CS330: Operating Systems
Assignment#2

1 Preamble

In this assignment, we will experiment with a few scheduling algorithms in NachOS. In the first part of the assignment, you will incorporate a way to submit a batch of jobs to NachOS at time zero. The second part explores the scheduling algorithms. The batch submission component will be needed to evaluate the performance of the scheduling algorithms.

Download cs330assignment2.zip and unzip it in your home directory. This will create a new directory named cs330assignment2/ and place my version of NachOS under it.

If you plan to continue using your submission from the last assignment, you need to do the following.

- Please copy the contents of the ∼/cs330assignment2/nachos/code/test/ directory into your test directory.
- Please set NumPhysPages to 512 in machine.h.
- You may have to change the following line in Makefile to point to the correct path for the MIPS cross-compiler. GCCDIR = ∼/cs330assignment2/mips-i386-xgcc/bin/

2 Batch Submission of Jobs

Modify nachos/code/threads/main.cc so that it can accept a new option -F followed by a plain text file as follows.

```
> ./nachos -F filename
```

The plain text file contains several lines, where each line specifies an executable user program name followed by an optional priority value of the job. The priority value must be an integer between 0 and 100 (both inclusive). A higher priority value signifies a lower priority (as in UNIX). If a priority value for a job is not mentioned, a default priority value of 100 is assumed. The file contents may look like the following.

```
../test/printtest 90
../test/vectorsum 50
../test/forkjoin
../test/testloop 60
../test/testloop
../test/vectorsum 70
```

NachOS reads this file, creates one thread corresponding to each specified executable, and enqueues it in the ready queue. After enqueuing all jobs in the ready queue, it invokes the scheduler to start the execution. At this point the main thread may exit. Every thread in the system must do a clean exit. In other words, when your main thread exits, it must update the relevant data structures with this information so that when a child exits later it

doesn't think that the parent hasn't exited yet. The easiest thing to do is to execute the SysCall_Exit code for the main thread after it has created all the children. Assume an exit code of zero for the main thread.

You should always compile and run NachOS from the nachos/code/userprog directory, as you did in the last assignment. The code, as provided, may not compile from the nachos/code/threads/ directory.

3 Scheduling Algorithm

NachOS implements a non-preemptive scheduling algorithm if the timer is off. In this assignment, we will experiment with the following algorithms: non-preemptive algorithm of NachOS, non-preemptive shortest next CPU burst first algorithm with an exponentially averaged CPU burst estimation, a preemptive fixed quantum round-robin scheduling algorithm, and a UNIX-like preemptive priority scheduling algorithm. The first three algorithms will ignore the priority values of the processes. In all evaluations, we will use the testloop.c, testloop1.c, testloop2.c, testloop3.c programs, which run several iterations of the vectorsum loop with either no I/O burst or a short I/O burst in each iteration. We will evaluate each algorithm in terms of CPU utilization and average wait time in the ready queue. You will submit four different batches of programs to NachOS. Each batch will contain ten copies of one of the four testloop programs i.e., the first batch will have ten copies of testloop, the second will have ten copies of testloop1, the third will have ten copies of testloop2, and the fourth will have ten copies of testloop3. We will refer to these as Batch1, Batch2, Batch3, and Batch4.

To decide the fixed quantum in the last two algorithms, first get an estimate of the CPU burst lengths in the testloop.c program. This can be done by running a copy of this program on NachOS in the presence of its non-preemptive scheduling algorithm and measuring the ticks between I/O operations. Next, take the average of these burst lengths. Let this average be A. Evaluate the algorithms for three different quanta: one-fourth, half, and three-fourth of A. Let these three quanta be Q1, Q2, and Q3. Next, find out the minimum value of the quantum that offers the maximum achievable CPU utilization for testloop. Call this Q4. The minimum quantum length that you are allowed to use for this assignment is twenty ticks. You will use this set of four quantum values to evaluate the round-robin and UNIX scheduling algorithms on the four batches. Note that you should not be recomputing A and Q1, Q2, Q3, Q4 for each of the batches. All batches must be evaluated using the same set of four quanta that you determine through testloop.c program.

For the UNIX scheduler, submit the ten jobs with equal priority value of 70 in each batch. This priority value is added to the base priority value of 50 at time zero. On each context switch, the priority values of all jobs are updated as in the UNIX scheduler (see the worked out example in Maurice Bach's book on page 253 as depicted in Figure 8.4 of Chapter 8). The CPU usage is measured in terms of ticks.

The first line of your input text file should be a positive integer indicating which scheduling algorithm to simulate. The mapping between the scheduling algorithms and positive integers is discussed below.

You must make sure that your NachOS simulation prints at least the following statistics at the end. You are welcome to provide more statistics, if they are useful; otherwise please do not clutter the output.

- Total CPU busy time
- Total execution time
- CPU utilization
- Maximum, minimum, average CPU burst length (considering only the non-zero bursts)
- Number of non-zero CPU bursts observed
- Average waiting time in the ready queue
- Maximum, minimum, average, and variance of thread completion times (exclude the main thread's completion time from these statistics)

4 Few Facts to Keep in Mind

1. For non-preemptive scheduling, you should not do a context switch on timer interrupts. There are two ways to do this. The simple, but naive, solution is not to invoke the Timer constructor if the simulated algorithm is non-preemptive. This is a bad solution because it doesn't allow you to use any of the timer related calls in the user program (this is not a problem for the test programs provided to you for this assignment). A better solution is to construct a timer, but not to call YieldOnReturn in the timer interrupt handler if the simulated algorithm is non-preemptive.

2. For this assignment, we will define a CPU burst to be the time for which a thread remains in the RUNNING state before transitioning to READY state (in the case of a preemptive context switch due to quantum expiry) or BLOCKED state (in the case of a non-preemptive context switch due to a blocking system call). This could be used for measuring the CPU burst lengths. However, you are free to come up with any other correct ways of measuring the CPU burst lengths. CPU utilization is the sum of all CPU burst lengths over the total execution time. Note that this definition of CPU utilization is somewhat wrong because it implicitly assumes that the CPU is not utilized outside the observed CPU bursts of the submitted processes. This is not correct because even during context switches, the CPU is utilized for kernel activities. Therefore, it is probably safe to say that this definition of CPU utilization only keeps track of CPU utilization due to user programs. Even if it does not reflect the correct CPU utilization, we will stick to this definition for this assignment. The first CPU burst starts when the main thread's state is set to RUNNING. This is the starting time of the simulation as well. The simulation finishes when Halt() is called. You should not include zero-length CPU bursts as legitimate CPU bursts. They should not be used in your burst estimation algorithms. You should always use stats-¿totalTicks for measuring the length of the CPU bursts and all other timing stats such as waiting time, etc..

3. When setting the quantum length to 1/4, 1/2, 3/4th of average quantum, make sure to set the timer interval to match the quantum length. Also, when you experiment with the quantum length that maximizes the CPU utilization, set the timer interval to match the quantum length. Please make sure to mention these quantum lengths in your report so that we can properly test your submission.

4. We are interested only in the waiting time in the ready queue. Be careful not to include the time spent by a thread in BLOCKED state. During this time the thread is not waiting in the ready queue.

5. Depending on the setting of the timer interrupt interval, the scheduling quantum, and the start time of a CPU burst, it may happen that the quantum expiry time does not coincide with the time when the timer interrupt comes. This will be a very common situation. You need to check on every timer interrupt, if the on-going CPU burst length is at least the quantum length. If yes, you should do a context switch. The upshot is that certain CPU bursts may exceed the quantum, but the maximum overshoot is bounded by the timer interrupt interval. A nice way of checking this is to print out the maximum and minimum CPU burst lengths.

6. The scheduling quantum in a preemptive scheduling algorithm must always be positive.

7. You should use 0.5 for the value of 'a' in the burst estimation algorithm. Remember that the length of the next burst for each thread must be estimated independently. If you use some other value of 'a', you must document clearly the reason for that in your assignment report.

8. Regarding the UNIX scheduler:

(i) A process comes with an initial priority value. This priority value should be added to the default base priority value (50) to calculate the new base priority value for this process. This new base priority value should be used in all subsequent calculations of priority value for this process. For example, if a process comes with an initial priority value of 70, its base priority value will be 120 and the priority value of this process will never go below 120. On each context switch, the new priority value should be updated as in the UNIX scheduler assuming 120 as the base priority value. The initial priority value of a process serves the purpose of the nice value. The initial priority value of the main thread can be assumed to be zero for this assignment.

(ii) The CPU usage and priority values of all active threads (i.e., those which haven't called exit yet) are updated on every context switch (i.e., when the currently running thread moves out of the RUNNING state). Some of the threads may be waiting in the ready queue, while some others may be blocked on some event or I/O. The UNIX

scheduler does not update the priorities of the threads that are not in the ready queue, but in this assignment, we will ignore this fact and update the priorities of all the threads. Remember that a context switch can happen due to two reasons: expiry of quantum (preemptive context switch) and a blocking system call (non-preemptive context switch). Just like you ignore the zero-length CPU bursts in burst estimation, you should not update the priority value and usage of any thread if the currently concluded CPU burst was of length zero.

(iii) The currently running thread remains eligible for scheduling in the next quantum as well. So, remember to update the priority values of all threads including the one which has just completed its quantum before selecting the next thread.

(iv) For this assignment, you do not have to apply any additional rules to boost the priority of a thread when returning from sleep or when entering the kernel mode from user mode. You also do not need to increase the priority value of a thread when returning from kernel mode to user mode. These are normally found in a UNIX scheduler, but we will keep things simple and only update the priority values as discussed in (ii) above.

9. It may happen that there is no thread in the ready queue and the current thread yields and is executed again. Even in this case, at the yield point you will break its CPU burst and count these as two different CPU bursts.

5 What to Submit

[PART I] For each of the four batches (Batch1, Batch2, Batch3, Batch4), prepare two tables, one for CPU utilization and one for average wait time in ready queue. Each table should report the results for the following ten scheduling policies.

1. Non-preemptive default NachOS scheduling

2. Non-preemptive shortest next CPU burst first algorithm

3, 4, 5, 6. Round-robin with three different quanta and the minimum quantum that maximizes CPU utilization

7, 8, 9, 10. UNIX scheduler with three different quanta and the minimum quantum that maximizes CPU utilization

Put these tables along with explanations in a pdf document. Name the pdf file groupX.pdf (replace X by your assigned group id). Remember to mention the quantum lengths used in the cases 3, 4, 5, 6, 7, 8, 9, 10 above. Explain why you think the minimum value of quantum that you have got for maximum CPU utilization should actually maximize the utilization. Mention any downside of using such a quantum value.

[PART II] For evaluating the difference between the two non-preemptive algorithms, use testloop4 and testloop5 programs. The primary difference between these two programs is that the inner loop size of testloop5 is smaller compared to testloop4. Prepare a batch of ten jobs such that the first five jobs are testloop4 and the last five jobs are testloop5. We will refer to this batch as Batch5. For this batch, report the average waiting time in the ready queue for the two non-preemptive algorithms. Explain your results. Include these results and explanations in the pdf document.

[PART III] For all the five batches, report the overall estimation error in CPU bursts for the second non-preemptive algorithm. This is how you should compute the error: Compute the absolute value of the error in CPU burst estimation at the end of each non-zero burst. Accumulate these into a single sum. Report the ratio of this sum to the sum total of all CPU bursts. The smaller this number, the better is the estimation. What happens to this ratio if you increase the OUTER_BOUND in the test programs used to prepare these batches? Explain your observation. Include these results and explanations in the pdf document.

[PART IV] For evaluating the difference between round-robin and the UNIX scheduling algorithms, prepare a batch of ten testloop.c jobs. We will refer to this batch as Batch6. Assign priority values to these jobs in the order 100, 90, ..., 10. Set the quantum length and timer interval both to 100. Report the maximum, minimum, average, and variance of job completion times (exclude the main thread's completion time from these statistics) for this batch running with round-robin and the UNIX scheduling algorithms. Explain your results. Completion time of a thread is defined as the value of totalTicks when the thread calls exit. Include these results and explanations in the pdf document.

Execute make clean in the threads and userprog directories. Prepare the submission zip ball of your machine/, threads/, and userprog/ directories.

```
> cd nachos/code/
> zip -r groupX.zip machine/ threads/ userprog/
```

Replace X by your assigned group id. Send an email to cs330autumn2017@gmail.com with subject "[CS330] Assignment2" and attach groupX.zip and groupX.pdf to the mail. The body of the mail should contain the following two sentences.

1. I have not copied any part of this submission from any other group.
2. I have not helped any other group copy any part of this submission.

All the members of the group should put their names and roll numbers below the statement.

6 Punishment for Cheating

Please refer to the section on academic integrity on course web page.