System Call and CPU scheduling on NachOS

2019Fall OS Project2

EE4 B05901064 林承德

Part1. System Call

Motivation

In order to implement the system call $Sleep()$, we need to do the following things :

- Add a list to store the sleeping threads

  Since the sleeping thread will be waken afterwards,
- implement the assembly and the exceptions.

Implementation

Define the system call

Add the definition at . $/code/userprog/syscall.h$, . $/code/test/start.s$, and . $/code/userprog/exception.cc$.

```
void
ExceptionHandler(ExceptionType which)
{
    int type = kernel->machine->ReadRegister(2);
    int val;

    switch (which) {
    case SyscallException:
        switch(type) {
        // pass
        case SC_Sleep:
            val=kernel->machine->ReadRegister(4);
            // cout << "Sleep Time:" << val << "(ms)."<< endl;
            kernel->alarm->WaitUntil(val);
            return;
        }
        break;
    }
}
```

In $ExceptionHandler()$, we call the function $WaitUntil()$ in class $Alarm$. Therefore, we add the sleepingList .

SleepingList

Since the threads is scheduled by the scheduler, I add the SleepingList to $class\ Scheduler$.

```
// ./code/threads/scheduler.h
class Scheduler {
  public:
    void FallAsleep(Thread*, int); // called by Alarm::WaitUntil to put the
thread to sleep
    bool WakeUp(); // called by Alarm::CallBack() to wake up the thread
    bool hasSleeping() { return !sleepingList.empty(); }

  private:
    typedef std::pair<Thread*, int> thread_clk;
    int current; // clock, add 1 when calling WakeUp()
    std::vector<thread_clk> sleepingList;
};
```

The list consists of the thread and the clock when it wakes up and puts to ready list later. When calling $Scheduler :: FallAsleep()$, (Thread, current + val) will put into $SleepingList$. When calling $Scheduler :: WakeUp()$, the scheduler will check every element in $SleepingList$ and put the threads that arrive their waken time to the ready list.

Result

There are two threads for testing :

- sleep : print 1 for every 10^5 time units.
- sleep1 : print 10 for every 10^6 time units.

```
Total threads number is 2
Thread test/sleep is executing.
Thread test/sleep1 is executing.
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:10
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:1
Print integer:10
Print integer:1
return value:0
Print integer:10
Print integer:10
Print integer:10
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 500000600, idle 499999671, system 300, user 629
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
```

Part2. CPU Scheduling

Motivation

In order to implement the CPU scheduling, we need to :

- edit the ready list in class $Scheduler$
- edit the condition of the yield.
- add argument that can choose scheduler and testcase of main.

Implementation

Ready list

set the proper property for the $readylist$.

- FCFS : FIFO Queue.
- SJF、Priority : List that sorted by the burst time or priority.

```
// ./code/threads/scheduler.cc

int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTime() == b->getBurstTime()) return 0;
    return a->getBurstTime() > b->getBurstTime() ? 1 : -1;
}
int PriorityCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority()) return 0;
    return a->getPriority() > b->getPriority() ? 1 : -1;
}

Scheduler::Scheduler(SchedulerType type) : schedulerType(type), current(0)
{
    switch(schedulerType) {
    case FCFS:
        readyList = new List<Thread *>;
        break;
    case SJF:
        readyList = new SortedList<Thread *>(SJFCompare);
        break;
    case Priority:
        readyList = new SortedList<Thread *>(PriorityCompare);
        break;
    }
    toBeDestroyed = NULL;
}
```

Yield

In default of $Alarm :: CallBack()$, it always try to yield. However, is depends on different CPU scheduling. So we need to ask scheduler whether to yield or not.

```
// ./code/threads/scheduler.cc

bool Scheduler:: needYield() {
    Thread *now = kernel->currentThread;
    Thread *next = readyList->GetFront();
    if(!next) return false;
    switch (schedulerType) {
        case FCFS      : return false;
        case SJF       : return SJFCompare(now, next) > 0;
        case Priority  : return PriorityCompare(now, next) > 0;
    }
}
```

Test Argument

> Usage : ./nachos [-s SchedulingType] [-t testcase]
>
> SchedulingType : FCFS, SJF, PRI
>
> testcase : 0, 1, 2

Add the argument in $main.cc$ for testing.

```
// ./code/thread/main.cc

for (i = 1; i < argc; i++) {
    if (strcmp(argv[i], "-s") == 0) {
        ASSERT(i + 1 < argc);   // next argument is debug string
        ++i;
        if(strcmp(argv[i], "SJF") == 0)type = SJF;
        else if(strcmp(argv[i], "FCFS") == 0)type = FCFS;
        else if(strcmp(argv[i], "PRI") == 0)type = Priority;
        else ASSERT(false);
    } else if (strcmp(argv[i], "-t") == 0) {
        ASSERT(i + 1 < argc);   // next argument is debug string
        ++i;
        testcase = atoi(argv[i]);
    }
}

kernel->Initialize(type);
kernel->SelfTest(testcase);
```

Result

There are three testcase written in $Scheduler :: SelfTest(inttestcase)$:

Since there is no additional test thread durning running, so there is no yielding between the test threads. However, yielding still happens when system threads appear.

testcase = 0

	A	B	C	D
Priority	5	1	3	2
Burst time	3	9	7	3
FCFS Order	1	2	3	4
SJF Order	1	4	3	2
Priority Order	4	1	3	2

testcase = 1

	A	B	C	D
Priority	5	1	3	2
Burst time	1	9	2	3
FCFS Order	1	2	3	4
SJF Order	1	4	2	3
Priority Order	4	1	3	2

testcase = 2

	A	B	C	D
Priority	10	1	2	3
Burst time	50	10	5	10
FCFS Order	1	2	3	4
SJF Order	4	2	1	3
Priority Order	4	1	2	3