jerry must pick 3 stones to win the game. The reason why is because if there are 9 stones left and Jerry picks 3 stones, that leaves 9 – 3 = 6. Once again, this means that Tom has lost because no matter what he picks, Jerry will always leave 1 stone left.

Test Results:

To properly test the algorithm, three tests were performed, each with three random initial amounts. All three tests feature Tom picking a random number of stones, between 1 to 4, and Jerry carefully choosing the correct amount of stones, depending the remainder from the modulus 5 operation. For every test, Tom will always go first, and he always waits for the signal from the broadcast to begin his turn. Once his turn ends, it's Jerry's turn, but he can't begin until he receives the signal. Once he receives his signal, he then carefully chooses the amount of stones based off of the mod 5 remainder.

For every test, Jerry will always leave 1 + 5n number of stones left on the pile, forcing Tom to pick the last stone, and thus he wins the game. Figures 1 – 3 display all three tests and their corresponding results.

```
[barquerolr@cmsc312 531]$ ./RR

Tom's picks up 4 stones, 77 left

Jerry's picks up 1 stones, 76 left

Tom's picks up 2 stones, 74 left

Jerry's picks up 3 stones, 71 left

Tom's picks up 3 stones, 68 left

Jerry's picks up 2 stones, 66 left

Tom's picks up 3 stones, 63 left

Jerry's picks up 2 stones, 61 left

Tom's picks up 3 stones, 58 left

Jerry's picks up 2 stones, 56 left

Tom's picks up 1 stones, 55 left

Jerry's picks up 4 stones, 51 left

Tom's picks up 2 stones, 49 left

Jerry's picks up 3 stones, 46 left

Tom's picks up 3 stones, 43 left

Jerry's picks up 2 stones, 41 left

Tom's picks up 4 stones, 37 left

Jerry's picks up 1 stones, 36 left

Tom's picks up 4 stones, 32 left

Jerry's picks up 1 stones, 31 left

Tom's picks up 1 stones, 30 left

Jerry's picks up 4 stones, 26 left

Tom's picks up 1 stones, 25 left

Jerry's picks up 4 stones, 21 left

Tom's picks up 2 stones, 19 left
```



```
Jerry's picks up 3 stones, 16 left

Tom's picks up 4 stones, 12 left

Jerry's picks up 1 stones, 11 left

Tom's picks up 4 stones, 7 left

Jerry's picks up 1 stones, 6 left

Tom's picks up 4 stones, 2 left

Jerry's picks up 1 stones, 1 left

Tom picks 1 stone, 0 left

Jerry won
[barquerolr@cmsc312 531]$
```

Figure 1 – Figure 1 shows the output of the first test performed. As described above, Jerry always leaves 1 + 5n stones left in the pile in order to ensure his victory.

```
[barquerolr@cmsc312 531]$ ./RR

Tom's picks up 3 stones, 50 left

Jerry's picks up 4 stones, 46 left

Tom's picks up 2 stones, 44 left

Jerry's picks up 3 stones, 41 left

Tom's picks up 4 stones, 37 left

Jerry's picks up 1 stones, 36 left

Tom's picks up 4 stones, 32 left

Jerry's picks up 1 stones, 31 left

Tom's picks up 2 stones, 29 left

Jerry's picks up 3 stones, 26 left

Tom's picks up 1 stones, 25 left

Jerry's picks up 4 stones, 21 left

Tom's picks up 1 stones, 20 left

Jerry's picks up 4 stones, 16 left

Tom's picks up 2 stones, 14 left

Jerry's picks up 3 stones, 11 left

Tom's picks up 4 stones, 7 left

Jerry's picks up 1 stones, 6 left

Tom's picks up 2 stones, 4 left

Jerry's picks up 3 stones, 1 left

Tom picks 1 stone, 0 left

Jerry won
[barquerolr@cmsc312 531]$
```

Figure 2 – Figure 2 shows the output of the first test performed. As described above, Jerry always leaves 1 + 5n stones left in the pile in order to ensure his victory.

```
[barquerolr@cmsc312 531]$ ./RR

Tom's picks up 1 stones, 69 left

Jerry's picks up 3 stones, 66 left

Tom's picks up 1 stones, 65 left

Jerry's picks up 4 stones, 61 left

Tom's picks up 2 stones, 59 left

Jerry's picks up 3 stones, 56 left

Tom's picks up 3 stones, 53 left

Jerry's picks up 2 stones, 51 left

Tom's picks up 3 stones, 48 left

Jerry's picks up 2 stones, 46 left

Tom's picks up 2 stones, 44 left

Jerry's picks up 3 stones, 41 left

Tom's picks up 3 stones, 38 left

Jerry's picks up 2 stones, 36 left

Tom's picks up 4 stones, 32 left

Jerry's picks up 1 stones, 31 left

Tom's picks up 3 stones, 28 left

Jerry's picks up 2 stones, 26 left

Tom's picks up 4 stones, 22 left

Jerry's picks up 1 stones, 21 left

Tom's picks up 3 stones, 18 left

Jerry's picks up 2 stones, 16 left

Tom's picks up 1 stones, 15 left

Jerry's picks up 4 stones, 11 left

Tom's picks up 3 stones, 8 left
```

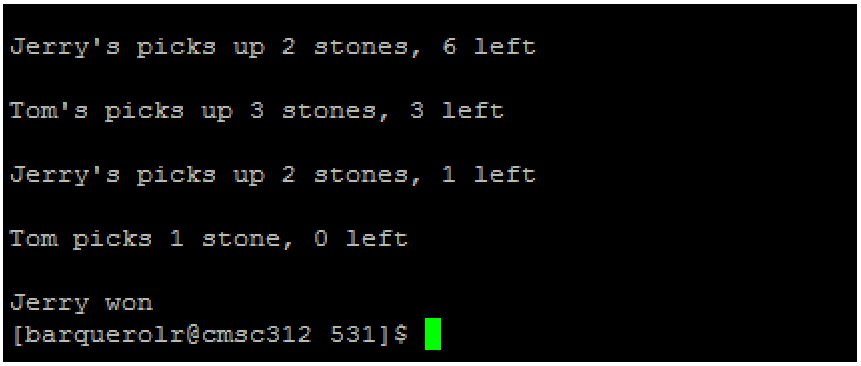


Figure 3 – Figure 3 shows the output of the first test performed. As described above, Jerry always leaves 1 + 5n stones left in the pile in order to ensure his victory.

<u>Problems Encountered:</u>

The main problems was determining the algorithm to make Jerry win every game. Once that was determined, the next problem was implementing it with threads. This caused some issues, since I realized I needed the variables for the number of stones in the pile, the number of stones Tom and Jerry can pick, and the remainder after performing the mod 5 operation and the thread functions only take pointers. To solve this issue, those variables were turned into static global variables, and from there, the thread implementation was complete.

Appendix A

Lab_5.cpp Source Code

```
// Lab5.cpp : Defines the entry point for the console application.
/**********************************************************************''
EGRE 531 Lab 5
Programmed by: Luis Barquero
Purpose: Program will use threads to simulate the stone game, which two people
         Tom and Jerry will be picking stones from a basket. The person to pick
         the last stone loses.
***********************************************************************/

#include<iostream>
#include<stdio.h>
#include<time.h>
#include<cstdlib>
#include<ctime>
#include<pthread.h>

using namespace std;

static int num;                     //number of stones
static int mod;                     //number mod 5. Used for determining Jerry's next move.
static int tom;                     //Number of stones Tom will pick
static int jerry;                   //Number of stones Jerry will pick

void *tomStones(void*);              //Void function for creating thread for Tom
void *jerryStones(void*);            //Void function for creating thread for Jerry

pthread_mutex_t mutextom;            //Mutex for Tom
pthread_mutex_t mutexjerry;          //Mutex for Jerry
pthread_cond_t count_threshold_cv;   //Signal

void *tomStones(void *toms)
{
    pthread_t jerrys;               //Instantiates Jerry's thread

    wait(signal);                   //Waits for the signal to begin its turn

    srand(time(NULL));              //Random function
    num = 21 + rand() % 80;         //Num = number of stones, and this will determine the number of starting stones

    for(int i = 0; i < num; i++)    //Loop will acquire a random number of stones that Tom will pick, and will call Jerry's thread
    {
        pthread_mutex_lock(&mutextom);  //Locks the thread to ensure Tom gets the correct number of stones, and there is no interference
        tom = 1 + rand() % 4;           //Calculates random number of stones for Tom to pick
        num = num - tom;                //This will subtract the number of stones Tom picked from the overall number of stones
        cout << "\nTom's picks up " << tom << " stones, " << num << " left" << endl;
        mod = num % 5;                      //This is used to determine Jerry's course of action when picking the stones
        pthread_mutex_unlock(&mutextom);    //Once the calculations are complete, the mutex is unlocked, and the thread is free to go

        pthread_create(&jerrys, NULL, jerryStones, NULL);       //Creation of Jerry's thread
        pthread_join(jerrys, NULL);                             //Jerry's thread joins
        pthread_exit(NULL);                                     //Tom's thread exits
    }

   pthread_exit(NULL);
}

/*
In order for Jerry to win, the following algorithm must be implemented:
    1) If the number of stones mod 5 - num % 5 - == 0, this means the Jerry must pick 4 stones.
```

```
        For example, if there are 5 stones left, Jerry has to pick 4 because this leaves one stone left, thus forcing Tom to pick the last one.

    2) If the number of stones mod 5 == 1, Jerry must pick (5 - T) stones, T being Tom's last number of stones picked.
        For example, if there are 6 stones left and Tom picks 3 stones, Jerry will pick (5 - 3 = 2) stones, leaving only 1 stone left, forcing Tom to pick the last one

    3) If the number of stones mod 5 == 2, Jerry must pick 1 stone. For example, if there are 7 stones left, Jerry must pick 1 stone, leaving 6 in total. This allows Jerry
        to win, because no matter what Tom picks, Jerry will be able to leave 1 stone left in the basket, forcing Tom to pick it.

    4) If the number of stones mode 5 == 3, Jerry must pick 2 stones. For example, if there are 8 stones, Jerry must pick 2 stones, leaving 6 in total. Once again, this allows
        Jerry to win, because no matter what Tom picks, Jerry will always leave 1 stone left in the basket, thus forcing Tom to pick the last stone.

    5) If the number of stones mod 5 == 4, Jerry must pick 3 stones. For example, if there are 9 stones, Jerry must pick 3 stones, leaving 6 in total. This, once again,
        ensures Jerry's victory because no matter what Tom picks, Jerry will always leave 1 stone left, forcing Tom to pick it.

        The idea is to have Jerry pick an amount that will leave 1 + 5n stones left, so that Jerry can leave 1 stone in the end, thus winning the game.

*/

void *jerryStones(void *jerrys)
{
    wait(signal);                                       //Waits for the signal to begin its turn

    if(mod == 0)                                        //If num % 5 = 0, Jerry must pick 4, so that there is only one stone left, forcing Tom to pick it
        {
            pthread_mutex_lock(&mutexjerry);            //Locks the mutex
            jerry = 4;                                  //Sets the number Tom can pick to 4
            num = num - jerry;                          //Subtracts the number picked by Jerry from the number of stones.
            cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
            pthread_mutex_unlock(&mutexjerry);          //Unlocks the mutex
        }

        if(mod == 1)                                    //If num % 5 = 1, Jerry must pick ( 5 - T) where T is the number of stones Tom previously picked,
                                                        //so that there is only one stone left, thus forcing Tom to pick the last stone.
        {
            pthread_mutex_lock(&mutexjerry);            //Locks the mutex
            jerry = 5 - tom;                            //Sets the number of stones Jerry can pick ( 5 - T), where T is the number of stones Tom previously picked
            num = num - jerry;                          //Subtracts the number of stones Jerry picked from the overall total
            cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
            pthread_mutex_unlock(&mutexjerry);          //Unlocks the mutex
        }

        if(mod == 2)                                    //If num % 5 = 2, Jerry must pick 1, so the there is only one stone left, forcing Tom to pick it
        {
```

```
            pthread_mutex_lock(&mutexjerry);                          //Locks the mutex
            jerry = 1;                                                //Sets the number of stones Jerry can pick to 1
            num = num - jerry;                                        //Subtracts the number of stones picked by Jerry from the overall total
            cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
            pthread_mutex_unlock(&mutexjerry);                        //Unlocks the mutex
        }

        if(mod == 3)                                                  //If num % 5 = 3, Jerry must pick 2, so that there is only 1 stone left, forcing Tom to pick it
        {
            pthread_mutex_lock(&mutexjerry);                          //Locks the mutex
            jerry = 2;                                                //Sets the number of stones Jerry picks to 2
            num = num - jerry;                                        //Subtracts the number of stones Jerry picked from the overall total
            cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
            pthread_mutex_unlock(&mutexjerry);                        //Unlocks the mutex
        }

        if(mod == 4)                                                  //If num % 5 == 4. Jerry must pick 3, so that there is only 1 stone left, forcing Tom to pick it
        {
            pthread_mutex_lock(&mutexjerry);                          //Locks the mutex
            jerry = 3;                                                //Sets the number of stones Jerry must pick to 3
            num = num - jerry;                                        //Subtracts the number of stones picked by Jerry from the overall total
            cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
            pthread_mutex_unlock(&mutexjerry);                        // Unlocks the mutex
        }

    while(num >= 1)                                                   //This while loop is used because the loop above only iterates to number 11 or 16
                                                                      //Therefore, this while loop will pick up and finish the game properly
    {

        if(num == 1)                                                  //If the number of stones == 1, Tom picks the last stone and Jerry wins, thus ending the game
        {
            pthread_mutex_lock(&mutextom);                            //Locks the mutex
            cout << "\nTom picks 1 stone, 0 left" << endl;
            cout << "\nJerry won" << endl;
            pthread_mutex_unlock(&mutextom);                          //Unlocks the mutex
            exit(1);                                                  //The program completes
        }

        else                                                          //If the number of stones is not 0, the program will keep on subtracting stones until there is only 1 left
        {
            if(num % 5 == 1)                                          //If num % 5 == 1, Tom will go next, and Jerry will pick (5 - T) stones, where T is the number of stones
                                                                      //Tom previously picked, so that there is only 1 stone left, forcing Tom to pick it, making Jerry win
            {
                pthread_mutex_lock(&mutextom);                        //Lock the mutex
                tom = 1 + rand() % 4;                                 //Tom picks a random amount of stones
```

```
                num = num - tom;                            //Subtracts the number of stones Tom picked from the overall amount
                cout << "\nTom's picks up " << tom << " stones, " << num << " left" << endl;
                pthread_mutex_unlock(&mutextom);            //Unlocks the mutex

                pthread_mutex_lock(&mutexjerry);            //Locks the mutex
                jerry = 5 - tom;                            //Sets the number of stones Jerry can pick to (5 - T)
                num = num - jerry;                          //Subtracts the number of stones Jerry picked from the overall total
                cout << "\nJerry's picks up " << jerry << " stones, " << num << " left" << endl;
                pthread_mutex_unlock(&mutexjerry);          //Unlocks the mutex
            }
        }
    }
}

int main()
{
    pthread_t toms;                                                     //Instantiating thread clause for Tom
    pthread_create(&toms, NULL, tomStones, NULL);                       //Creates the thread for Tom
    pthread_join(toms, NULL);                                           //Tom's thread joins

    pthread_attr_t attr;                                                //Initializes thread attributes, in this case signal.
    pthread_cond_init (&count_threshold_cv, NULL);                      //Initializes thread condition, in this case for the signal to be broadcasted
    pthread_attr_init(&attr);                                           //Initializes thread attribute
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);        //Sets the detach state attribute of the thread attributes object referred to by attr to the value specified in detachstate.

    pthread_attr_destroy(&attr);                                        //Destroys attribute
    pthread_cond_destroy(&count_threshold_cv);                          //Destroys condition variable
    pthread_mutex_destroy(&mutextom);                                   //Destroys Tom's Mutex
    pthread_mutex_destroy(&mutexjerry);                                 //Destroys Jerry's Mutex

    pthread_exit(NULL);                                                 //Tom's thread exits
    return 0;
}
```