Computer Science and Engineering & Information Technology (2^{nd} Year B.Tech.) IIIT Kalyani, West Bengal

Operating System Lab (CS 411): (Spring: 2019-2020)

Assignment - 8 *Marks: 10*

Assignment Out: *13th March, 2020*

In this experiment we shall see how a data structure gets corrupted due to race condition and how to solve it. Consider the following definition of a stack that stores data of type `int`. It is implemented on a singly-linked list.

```
// stack.h
#ifndef _STACK_H_
#define _STACK_H_
#include <cstdio>

#define ERROR 1
#define OK 0
#define DELAY 1000000

typedef struct node {
        int data;
        struct node *next;
} node;

class stack {
      node *sp;
public:
      stack();
      int push(int);
      int pop(int &); // pop() is pop()+top()
      bool isEmpty();
};
#endif

// stack.c++
// c++ -Wall -c stack.c++
#include "stack.h"

stack::stack(){ sp=NULL; }

bool stack::isEmpty(){ return sp==NULL; }

int stack::push(int n){
    node *tp;

    tp = new node;
    tp->data = n;
    tp->next = sp;
    for(int jpush=1; jpush<=DELAY; ++jpush); // delay loop
    sp = tp;
    return OK;
}

int stack::pop(int &n){
    node *tp;
    if(isEmpty()) return ERROR;
    n = sp->data;
    tp = sp;
    for(int jpop=1; jpop<=DELAY; ++jpop); // delay loop
    sp = sp->next;
    delete(tp);
    return OK;
}
```

1. Write a C++ program to do the following:

   (a) It reads a positive integer n.

   (b) Then it creates two pthreads *producer* and *consumer*.

   (c) The *producer* thread pushes integer $1 \cdots n$ in a stack (declared globally).

   (d) The *consumer* thread is in a non terminating loop, pops data from the stack when it is not empty and prints it.

   (e) The output (popped data) shows the race condition and the corrupted data structure. Note that the program does not terminate normally as one thread has non terminating loop.

   (f) Compilation:

   ```
   $ c++ -Wall -c stack.c++
   $ c++ -Wall -c prodConStack.c++
   $ c++ stack.o prodConStack.o -lpthread
   ```

   You may use a Makefile.

   ```
   a.out: prodConStack.o stack.o
   c++ stack.o prodConStack.o -lpthread

   prodConStack.o: prodConStack.c++
   c++ -Wall -c prodConStack.c++

   stack.o: stack.c++ stack.h
   c++ -Wall -c stack.c++

   clean:
   rm a.out *.o
   ```

   Input/Output:

   ```
   $ ./a.out
   Enter a positive integer: 2
   val: 1
   val: 2
   val: 0

   $ ./a.out
   Enter a positive integer: 10
   val: 1
   val: 2
   val: 0
   val: 3
   val: -201323712
   val: 4
   val: -201323680
   val: 5
   val: -201323648
   val: 6
   val: 7
   double free or corruption (fasttop)
   Aborted (core dumped)
   ```

2. Identify the critical sections of code within push() and pop() functions of `stack.c++` (do not remove the delay loop).

3. Declare a global variable `int lockMe`. Use `x86-64` machine instructions `lock` and `btsl` and the global variable `lockMe` to lock the critical section. You may prepare `void myLock(int *);` and `void myUnlock(int &);` if you wish.

4. Show that there is no race condition.

Input/Output:

```
$ ./a.out
Enter a positive integer: 2
val: 1
val: 2

$ ./a.out
Enter a positive integer: 10
val: 1
val: 2
val: 3
val: 4
val: 5
val: 6
val: 7
val: 8
val: 9
val: 10
```