COMP3511 Fall 2018 Project #1: Nachos and Thread

In this project you will learn how to create threads in Nachos. You are given a simple thread system for Nachos. Your tasks are:

1. Compile Nachos and run the system.
2. Add some lines to the Nachos code. Recompile and run it.
3. Add several specified lines in the Nachos code. Recompile and run it.
4. Save the output and explain the results.

Don't be overwhelmed by the sheer amount of code provided. In fact you don't need to care about most of it. The part that you need to read or modify will be given in this instruction.

Task 1: Running Nachos with pre-implemented semaphore

Step 1: download Nachos source code, with pre-implemented semaphore.

```
wget http://course.cse.ust.hk/comp3511/project/project1/os_nachos_proj1.tar.gz
```

Step 2: Extract the source code.

```
tar -zxvf os_nachos_proj1.tar.gz
```

Step 3: Compile the code

Enter the folder "os_nachos_ proj1" and then run "make"

Step 4: Run nachos

In the folder "os_nachos_ proj1", run "./nachos"
In this program, Nachos creates one thread, which does nothing except telling us its name "Thread1" and its termination.
If you succeed in running nachos, you will see these messages:

```
Hello, my name is Thread1
Thread1 ends
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 30, idle 0, system 30, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Step 5: Save the output to file

```
./nachos > project1_output1.txt
```

This command runs Nachos and saves the output to file project1_output1.txt
Keep the file project1_output1.txt for grading.

Task 2: Adding codes to Nachos

Open file os_nachos_ proj1/threads/project1.cc. All the code you need to write is in this file. There are 4 functions in the file. running_for_calculation(), latter_thread(), prior_thread() and project1(). Do **NOT** modify the function running_for_calculation().
Your work is to add your code in function **latter_thread(), prior_thread() and project1()**. Nachos will automatically invoke project1(). Your tasks are:

Step 1:
In Thread1 (i.e. function prior_thread()), invoke function running_for_calculation() to perform certain number of calculations specified by the parameter "arg" . Before the function running_for_calculation() is invoked, output the information which claims that Thread1 will perform calculation for "arg" * 10000 times.

Step 2:
Create second thread called Thread2. This thread invokes function latter_thread ().

Step 3:
In Thread2 (i.e. function latter_thread()), invoke function running_for_calculation() to perform certain number of calculations specified by the parameter "arg" . Before the function running_for_calculation() is invoked, output the information which claims that Thread2 will perform calculation for "arg" * 10000 times.

Step 4:
Add **two** code lines in Thread1 (i.e. function prior_thread(), under Task-Comment 1) and Thread2 (i.e. function latter_thread(), under Task-Comment 7) to get the priority of the current thread and output it. Please read the source file *thread.h* and *thread.cc* to find the proper function to get the priority of current thread.

Don't know what to do? Here are some instructions that may be helpful.
1. How to create threads in Nachos?
First, you need to define a Thread object, then invoke Fork():

```
Thread *th1 = new Thread("Thread1");
th1->Fork(prior_thread, 6);
```

A thread named "Thread1" will be created, and it invokes prior_thread as its working thread function. The working thread function must have a parameter of int type. e.g. prior_thread() function has a "int arg" parameter. The second line invokes the working thread function and passes the value 6 to the parameter.
Note: Do **NOT** define the object like "Thread th1("Thread1")". This will cause Nachos to crash.
2. How to get the name of a thread in Nachos?

```
currentThread->getName();
```

This function returns the name (pointer of char *) of current thread.
3. About running_for_calculation(int t) function
This function is provided to you. Do NOT modify it. This function will perform the calculation of addition for certain times specified by parameter t. When the calculation is completed, current

thread invokes Yield() function to relinquish the CPU if any other thread is ready to run. If so, current thread is put at the end of the ready list, so that it will then eventually be re-scheduled.
4. About latter_thread(int arg) function
This function will make Thread2 invoking running_for_calculation() function as Thread1 did previously.

Step 5:
After you finish coding in Step 2, re-run "make" and "./nachos". Your output should be:

```
Hello, my name is Thread1
my priority is 10
Thread1 will perform calculation for 60000 times
Hello, my name is Thread2
my priority is 10
Thread2 will perform caculation for 70000 times
Thread1 ends
Thread2 ends
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 70, idle 0, system 70, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Your output should include the same value provide above, and 3 points must be satisfied:
1. Print the name and priority of Thread1 and Thread2.
2. Print the information claiming that thread1 is going to perform calculation for certain times with the right parameter.
3. Thread1 ends before Thread2

Step 6: Save the output to file

```
./nachos > project1_output2.txt
```

This command runs the Nachos and save the output to file project1_output2.txt
Keep project1_output2.txt for grading.

Task 3: Simple Thread Scheduling in Nachos

In this task, you are asked to fulfill the following 8 steps.

Step 1: Add four code lines in source file project1.cc
Add **two** code lines under Task-Comment 5 (within the function latter_thread()) and Task-Comment 12 (within the function prior_thread()) for printing the global variable “global”

information in the two threads and the printed message should claim the thread name where the global variable is. For example, the printed message should be in a format like:

```
The global variable in Thread1 is 1
```

Then, please add the code line

```
th1->Suspend();
```

under Task-Comment 2 of the source file project1.cc (within the function latter_thread()). Please do NOT delete the comment line.

Add the code line

```
th1->Resume();
```

under Task-Comment 6 of the source file project1.cc (within the function latter_thread()). Please do NOT delete the comment line.

Step 2: Rebuild the project and re-run "nachos"

Enter the project1 folder and run "make" again to rebuild the project. Make sure the building process is without errors or you should revise the code and get rid of them. When you run "./nachos", your output should be:

```
Hello, my name is Thread1
my priority is 10
Thread1 will perform calculation for 60000 times
Hello, my name is Thread2
my priority is 10
Student's suspend routine called
Thread2 will perform calculation for 70000 times
The global variable in Thread2 is 5
Thread2 ends
The global variable in Thread1 is 8
Thread1 ends
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 90, idle 0, system 90, user 0
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Step 3: Save the output to file

Right after the previous step, please run command

```
./nachos > project1_output3.txt
```

which saves output of the modified "nachos" to project1_output3.txt.

Step 4: Scheduling the two threads in a different order

Please **comment out** your comment under Task-comment 2 and Task-Comment 6, and then **add** two code lines to proper position of prior_thread() for scheduling the two threads in a **different order** comparing to Step 1.

Step 5: Rebuild the project and save the output to file

Enter the project1 folder and run "make" again to rebuild the project. Please run command

```
./nachos > project1_output4.txt
```

which saves output of the modified "nachos" to project1_output4.txt.

Step 6: Scheduling the two threads with an additional command line

Please **comment out** your comment under Task-comment 8 and Task-Comment 13.
Then, please add the code line

```
currentThread->Yield();
```

under Task-Comment 11 of the source file project1.cc (within the function prior_thread ()).
Please do NOT delete the comment line.

Step 7: Rebuild the project and save the output to file

Enter the project1 folder and run "make" again to rebuild the project. Please run command

```
./nachos > project1_output5.txt
```

which saves output of the modified "nachos" to project1_output5.txt.

Task 4: Understand the core functions in scheduler.cc.

In this task, you are asked to understand several functions related to thread scheduling. In specific, you need to focus on threadtest.cc, scheduler.h, scheduler.cc, list.h, list.cc.
Please read the code carefully. Try to understand how the given scheduling algorithm is implemented. Here you are required to briefly answer how the following three functions work during the thread scheduling.

1. ReadyToRun()
2. FindNextToRun()
3. ShouldISwitch()

You are required to write the answers in project1_functions.txt.

Task 5: Answer the following questions and save the answers in in project1_report.txt

1. Why did we use the function *running_for_calculation()?*
2. What kind of service does the function *currentThread->Yield()* provide?

3. What is the relationship between the function *Fork()* and the function *prior_thread(int arg)?*

4. Explain how were the values of 'The global variable in Thread1' and 'The global variable in Thread2' generated in project1_output4.txt.

5. Find out the difference among the outputs of task 3 which are saved in project1_output3.txt, project1_output4.txt and project1_output5.txt, respectively. Describe and explain the difference briefly.

You are required to write the answers in project1_report.txt.
Note: Keep project1.cc, project1_output3.txt, project1_output4.txt , project1_output5.txt and project1_report.txt for grading.

After you finish these tasks:

1) Please generate a single file using ZIP and submit it through CASS.
2) The name of the ZIP should be "proj1_********.zip", using your student ID to replace star symbols.
3) The following files should be included insides the ZIP:

File Name	Description
project1.cc	Source file you have accomplished by the end of Task 3
project1_output1.txt	Output of Task 1
project1_output2.txt	Output of Task 2
project1_output3.txt	The first output of Task 3
project1_output4.txt	The second output of Task 3
project1_output5.txt	The third output of Task 3
project1_functions.txt	The answer to the questions in Task 4
project1_report.txt	The answer to the questions in Task 5