OPERATING SYSTEMS LAB

LAB MANUAL # 14

Instructor: Muhammad Abdullah Orakzai

Semester Fall 2021

Course Code: CL2006

Fast National University of Computer and Emerging Sciences, Peshawar

Department of Computer Science

Contents

Synchronization & Deadlocks

Busy Waiting

The following sections will help you explain the difference between normal waiting (using sleep()) and busy waiting. Run both the codes. Both codes are ALMOST the same. The only difference is the contents of the sleepy() function. Make sure you are not doing ANYTHING ELSE besides staring at your screen while executing these programs.

Normal Sleep

```
#include <do yourself>
int counter = 60;
void *sleepy()
{
sleep(60);
printf("Hello");
}
int main()
{
pthread_t tid;
pthread_create(&tid, NULL, sleepy, NULL);
while (counter > 0)
{
counter--;
sleep(1);
}
pthread_join(tid, NULL);
}
```

Compile the above code, and for running, enter the following command:

```
# uptime && ./programName && uptime
```

The above command bundle is going to run the uptime tool, followed by your compiled program, and followed by the uptime tool again in quick succession. Note down the load average for the first 1 minute.

Busy Wait

```
#include <do yourself>
int counter = 60;
void *sleepy()
{
while(counter > 0)
{
}
printf("Hello");
}
int main()
{
pthread_t tid;
pthread_create(&tid, NULL, sleepy, NULL);
while (counter > 0)
{
counter--;
sleep(1);
}
pthread_join(tid, NULL);
}
```

Compile the above code, and for running, enter the following command:

```
# uptime && ./programName && uptime
```

The above command bundle is going to run the uptime tool, followed by your compiled program, and followed by the uptime tool again in quick succession. Note down the load average for the first 1 minute.

What is the difference?

Banking Example

Since it will be too much work (for me) to have a separate problem set for each synchronization algorithm, I will put forward a simple case of an account debit and account credit in a banking software.

Consider a software for a bank which has a shared global variable called `balance`. At the moment, this bank only has support for handling one customer. Whenever the customer visits the bank, the bank manager will note down his time of arrival.

If the customer visit is in regard of adding money to his account, the bank manage will note down the amount deposited. If the customer wants to withdraw money from the bank, the bank manager will note down the amount deducted.

Later on, before closing hours, the bank manager will take his daily log book and start making entries for the number of times the customer visited the bank and what kind of transaction was the customer interested in.

If at any time the transaction was only credit, the bank manager will select option 1. If the transaction was only debit, the bank manager will select option 2. If the transaction was both debit and credit, then the manager will choose option 3.

A sample code (not run, not tested, may have errors) for this bank software is given below:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
double balance = 0;
double temp1 = 0;
double temp2 = 0;
int times = 0;
void *credit(void * arg)
{ int a = *(int *) arg;
balance = balance + a;
}
void *debit(void * arg)
{ int a = *(int *) arg;
```

```
balance = balance - a;
}
int main()
{ int choice;
pthread_t credit_thread, debit_thread;
while(1)
{
   system("clear");
   printf("Name: Alif Noon\nAcc No.: 420\n");
   printf("Available Balance: Rs. %f/-\n", balance);
   printf("\nChoose Transaction Type:\n");
   printf("------------------------\n");
   printf("1. Account Credit\n2. Account Debit\n
             3. One Credit + One Debit\n
             4. Multiple Credits\n

             5. Multiple Debits\n");

  printf("6. Multiple Credit and Debit\n

             7. Exit\n");
  scanf("%d", &choice);
 if (choice == 1)
{
    printf("Enter amount to credit\n");
   scanf("%d", &temp1);
   printf("main. balance = %d\n", balance);
   pthread_create(&credit_thread, NULL,
   credit, &temp1);
}
else if (choice == 2)
{
   printf("Enter amount to debit\n");
   scanf("%d", &temp2);
   pthread_create(&debit_thread, NULL,
   debit, &temp2);
```

```
}
else if (choice == 3)
{
   printf("Enter amount to credit\n");
   scanf("%d", &temp1);
   printf("Enter amount to Debit\n");
   scanf("%d", &temp2);
   pthread_create(&credit_thread, NULL,
   credit, &temp1);
   pthread_create(&debit_thread, NULL,
   debit, &temp2);
}
else if (choice == 4)
{
   printf("\nHow many times to make
   credit transaction?");
   scanf("%d", &times);
   printf("Enter amount to credit\n");
   scanf("%d", &temp1);
   int loop;
   for (loop = 0; loop < times; loop++)
   {
      pthread_create(&credit_thread, NULL, credit, &temp1);
   }
}
else if (choice == 5)
{
   printf("\nHow many times to make
   debit transaction?");
   scanf("%d", &times);
   printf("Enter amount to debit\n");
   scanf("%d", &temp2);
  int loop;
  for (loop = 0; loop < times; loop++)
```

```
 {
    pthread_create(&debit_thread, NULL, debit, &temp2);
  }
}
else if (choice == 6)
{
   printf("Enter amount to credit\n");
   scanf("%d", &temp1);
   printf("Enter amount to Debit\n");
   scanf("%d", &temp2);
   printf("Enter number of times to Credit and Debit\n");
  scanf("%d", &times);
  pthread_create(&credit_thread, NULL,credit, &temp1);
  int loop;
  for (loop = 0; loop < times; loop++)
 {
  pt hread_create(&credit_thread, NULL, credit, &temp1);
 }
for (loop = 0; loop < times; loop++)
{
pthread_create(&debit_thread, NULL, debit, &temp2);
 }
}
else if (choice == 7)
{
break;
}
else
{
   printf("Wrong Choice");
   continue;
}
}
```

```
   pthread_join(credit_thread, NULL);
   pthread_join(debit_thread, NULL);
   printf("Account Summary:\n----------------\n");
   printf("Available Balance: Rs. %f/-\n", balance);
   exit(0);
}
```

The code is also given on http://lectures for you to copy paste. Run and compile the code and play around with it. Most of the code is self explanatory and you have done it already in previous labs.

Check the program to see if it is giving any improper behaviour and note it down.

In the following block:

```
void *credit(void * arg)
{ int a = *(int *) arg;
balance = balance + a;
}

void *debit(void * arg)
{ int a = *(int *) arg;
balance = balance - a;
}
```

Add a sleep(1); at the end of each function and then run the program. Why is it that still we are given the same kind of behavior. Whereas in previous lab you will recall that adding sleep(1) actually gave you different behavior.

Modify the code such as follows:

- Instead of balance = balance + a;

Use

```
i n t l o c a l = balanc e ;
l o c a l = l o c a l + a ;
balanc e = l o c a l ;
```

- Instead of balance = balance - a;

Use

```
i n t l o c a l = balanc e ;
l o c a l = l o c a l ⍰ a ;
balanc e = l o c a l ;
```

Both the above modifications are doing the same thing. There is nothing new being implemented. The only difference is that instead of a short atomic instruction, you have divided the task over a series of 3 instructions.

Now run your code. What is the behavior you are getting now?

Try adding a sleep(1) at the end of the function and study the behavior now.

Now move the sleep(1) call from the end of the function to the middle of the function

(in between the instructions). Check the behaviour

now.

Presto!!! The software has gone nuts. Welcome race condition.

The difference we have seen is that initially we had multiple threads but they were executing sequentially. When we place sleep(1) in the middle, we are imposing some kind of rudimentary synchronization so that the threads execute parallely rather than sequentially. Parallel execution of threads mean that multiple threads are trying to enter to access/modify the shared resource (balance variable) at the same time. It is this simultaneous usage that leads to race conditions.

Exercise: Introducing Synchronization

Try introducing synchronization in the banking software according to the 3 methods that we studied during lectures. The techniques are:

1. Algorithm 1 (Usage of turn variable)
2. Algorithm 2 (Usage of boolean array of size 2)
3. Algorithm 3 (Peterson's Algorithm: Usage of both Algorithm 1 + Algorithm 2)

You may use your lecture notes as a reference.