Nachos Term Project #2-1

❑ Project Overview

Priority Scheduler 구현 및 Condition variable 동작 원리 이해

- **Project #2은 nachos/proj1 디렉토리에서 빌드 및 실행할 것.**
  - **Priority Scheduler 적용을 위해서 nachos.conf의 내용 수정 필요. Default로 Round-Robin Scheduler로 설정되어 있음**
- **과제 제출 관련**
  - **제출 마감일: (※수업 시작 전까지 제출할 것. 이 후 제출물은 평가 대상 제외)**
    - **① 화요일 분반: 2019.5.14(화)**
    - **② 목요일 분반: 2019.5.16(목)**
  - **제출물:**
    - **① Project Report (프린트 제출)**
    - **② 소스 코드 전체 (nachos 폴더 및 하위 폴더 전체 압축해서 메일로 제출: blee@kgu.ac.kr)**
    - **③ zip 파일로 제출 - 파일이름: 수요일_2017101010홍길동_201727272_김길동.zip**

❑ Task 1

(40 Points) Implement priority scheduling in Nachos by completing the PriorityScheduler class. Priority scheduling is a key building block in real-time systems. Note that in order to use your priority scheduler, **you will need to change a line in nachos.conf that specifies the scheduler class to use.** The ThreadedKernel.scheduler key is initially equal to nachos.threads.RoundRobinScheduler. You need to change this to nachos.threads.PriorityScheduler when you're ready to run Nachos with priority scheduling.

Note that all scheduler classes extend the abstract class nachos.threads.Scheduler. In choosing which thread to dequeue, the scheduler should always choose a thread of the highest effective priority. If multiple threads with the same highest priority are waiting, the scheduler should choose the one that has been waiting in the queue the longest.

❑ Task 2

(30 Points) Implement the priority scheduler so that it donates priority, where possible. Be sure to implement Scheduler.getEffectivePriority(), which returns the priority of a thread after taking into account all the donations it is receiving. Note that while solving the priority donation problem, you will find a point where you can easily calculate the effective priority for a thread, but this calculation takes a long time. To receive full credit for the design aspect of this project, you need to speed this up by caching the effective priority and only recalculating a thread's effective priority when it is possible for it to change. It is important that you do not break the abstraction barriers while doing

this part -- the Lock class does not need to be modified. Priority donation should be accomplished by creating a subclass of ThreadQueue that will accomplish priority donation when used with the existing Lock class, and still work correctly when used with the existing Semaphore and Condition classes.

❑ Task 3

(30 Points) Task 1과 Task 2가 정상 동작하는지 확인하기 위한 Test 프로그램 작성

- ThreadedKernel.java의 selfTest 함수를 수정