CMPT 300 Project #1: Threads

In this project you will learn how to handle threads in Nachos. You are given a simple thread system, and your tasks are:

1. **Compile Nachos and run the system.**
2. **Add some lines to the Nachos code, then recompile and run it.**
3. **Delete one specified line in the Nachos code. Recompile and run it.**
4. **Save the output and explain the results.**
5. **Write a more complex program to parse the argument and print out vows and consonants.**

Please don't be overwhelmed by the sheer amount of code provided. In fact you don't need to worry about most of it. The parts that you need to read or modify are given in the following instructions. Please read them carefully, and follow the steps.

Task 1: Run Nachos

Step 1: Download Nachos source code

```
wget http://www.cs.sfu.ca/CourseCentral/300/zonghuag/projects/project1.tar.gz
```

Step 2: Extract the source code

```
tar zxvf project1.tar.gz
```

Step 3: Compile the code

Enter the folder "project1/threads" and then run "make". Then 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 > exp1_output1.txt
```
This command runs Nachos and saves the output to file exp1_output1.txt

Note: Keep the file exp1_output1.txt for grading.

Task 2: Add code to Nachos

Open file threads/threadtest.cc. All the code you need to write is in this file. There are 4 functions in the file, busy_for_some_time(), wait_for_busy_thread(), busy_thread() and ThreadTestSimple() . DO NOT modify the function busy_for_some_time() and wait_for_busy_thread().All changes you make to Nachos source code must be clearly marked as follows:

```
//project 1 changes start here
<put your changes here>
//project 1 changes end here
```

Your work is to add your code in function busy_thread() and ThreadTestSimple(). Your tasks are:

Step 1:

In Thread1 (i.e. in function busy_thread()), invoke function busy_for_some_time() to a certain time specified by the parameter "arg" . Before the function busy_for_some_time() is invoked, output the information which claims Thread1 will busy for a time slot of "arg".

Step 2:

Create another thread called Thread2. This thread invokes function wait_for_busy_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(busy_thread, 1);
```

A thread named "Thread1" will be created, and it invokes busy_thread as its working thread function. The working thread function must have a parameter of int type, e.g., busy_thread() function has a "int arg" parameter. The second line invokes the working thread function and passes the value 1 to the parameter.

Note: DO NOT define the object like "Thread th1("Thread1")". This will cause Nachos to crash.

At this point, you don't need to worry about the thread switching strategy.

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 busy_for_some_time(int t) function

This function is provided to you. Do NOT modify it. This function will pause the thread for some time and make other threads run. Parameter t is the time you want it pause for. The unit is approximately 0.1 second.

4. About wait_for_busy_thread(int arg) function

The system call `th1->Join()` within Thread2 will make Thread2 wait for Thread1's termination before Thread2 can continue its execution. Notice that the parameter "arg" in this function is of no use. We write it like this just to satisfy the parameter format of the Fork() function.

After you finish your coding in Step 1, re-run "make" and "./nachos". Your output should look like the following:

```
Hello, my name is Thread1
Thread1 will keep busy for a time slot of 1
Hello,my name is Thread2
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 is not necessarily identical, but it must fulfill three requirements:

1. Print the name of Thread1 and Thread2.
2. Print the information claiming that thread1 is going to keep busy for a time with the right parameter. Notice that this parameter in the output must be the same with the parameter passed to the busy_for_some_time() function.
3. Thread1 ends before Thread2.

Step 3: Save the output to file

```
./nachos > exp1_output2.txt
```

This command runs the Nachos and save the output to file exp1_output2.txt

Task 3: Simple Multithreaded Programming Exercise

In this task, you are asked to fulfill the following 4 steps.

Step 1: Delete one code line in source file threadtest.cc which you have done by the end of Task 2

Delete the code line

```
th1->Join();
```

in the function wait_for_busy_thread(). If you want to easily restore this line afterwards, you can put a double-slashes // in the beginning of the line to comment it out instead of simply deleting.

Step 2: Rebuild the project

Rebuild the project. Make sure the building process is without errors or you should revise the code and get rid of them.

Step 3: Rerun "nachos" and save the output to file

Right after the former step, please run command

```
./nachos > exp1_output3.txt
```

which saves output of the modified "nachos" to exp1_output3.txt.

Step 4: Compare the outputs in Task 2 and 3

Find out the difference between the outputs in Task 2 and 3 which are saved in exp1_output2.txt and exp1_output3.txt respectively. Describe and explain the difference briefly. You are required to write the answer in exp1_report.txt.

Task 4: Multithreaded Programming

Modify Nachos main() function so that it accepts a new option "-a". This option can be followed by a phrase of unspecified length. For example, the nachos executable can be run as:

```
prompt% nachos –a Object Oriented Programming at SFU
```

Write a function `Threadtest` so that it creates two threads (vow and cons). The threads should take turns printing the respective words of the phrase supplied on the command line *while preserving the original word sequence in the argument list*. The vow thread should print all the words that start with a vowel and the cons thread should print all words starting with a consonant. Note that the `ThreadTest` should not print anything itself – the output should be supplied by the threads it creates. Note that the order of the words in the phrase should not be changed in the printout. Your program should work for a phrase of any reasonable length, not just the one given in the example. The output of your program should look similar to the following.

```
prompt% nachos -a Object Oriented Programming at SFU
Entering main
vow: Object
vow: Oriented
cons: Programming
vow: at
```

```
cons: SFU
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
[…]
Cleaning up...
prompt%
```

Run

```
./nachos –a Object Oriented Programming at SFU > exp1_output4.txt
```

which saves output of the modified "nachos" to exp1_output4.txt.

Hint: declare a globally accessible list of stings `argumentlist`, store the command line arguments there and use list handling routines specified in `threads/list2.h`, `threads/list2.cc` to manipulate your list. Do not use `list.h` and `list.cc`, since `List` is defined with C++ templates, which make it difficult to handle list of strings, while `List` in `list2.h` and `list2.cc` are defined as `void *`. We have defined and filled in the `argumentlist` data structure in `main.cc`, so you can use it in `threadtest.cc`.

The system call `Yield()` lets the calling thread give up control to the OS scheduler, in order to give other active threads a chance to run, The two threads should use `Yield()` to take turns in printing out the vows and cons, in order to preserve the original word sequence in the argument list. No mutex protection is needed here. If your program output does not preserve the original word sequence, i.e., it prints out all the vows before printing out all the cons, then you will get partial credit.

Since Tasks 1-3 and Task 4 are independent pieces, we have defined the following headers in `main.cc` for your convenience:

```
#define THREADTESTSIMPLE
//#define THREADTEST
```

If `THREADTESTSIMPLE` is defined, then `ThreadTestSimple()` is invoked by `main()`, and you are working on Tasks 1-3; If `THREADTEST` is defined, then `ThreadTest()` is invoked, and you are working on Task 4.

After you finish these tasks:

1) Please make a single tar.gz file, and submit it. (the command is "`tar cvf project1.tar file1 file2…`")
2) The name of the ZIP should be "proj1_********.tar.gz". (* as student ID)
3) The following files should be included inside the bundle:

File Name	Description
threadtest.cc	Source file
exp1_output1.txt	Output of Task 1
exp1_output2.txt	Output of Task 2
exp1_output3.txt	Output of Task 3
exp1_output4.txt	Output of Task 4

exp1_report.txt	The answer to the question in Step 4 of Task 3