Implement Priority Scheduling in NachOS

2016125003 구영민

2023. 05. 24.

Requirement Analysis

본 과제의 요구 사항은 다음과 같다.

1. FIFO로 구현되어 있는 NachOS의 스케줄러를 Priority로 수정한다.

Source Code Analysis

NachOS 3.4 소스 코드를 내려받아 소스 코드의 분석을 진행하였다.

Thread

threads/thread.h 파일에 Thread 클래스가 구현되어 있다.

```
class Thread {
    ...
};
```

우선 순위를 저장하는 필드가 아직 존재하지 않는다.

Scheduler

threads/scheduler.cc 파일에 ReadyToRun() 함수와 FindNextToRun() 함수가 구현되어 있다.

```
void Scheduler::ReadyToRun (Thread *thread) {
    DEBUG('t', "Putting thread %s on ready list.\n", thread->getName());

    thread->setStatus(READY);
    readyList->Append((void *)thread);
}

Thread *Scheduler::FindNextToRun() {
    return (Thread *)readyList->Remove();
}
```

readyList->Append 함수와 Remove 함수를 이용하여 FIFO를 구현하고 있다.

Test Code

threads/threadtest.cc 파일에 테스트 코드를 구현하고 실행할 수 있도록 설계되어 있다.

```
int testnum = 1; // testnum is set in main.cc

void ThreadTest() {
    switch (testnum) {
    case 1:
```

```
        ThreadTest1();
        break;
    default:
        printf("No test specified.\n");
        break;
    }
}
```

main.cc 파일에서 args를 파싱해 testnum 변수를 설정하도록 구성되어 있다.

```
extern int testnum;

int main(int argc, char **argv) {
    testnum = atoi(argv[1]);
}
```

위 테스트를 실행하기 위해서는 다음과 같이 실행할 수 있다.

```
$ threads/nachos -q 1

*** thread 0 looped 0 times
*** thread 1 looped 0 times
*** thread 0 looped 1 times
*** thread 1 looped 1 times
*** thread 0 looped 2 times
*** thread 1 looped 2 times
*** thread 0 looped 3 times
*** thread 1 looped 3 times
*** thread 0 looped 4 times
*** thread 1 looped 4 times
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 130, idle 0, system 130, 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...
```

Plan

1. threads/thread.h 파일에 우선 순위를 저장하는 필드를 추가하고, 게터와 세터를 추가한다.

2. threads/scheduler.cc 파일의 ReadyToRun() 함수와 FindNextToRun() 함수가 우선 순위 큐를 사용하도록 수정한다.
3. 우선 순위에 맞게 스케줄링 되는지 테스트 코드를 작성하여 확인한다.

Implement

threads/thread.h 파일에 우선 순위를 담는 priority 필드를 및 상응하는 세터와 게터를 추가한다.

```
class Thread {
  public:
    void setPriority(int pr) { priority = pr; }
    int getPriority() { return priority; }

  private:
    int priority;
};
```

threads/scheduler.cc 파일의 ReadyToRun() 함수와 FindNextToRun() 함수를 수정한다.

```
void Scheduler::ReadyToRun (Thread *thread) {
    DEBUG('t', "Putting thread %s on ready list.\n", thread->getName());

    thread->setStatus(READY);
    readyList->SortedInsert((void *)thread, thread->getPriority());
}

Thread *Scheduler::FindNextToRun() {
    return (Thread *)readyList->SortedRemove(NULL);
}
```

NachOS 구현의 List에는 우선 순위 큐를 사용하기 위한 SortedInsert() 함수와 SortedRemove() 함수가 미리 구현되어 있어, 호출하는 함수만 수정하면 우선 순위 큐로의 수정을 손쉽게 할 수 있다.

이제 테스트 코드를 작성한다. 우선 순위 스케줄링이 도입되었으므로, 정렬되지 않은 순서대로 쓰레드를 Fork()를 통해 실행하여도, 쓰레드의 우선 순위대로 정렬되어 실행되어야 한다.

```
void ForkFunction(int dummy) {
    printf("[ForkFunction] Current Thread's Priority: %d\n",
           currentThread->getPriority());
}

void PrioritySchedulingTest() {
    Thread *t1 = new Thread("t1");
    t1->setPriority(0);
    Thread *t2 = new Thread("t2");
    t2->setPriority(10000);
```

```
    Thread *t3 = new Thread("t3");
    t3->setPriority(5000);

    t1->Fork(ForkFunction, (void *) 0);
    t2->Fork(ForkFunction, (void *) 0);
    t3->Fork(ForkFunction, (void *) 0);
}

void ThreadTest() {
    switch (testnum) {
    case 1:
        ThreadTest1();
        break;
    case 3:
        PrioritySchedulingTest();
        break;
    default:
        printf("No test specified.\n");
        break;
    }
}
```

Test Result

세 개의 쓰레드에 각각 0, 10000, 5000의 우선 순위를 설정하고 실행하면, 아래와 같이 우선 순위가 낮은 쓰레드 순으로 실행되는 것을 확인할 수 있다.

```
$ threads/nachos -q 3

[ForkFunction] Current Thread's Priority: 0
[ForkFunction] Current Thread's Priority: 5000
[ForkFunction] Current Thread's Priority: 10000

No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

Source Code

본 보고서에 사용된 소스 코드는 https://github.com/youngminz/OperatingSystem/pull/5/files 에서 확인할 수 있다.