# Nachos: Non-Preemptive Multiprogramming

## 1 An Important Reminder

Review the university's academic integrity policy (http://class.syr.edu/academic-integrity). Remember that violating academic integrity policy will significantly jeopardize your grade.

## 2 Learning Objectives

After completing this project assignment, students can

1. describe the difference between a preemptive and a non-preemptive scheduler,
2. describe the role of `Yield()` function,
3. describe the effect of the timer device in thread execution sequence,
4. eventually implement a non-preemptive multi-programming in Nachos.

## 3 Preliminary Requirement

Get a fresh copy of Nachos from `nachos.tar`. Refer to Lab 1.

## 4 Preemptive and Non-Preemptive Scheduling

### 4.1 Preliminary Requirement for this section

- Download `threadtest.cc` files from Bb and place them in the right directory in your Nachos directory.
- Turn off the timer device by commenting out `alarm = new Alarm(randomSlice);` in line 99 of `kernel.cc`. By turning the timer device off, you will be able to see more predictable outputs.
- Compile Nachos in `build.linux`

**Question 1** *(E: 10 points) Run the following command. You should see the same output as below.*

```
build.linux>./nachos -K
*** thread 0 looped 0 times
*** thread 0 looped 1 times
*** thread 0 looped 2 times
*** thread 0 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
*** thread 2 looped 0 times
*** thread 2 looped 1 times
*** thread 2 looped 2 times
*** thread 2 looped 3 times
*** thread 2 looped 4 times
*** thread 3 looped 0 times
*** thread 3 looped 1 times
*** thread 3 looped 2 times
*** thread 3 looped 3 times
*** thread 3 looped 4 times
```

*Notice that* `kernel->currentThread->Yield();` *in* `void SimpleThread(int which)` *is commented out. Delete '*`//`*' to activate the line and recompile Nachos and run it with the same command* `./nachos -K`*. Explain why this program's output is different from the previous one.*

**Question 2** *(M: 10 points)* **Activate the Timer Device:** *Now turn on the timer device by activating line* `alarm = new Alarm(randomSlice);` *in line 99 of* `kernel.cc`*. Recompile Nachos and run it with the same command* `./nachos -K`*. My output looks like:*

```
build.linux>./nachos -K
*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 2 looped 0 times
*** thread 3 looped 0 times
*** thread 0 looped 1 times
*** thread 2 looped 1 times
*** thread 3 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 1 times
*** thread 2 looped 2 times
*** thread 3 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 2 times
*** thread 2 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 3 times
*** thread 2 looped 4 times
*** thread 3 looped 3 times
*** thread 1 looped 4 times
*** thread 3 looped 4 times
```

*Notice that after activating the timer device, my output looks different from the previous one. How is yours? Explain why they are different.*

## 5 Resolving the Second Limitation

### 5.1 Preliminary Requirement for this section

- Download `addrspace.[cc|h]` files from Bb and place them in the right directories in your Nachos directory.
- Turn off the timer device by commenting out `alarm = new Alarm(randomSlice);` in line 99 of `kernel.cc`. By turning the timer device off, you will be able to see more predictable outputs.

Recall that we identified three limitations in Nachos in supporting multi-programming.

- **First Limitation:** Nachos must be able to accept multiple user program names.
- **Second Limitation:** Nachos must be able to create an individual thread for each user program.

- **Third Limitation:** Nachos must be able to load multiple programs and create address space for each program in the main memory.

We resolved the first limitation in Project 1. In this project, we will resolve the second limitation. To resolve the third limitation, we will provide a simple contiguous memory management scheme that can load multiple programs in the memory without overwriting—i.e., a new `addrspace.[cc|h]`. Use the `diff` command to see the differences between the new `addrspace.[cc|h]` and the original files. Through these new `addrspace.[cc|h]` files, the memory management is done for you, but you will still need to modify Nachos to address the second limitation; See Section 6 below.

**Question 3** *(E: 10 pts) Compile Nachos and run it with the command* `./nachos -x ../test1/write -x ../test1/read`. *Nachos still won't run both* `write` *and* `read`. *Note that your Nachos version currently resolves the limitation 1 and 3, but not 2. How many user program threads are created in your current Nachos? Which block of code in* `main.cc` *creates the address space to run a user program?*

**Question 4** *(M: 10 pts) Carefully study the* `Fork(·)` *function defined in* `thread.[cc|h]`.

1. *What is the role of the first argument of the function* `Fork(·)`*?*
2. *If we use a pointer to the function* `RunUserProg(·)` *as the first argument of* `Fork(·)`*, what will be the effect?*

## 6 Programming Requirements

Carefully consider all the exercises above. With the new `addrspace.[cc|h]` files Nachos can load multiple user programs in the main memory, creating an address space for each program loaded. The algorithm in `RunUserProg()` function in `main.cc` shows an example for creating an address space. It loads the executable file in the code segment and executes the user program. Notice that the `AddressSpace::Execute()` function never returns. However, the current Nachos can only run one user program thread at a time.

**Question 5** *(M: 60 points)*

*To load and run multiple user programs, you need to create a new thread for each user program. Modify* `main.cc` *so it creates a user-level thread for each program specified by each* `-x` *flag. Hint: Study carefully how* `threadtest.cc` *creates multiple threads. Pay attention to the first argument of* `Fork(·)` *and think about which function pointer should be the first argument to run a user program.*

### 6.1 Tests and Output

Your program will be tested with the following commands:

- `./nachos -x ../test1/read -x ../test1/write`
- `./nachos -x ../test1/write -x ../test1/read`
- `./nachos -x ../test1/write -x ../test1/read -x ../test1/read-write`

Your program must be able to run all user programs in the command line, one-by-one.

## 7 What to Submit

You must submit two files to Bb: (1) The PDF report and (2) The source code zip file.

1. **PDF report:** You need to submit a detailed project report describing what you have done to implement the requirements. Answer all questions. Please also list the important code snippets followed by an explanation. These code snippets should include the modified part of the Nachos source code. Simply attaching code without any description will not receive credits.
2. **Source code zip file:** After "`make clean`" in `build.linux`, please compress your whole `nachos` folder, name the compressed file LastName,FirstName.zip (NOTE: .zip ONLY!)