COMP 3511
Operating Systems

Project #2

Objectives and Tasks

- Run Nachos with Pre-implemented Scheduling System Skeleton
- Read source code of Nachos, implement ancillary data structure
- Implement SJF Scheduling Algorithms
- Explain the Results

Task 1

- **Task 1: Run Nachos with Pre-implemented Scheduling System Skeleton**
  - Step 1: Download Nachos source code of this project
  - Step 2: Extract the source code
  - Step 3: Compile the code
  - Step 4: Run nachos
  - Step 5: Read the code

Task 1

- Two scheduling algorithms
  - First Come First Serve (FCFS)
  - Shortest Job First (SJF)

Task 1

Executable File	Source File	Corresponding Algorithm	Already Implemented?
test0	test.0.cc	FCFS	Yes
test1	test.1.cc	SJF	No

Task 1

- *Read the codes*

- *ReadyToRun()*
  - decides the policy of placing a thread into ready queue (or multilevel queues) when the thread gets ready
- *FindNextToRun()*
  - decides the policy of picking one thread to run from the ready queue
- *ShouldISwitch()*
  - decides whether the running thread should preemptively give up to a newly forked thread

Task 2

- **Task 2: Implement PrintListSize() function of data structure “List”**

- ReadyList:
  - Record the threads ready to be executed.
  - Described in list.h and list.cc

- PrintListSize()：
  - Print the number of threads currently waiting in the readyList

Task 3

- **Task 3: Implement SJF Scheduling Algorithms**
- Shortest Job First
- Only modify *scheduler.cc*
  - Scheduler::ReadyToRun
  - Scheduler::FindNextToRun
  - Scheduler::ShouldISwitch

Task 3

- Shortest Job First
  - the thread with the shortest burst time in the ReadyList should be scheduled for running after the current thread is done with burst.
  - If there are more than one thread with the same shortest burst time in the ReadyList, they must be scheduled in FCFS manner
  - Return first thread when scheduler needs to pick one thread to run

Hint: insert the thread to ReadyList according to its burst time when a thread gets ready.

Task 3

- Shortest Job First
  - *Hint*: insert the thread to ReadyList according to its burst time when a thread gets ready.
    - Make use of the function *SortedInsert()* in *List.cc*
    - Example: list->SortedInsert(thread,thread->getPriority()); this line of code insert the thread into the list based on its priority.

Task 3

- Compile and Run
- Save your outputs to project2_test1.txt,
- Keep your source code scheduler.cc

Task 4

- Explain the Results

1. Understand the output of test0 (FCFS scheduling) and test1 (SJF scheduling). Then calculate the following performance metrics of each scheduling algorithms:
   a) Average waiting time;
   b) Response time;
   c) Turn-around time.

2. Compare the performance among the two scheduling algorithms in the aspects mentioned in question 1, then discuss the pros and cons of each scheduling algorithms. (Note: you are strongly encouraged to change the input threads in *test.0.cc and test.1.cc* in order to make your discussion more convincing. However, when submitting the outputs of test1, please do submit the outputs with the original input threads.)

Outputs

- Please generate a single file using ZIP and submit it through CASS
- Name of the ZIP: "proj2_********.zip" (* as student ID)
- Inside the ZIP file:

File Name	Description
scheduler.cc list.cc	Source codes you have accomplished by the end of Task3
project2_test0.txt	Output of test0 in Task2
project2_test1.txt	Output of test1
project2_report.txt	The answer to the questions in Task 4