2021-NTU-SCSE-CZ2005

Operating Systems

Laboratory Implementation **3** Guide

Prepared by: Nailah Gucon

Demo of Code

To demonstrate output of ***TestValueOne()***, for example, comment out other test functions in ***ThreadTest()***:

e.g. To test TestValueOne(), comment out TestValueMinusOne() & TestConsistency()

```
//select the function that you want to test.
void
ThreadTest()
{
        int loopTimes=0;
    DEBUG('t', "Entering SimpleTest");
    //for exercise 1.
    TestValueOne();
    TestValueMinusOne();
    //for exercise 2.
    TestConsistency();
}
```

Code to obtain output:

```
>> make or make clean

>> ./nachos
```

Exercise 1

Read the Nachos thread test program ***threadtest.cc*** carefully. There is a shared variable named **value** (initially zero). There are two functions, namely ***void Inc(_int which)*** and void ***Dec(_int which)***, which increases and decreases value by one, respectively.

In this exercise, you need to consider different interleaving executions of Inc and Dec so that the shared variable value is equal to a predefined value after the threads complete.

You need to implement the following functions:

```
void Inc_v1(_int which)

void Dec_v1(_int which)

void TestValueOne()

void Inc_v2(_int which)

void Dec_v2(_int which)

void TestValueMinusOne()
```

When all the threads (two *Inc_v1* threads and two *Dec_v1* threads) complete in TestValueOne(), **value=1**.

When all the threads (two *Inc_v2* threads and two *Dec_v2* threads) complete in TestValueMinusOne(), **value=-1**.

In *Inc_v1* and *Dec_v1*, you need to use Yield primitive in Nachos to induce context switch. Inc_v1 and Dec_v1 should have the same logic as Inc and Dec, respectively. You are only allowed to add Yield into those two functions. You need to implement ***ThreadValueOne()*** by creating two threads with *Inc_v1* and two threads with *Dec_v1*. The current thread should wait for all those threads to complete. At the end of ***TestValueOne()***, a checking is performed on whether the value is 1.

If the checking passes, you should get the message "congratulations! passed.". Otherwise, an error message is printed.

Exercise 1.1:

```
//exercise 1: two Inc threads and two Dec threads, and implement the interleaving
//so that value=targetV when all the four threads ends.


//targetV=1;
//After executing TestValueOne(), the value should be one.
//1. implement the new version of Inc: Inc_v1
void Inc_v1(_int which)
{
        //fill your code
        int a=value;
        a++;
        currentThread->Yield();
        value=a;
        printf("**** Inc thread %d new value %d\n", (int) which, value);
}

//2. implement the new version of Dec: Dec_v1
void Dec_v1(_int which)
{
        //fill your code
        int a=value;
        a--;
        value=a;
        printf("**** Dec thread %d new value %d\n", (int) which, value);
}
```

```
//3. implement TestValueOne by create two threads with Inc_v1 and two threads with Dec_v1
// you should pass the checking at the end, printing "congratulations! passed."
void TestValueOne()
{
        value=0;
        printf("enter TestValueOne, value=%d...\n", value);
        //1. fill your code here
        Thread *it1 = new Thread("Inc_v1_1");
        Thread *it2 = new Thread("Inc_v1_2");
        Thread *dt1 = new Thread("Dec_v1_1");
        Thread *dt2 = new Thread("Dec_v1_2");

        it1->Fork(Inc_v1, 1, 0);
        it2->Fork(Inc_v1, 2, 0);
        dt1->Fork(Dec_v1, 3, 0);
        dt2->Fork(Dec_v1, 4, 1);

        currentThread->Join(dt2);


        //2. checking the value. you should not modify the code or add any code lines behind
        //this section.
        if(value==1)
            printf("congratulations! passed.\n");
        else
                printf("value=%d, failed.\n", value);
}
```

Output:

```
enter TestValueOne, value=0...
**** Dec thread 3 new value -1
**** Dec thread 4 new value -2
**** Inc thread 1 new value 1
**** Inc thread 2 new value 1
congratulations! passed.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 150, idle 0, system 150, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0, outs 0, tlb miss: 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Code to obtain output:

```
>> make or make clean

>> ./nachos
```

Exercise 1.2:

```
//targetV=-1;
//After executing TestValueMinusOne(), the value should be -1.
//1. implement the new version of Inc: Inc_v2
void Inc_v2(_int which)
{
        //fill your code
        int a=value;
        a++;
        currentThread->Yield();
        value=a;
        printf("**** Inc thread %d new value %d\n", (int) which, value);
}

//2. implement the new version of Dec: Dec_v2
void Dec_v2(_int which)
{
        //fill your code
        int a=value;
        a--;
        currentThread->Yield();
        value=a;
        printf("**** Dec thread %d new value %d\n", (int) which, value);
}

//3. implement TestValueMinusOne by create two threads with Inc_v2 and two threads with
Dec_v2
// you should pass the checking at the end, printing "congratulations! passed."
void TestValueMinusOne()
{
        value=0;
        printf("enter TestValueMinusOne, value=%d...\n", value);

        //fill your code
        Thread *it1 = new Thread("Inc_v2_1");
        Thread *it2 = new Thread("Inc_v2_2");
        Thread *dt1 = new Thread("Dec_v2_1");
        Thread *dt2 = new Thread("Dec_v2_2");

        it1->Fork(Inc_v2, 1, 0);
        it2->Fork(Inc_v2, 2, 0);
        dt1->Fork(Dec_v2, 3, 0);
        dt2->Fork(Dec_v2, 4, 1);

        currentThread->Join(dt2);

        //2. checking the value. you should not modify the code or add any code lines behind
        //this section.
        if(value==-1)
                printf("congratulations! passed.\n");
        else
                printf("value=%d, failed.\n", value);
}
```

Output:

```
enter TestValueMinusOne, value=0...
**** Inc thread 1 new value 1
**** Inc thread 2 new value 1
**** Dec thread 3 new value -1
**** Dec thread 4 new value -1
congratulations! passed.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 170, idle 0, system 170, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0, outs 0, tlb miss: 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Code to obtain output:

```
>> make or make clean

>> ./nachos
```

Exercise 2

You need to implement the following three functions. When all the four threads (two *Inc_Consistent* threads and two *Dec_Consistent* threads) complete in ***TestConsistency()***, value=0. You need to achieve consistent results (value=0), regardless of different interleaving execution orders in *Inc_Consistent* and *Dec_Consistent* as well as different thread fork orders in ***TestConsistency()***.

```
void Inc_Consistent (_int which)

void Dec_Consistent (_int which)

void TestConsistency ()
```

In *Inc_Consistent* and *Dec_Consistent*, you use Yield interface in Nachos to induce a context switch. You need to implement ***TestConsistency()*** by creating two threads with *Inc_Consistent* and two threads with *Dec_Consistent*. The current thread should wait for all those threads to complete. At the end of ***TestConsistency()***, a checking is performed on whether the value is 0.

If the checking passes, you should get the message "congratulations! passed.". Otherwise, an error message is printed.

Code:

```
//Exercise 2: offer an implementation of Inc and Dec so that
//no matter what kind of interleaving occurs, the result value should be consistent.

//1. Declare any paramters here.
#include "synch.h"

//fill your code
Semaphore *s = new Semaphore("Semaphore_Consistent", 1);

//2. implement the new version of Inc: Inc_Consistent
void Inc_Consistent(_int which)
{
        //fill your code
        s->P();
        currentThread->Yield();
        int a=value;
        currentThread->Yield();
        a++;
        currentThread->Yield();
        value=a;
        currentThread->Yield();
        //currentThread->Yield();
        printf("**** Inc thread %d new value %d\n", (int) which, value);
        s->V();
}

//3. implement the new version of Dec: Dec_Consistent
void Dec_Consistent(_int which)
{
        //fill your code
        s->P();
        currentThread->Yield();
        int a=value;
        currentThread->Yield();
        a--;
        currentThread->Yield();
        value=a;
        currentThread->Yield();
        //currentThread->Yield();
        printf("**** Dec thread %d new value %d\n", (int) which, value);
        s->V();
}
```

```
//4. implement TestValueMinusOne by create two threads with Inc_Consistent and two threads
with Dec_Consistent
// you should pass the checking at the end, printing "congratulations! passed."
void TestConsistency()
{
        value=0;
        printf("enter TestConsistency, value=%d...\n", value);

        //fill your code
        Thread *it1 = new Thread("Inc_Consistent_1");
        Thread *it2 = new Thread("Inc_Consistent_2");
        Thread *dt1 = new Thread("Dec_Consistent_1");
        Thread *dt2 = new Thread("Dec_Consistent_2");

        it1->Fork(Inc_Consistent, 1, 1);
        it2->Fork(Inc_Consistent, 2, 1);
        dt1->Fork(Dec_Consistent, 3, 1);
        dt2->Fork(Dec_Consistent, 4, 1);

        currentThread->Join(it1);
        currentThread->Join(it2);
        currentThread->Join(dt1);
        currentThread->Join(dt2);


        //2. checking the value. you should not modify the code or add any code lines behind
        //this section.
        if(value==0)
                printf("congratulations! passed.\n");
        else
                printf("value=%d, failed.\n", value);
}
```

Output:

```
enter TestConsistency, value=0...
**** Inc thread 1 new value 1
**** Inc thread 2 new value 2
**** Dec thread 3 new value 1
**** Dec thread 4 new value 0
congratulations! passed.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 490, idle 0, system 490, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0, outs 0, tlb miss: 0
Network I/O: packets received 0, sent 0

Cleaning up...
```