Operating System Project 2

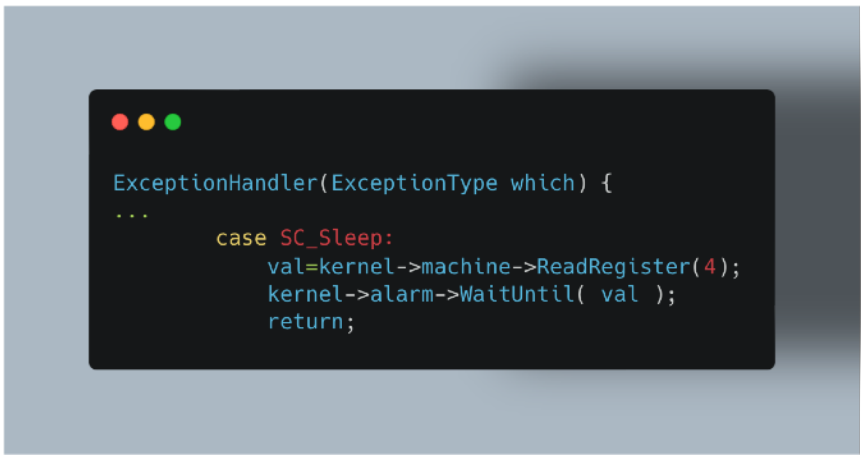
何孟遠
R11222007
November 23, 2023

Part 1

Implement a system call that can pause the program for several seconds.

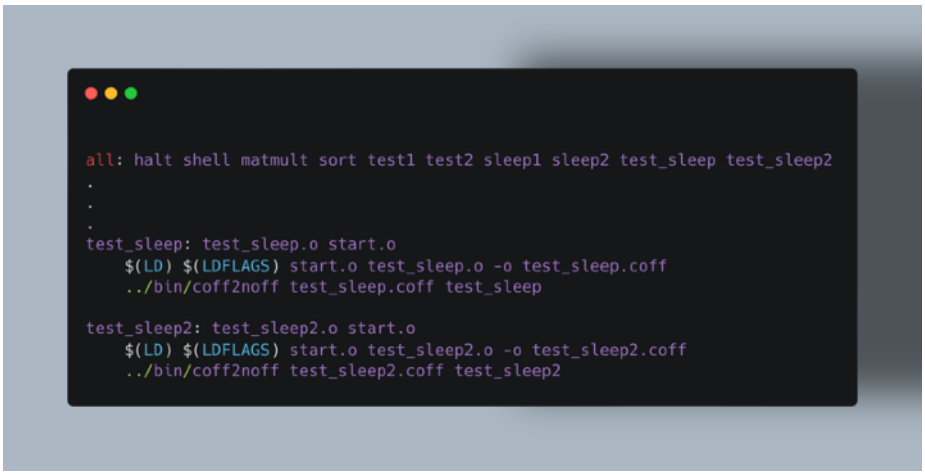
- Plans
  - Create a new type of a “system call”: Sleep.
  - Set the alarm for a thread.
    - Use a class to record the duration of sleeping and use a list to contain them.
  - Carefully decide which thread to wake up and let no other thread interrupt while sleeping.
- Code Modification
  - Create a new system call.
    - start.s & Exception

```
    .globl  Sleep
    .ent    Sleep
Sleep:
    addiu   $2,$0,SC_Sleep
    syscall
    j       $31
    .end    Sleep
```



```
ExceptionHandler(ExceptionType which) {
...
        case SC_Sleep:
            val=kernel->machine->ReadRegister(4);
            kernel->alarm->WaitUntil( val );
            return;
```

- Makefile



```
all: halt shell matmult sort test1 test2 sleep1 sleep2 test_sleep test_sleep2
.
.
.
test_sleep: test_sleep.o start.o
    $(LD) $(LDFLAGS) start.o test_sleep.o -o test_sleep.coff
    ../bin/coff2noff test_sleep.coff test_sleep

test_sleep2: test_sleep2.o start.o
    $(LD) $(LDFLAGS) start.o test_sleep2.o -o test_sleep2.coff
    ../bin/coff2noff test_sleep2.coff test_sleep2
```

- SleepThread & Thread list

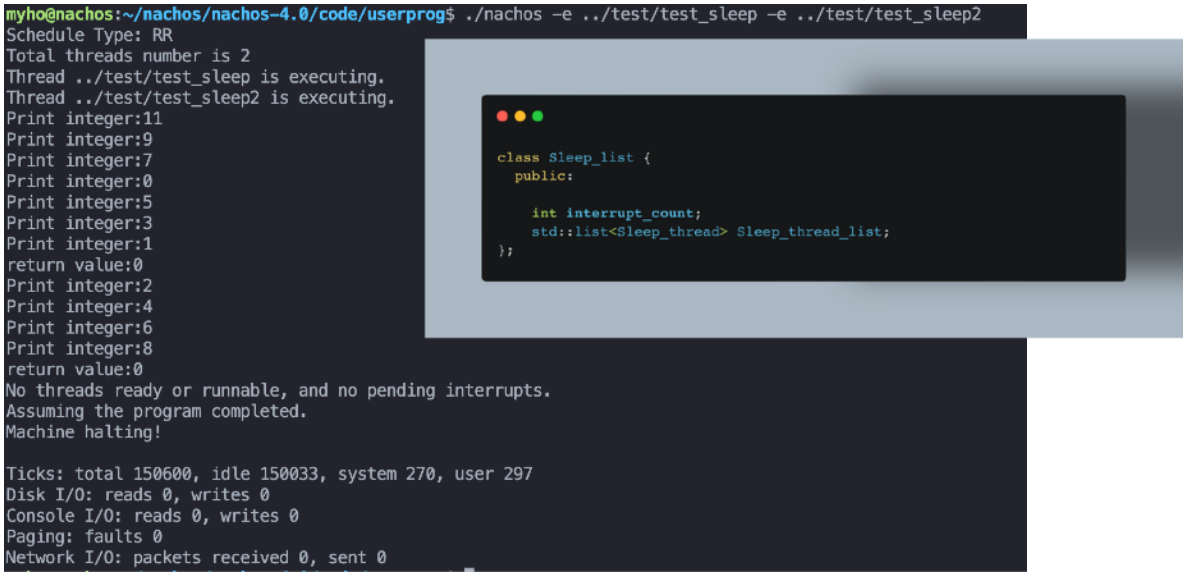
```
class Sleep_thread {
  public:
    Sleep_thread( Thread* t, int x) {
      thread_sleep  = t;
      sleep_time    = x;
    }
    Thread* thread_sleep;
    int sleep_time;
};

class Sleep_list {
  public:
    Sleep_list();

    void ToSleep( Thread* t, int x);
    bool ToReady();
    bool IsEmpty();
    std::list<Sleep_thread> Sleep_thread_list;
};
```

```
void
Alarm::CallBack()
{
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool wakeup = sleeplist.ToReady();

    if (status == IdleMode && !wakeup && sleeplist.IsEmpty())
    {   // is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
        timer->Disable();   // turn off the timer
    }
    } else {            // there's someone to preempt
        interrupt->YieldOnReturn();
    }
}

// Project 2 Add
void Alarm::WaitUntil( int x ) {
    // close interrupt
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread *t = kernel->currentThread;
    sleeplist.ToSleep(t, x);
    // open interrupt
    kernel->interrupt->SetLevel( oldLevel );
}

bool Sleep_list::IsEmpty() {
    return Sleep_thread_list.size() == 0;
}

void Sleep_list::ToSleep( Thread *t, int x) {
    ASSERT( kernel->interrupt->getLevel() == IntOff );
    Sleep_thread_list.push_back(
        Sleep_thread(t, kernel->stats->totalTicks + x) );
    t->Sleep(false);
}

bool Sleep_list::ToReady() {
    bool wakeup = false;
    for ( list<Sleep_thread>::iterator it = Sleep_thread_list.begin();
                    it != Sleep_thread_list.end(); it++ )
    {
        if (kernel->stats->totalTicks >= it->sleep_time ) {
            wakeup = true;
            kernel->scheduler->ReadyToRun( it->thread_sleep );
            it = Sleep_thread_list.erase( it );
        }

    }
    return wakeup;
}
```

- Problems & Results

```
myho@nachos:~/nachos/nachos-4.0/code/userprog$ ./nachos -e ../test/test_sleep -e ../test/test_sleep2
Schedule Type: RR
Total threads number is 2
Thread ../test/test_sleep is executing.
Thread ../test/test_sleep2 is executing.
Print integer:11
Print integer:0
Print integer:9
Print integer:7
Print integer:2
Print integer:5
Print integer:3
Print integer:4
Print integer:1
return value:0
Print integer:6
Print integer:8
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2100, idle 1533, system 270, user 297
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
```

- Problem:
    - At first, I set the tick (sleeping time) of the program inside the class “Sleep_list”.
    - However, I noticed that the rank of printed integers and the total ticks are different from the example.
    - I shouldn’t define a local variable inside the class and use it to decide whether the thread wakes up.
    - Instead, using “kernel->stats->totalTicks” will be a better way.



- Question: Explain the details of the function call path from Machine::Run to Alarm::CallBack().
    - Machine::Run: Run the nachos execution.
    - Machine::OneInstruction(instr);
        - Run every code in the user program.
    - Run “Sleep” System call in sleep.cc.
    - Raise the ExceptionHandler in userprog/exception.cc.
        - Case SC_Sleep in SyscallExcpetion.
        - Run kernel->alarm->WaitUntil(val).
    - Alarm::WaitUntil.
        - at threads/alarm.cc.
    - Interrupt::Idle.
        - at machine/interrupt.cc:212
    - Interrupt::CheckIfDue.
        - at machine/interrupt.cc:315.
    - Timer::CallBack. -> Callback object: Alarm.
        - at machine/timer.cc:56.
    - Alarm::CallBack().
        - at threads/alarm.cc:52
        - Check after every interrupt.

Part 2

- Plans: Implement different ways of CPU scheduling.
    - Design a test case in threads/thread.cc, and also print the info of every thread.
    - Design the interface to choose which scheduler to use.
    - Design the scheduler.
    - Prevent the other thread from preempting for the cases FCFS and SJF.
- Code Modification
    - Test case in threads/thread.cc.
    - The function “SchedulingTest()” should be added to ThreadedKernel::SelfTest().

```
// Project2 added
void Threadinfo() {
    Thread *thread = kernel->currentThread;
    while (thread->getBurstTick() > 0) {
        thread->setBurstTick(thread->getBurstTick() - 1);
        kernel->interrupt->OneTick();
        printf( "Thread: %s, remaining tick: %d.\n",
                kernel->currentThread->getName(),
                kernel->currentThread->getBurstTick() );
    }
}

void Thread::SchedulingTest() {

    const int thread_num = 4;
    char *name[thread_num] = {"A", "B", "C", "D"};
    int thread_priority[thread_num] = {5, 4, 7, 2};
    int thread_burst[thread_num] = {5, 9, 3, 7};

    Thread *t;
    for ( int i = 0; i < thread_num; i++ ) {
        t = new Thread( name[i] );
        t->setPriority( thread_priority[i] );
        t->setBurstTick( thread_burst[i] );
        t->Fork((VoidFunctionPtr) Threadinfo, (void *)NULL);
    }
    kernel->currentThread->Yield();
}
```

- Also, threads/thread.h.

```
class Thread {
  public:
    // Project2 add
    void setBurstTick( int t )  { burstTick = t; }
    int  getBurstTick()         { return burstTick; }
    void setStartTick( int t )  { startTick = t; }
    int  getStartTick()         { return startTick; }
    void setPriority( int p )   { threadPriority = p; }
    int  getPriority()          { return threadPriority; }
    static void SchedulingTest();

  private:
    // Project2 added
    int burstTick;        // the tick needed for a thread
    int startTick;        // the start tick of a thread
    int threadPriority;   // the priority of a thread

}
```

- Design the interface to choose which scheduler to use.
  - Need to add a new type “FCFS” in SchedulerType.

```
    // Project2 add
    SchedulerType type; // Scheduler type
    if(strcmp(argv[1], "FCFS") == 0) {
        type = FCFS;
        cout << "Schedule Type: FCFS" << endl;
    } else if (strcmp(argv[1], "SJF") == 0) {
        type = SJF;
        cout << "Schedule Type: SJF" << endl;
    } else if (strcmp(argv[1], "PRIORITY") == 0) {
        type = Priority;
        cout << "Schedule Type: Priority" << endl;
    } else {
        type = RR;
        cout << "Schedule Type: RR" << endl;
    }
```

- Design the scheduler in threads/scheduler.cc.

```
// Project2 add
Scheduler::Scheduler( SchedulerType type )
{
    schedulerType = type;
    switch (schedulerType) {
        case RR:
            readyList = new List<Thread *>;
            break;
        case SJF:
            readyList =
                new SortedList<Thread *>( SJFCompare );
            break;
        case Priority:
            readyList =
              new SortedList<Thread *>( PRIORITYCompare );
            break;
        case FCFS:
            readyList =
              new SortedList<Thread *>( FCFSCompare );
            break;
    }
    toBeDestroyed = NULL;
}
```

```
// Project2 add
int SJFCompare(Thread *a, Thread *b) {
    if(a->getBurstTick() == b->getBurstTick())
        return 0;
    else if (a->getBurstTick() > b->getBurstTick())
        return 1;
    else
        return -1;
}
int PRIORITYCompare(Thread *a, Thread *b) {
    if(a->getPriority() == b->getPriority())
        return 0;
    else if (a->getPriority() > b->getPriority())
        return 1;
    else
        return -1;
}
int FCFSCompare(Thread *a, Thread *b) {
    return 1;
}
```

- Here I use SortedList to decide which thread to execute first.
- Prevent the other thread from preempting in the cases of FCFS and SJF.
- In threads/alarm.cc.

```
void
Alarm::CallBack()
{
...
    } else {            // there
    // Only RR or Priority can p
        SchedulerType type =
            kernel->scheduler->g
        if ( type == RR || type
            interrupt->YieldOnRe
            cout << "Interrupt:
        }
    }
```

```
myho@nachos:~/nachos/nachos-4.0/code/threads$ ./nachos SJF
*** 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 1 looped 4 times
*** thread 0 looped 4 times
Thread: C, remaining tick: 2.
Thread: C, remaining tick: 1.
Thread: C, remaining tick: 0.
Thread: A, remaining tick: 4.
Thread: A, remaining tick: 3.
Thread: A, remaining tick: 2.
Thread: A, remaining tick: 1.
Thread: A, remaining tick: 0.
Thread: D, remaining tick: 6.
Thread: D, remaining tick: 5.
Thread: D, remaining tick: 4.
Thread: D, remaining tick: 3.
Thread: D, remaining tick: 2.
Thread: D, remaining tick: 1.
Thread: D, remaining tick: 0.
Thread: B, remaining tick: 8.
Thread: B, remaining tick: 7.
Thread: B, remaining tick: 6.
Thread: B, remaining tick: 5.
Thread: B, remaining tick: 4.
Thread: B, remaining tick: 3.
Thread: B, remaining tick: 2.
Thread: B, remaining tick: 1.
Thread: B, remaining tick: 0.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

- Of course, I need to modify every initializer, so that they can pass the scheduler type to the function of scheduler.
    - In main.cc.

```
kernel->Initialize( type );     // Project2
```

    - I set the type = RR (Round Robin) for default.
- Problems & Results
    - FCFS. Expected: A->B->C->D.
    - SJF. Expected: C->A->D->B.
    - Priority. Expected: D->B->A->C.
    - RR. Expected: Random.

```
myho@nachos:~/nachos/nachos-4.0/code/threads$ ./nachos FCFS
Schedule Type: FCFS
*** 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
Thread: D, remaining tick: 6.
Thread: D, remaining tick: 5.
Thread: D, remaining tick: 4.
Thread: D, remaining tick: 3.
Thread: D, remaining tick: 2.
Thread: D, remaining tick: 1.
Thread: D, remaining tick: 0.
Thread: B, remaining tick: 8.
Thread: B, remaining tick: 7.
Thread: B, remaining tick: 6.
Thread: B, remaining tick: 5.
Thread: B, remaining tick: 4.
Thread: B, remaining tick: 3.
Thread: B, remaining tick: 2.
Thread: B, remaining tick: 1.
Thread: B, remaining tick: 0.
Thread: A, remaining tick: 4.
Thread: A, remaining tick: 3.
Thread: A, remaining tick: 2.
Thread: A, remaining tick: 1.
Thread: A, remaining tick: 0.
Thread: C, remaining tick: 2.
Thread: C, remaining tick: 1.
Thread: C, remaining tick: 0.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

```
myho@nachos:~/nachos/nachos-4.0/code/threads$ ./nachos PRIORITY
*** 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 1 looped 4 times
*** thread 0 looped 4 times
Thread: D, remaining tick: 6.
Thread: D, remaining tick: 5.
Thread: D, remaining tick: 4.
Thread: D, remaining tick: 3.
Thread: D, remaining tick: 2.
Thread: D, remaining tick: 1.
Thread: D, remaining tick: 0.
Thread: B, remaining tick: 8.
Thread: B, remaining tick: 7.
Thread: B, remaining tick: 6.
Thread: B, remaining tick: 5.
Thread: B, remaining tick: 4.
Thread: B, remaining tick: 3.
Thread: B, remaining tick: 2.
Thread: B, remaining tick: 1.
Thread: B, remaining tick: 0.
Thread: A, remaining tick: 4.
Thread: A, remaining tick: 3.
Thread: A, remaining tick: 2.
Thread: A, remaining tick: 1.
Thread: A, remaining tick: 0.
Thread: C, remaining tick: 2.
Thread: C, remaining tick: 1.
Thread: C, remaining tick: 0.
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!
```

```
myho@nachos:~/nachos/nachos-4.0/code/threads$ ./nachos RR
Schedule Type: RR
*** 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
Interrupt: YieldOnReturn.
*** thread 1 looped 4 times
*** thread 0 looped 4 times
Interrupt: YieldOnReturn.
Thread: B, remaining tick: 8.
Thread: B, remaining tick: 7.
Thread: B, remaining tick: 6.
Thread: B, remaining tick: 5.
Thread: B, remaining tick: 4.
Thread: B, remaining tick: 3.
Thread: B, remaining tick: 2.
Thread: B, remaining tick: 1.
Interrupt: YieldOnReturn.
Thread: C, remaining tick: 2.
Thread: C, remaining tick: 1.
Thread: C, remaining tick: 0.
Thread: D, remaining tick: 6.
Thread: D, remaining tick: 5.
Thread: D, remaining tick: 4.
Thread: D, remaining tick: 3.
Interrupt: YieldOnReturn.
Thread: A, remaining tick: 4.
Thread: A, remaining tick: 3.
Thread: A, remaining tick: 2.
Thread: A, remaining tick: 1.
Thread: A, remaining tick: 0.
Thread: B, remaining tick: 0.
Interrupt: YieldOnReturn.
Thread: D, remaining tick: 2.
```

- Conclusion
    - All of the results meet our expectations.
- Problem
    - At first, I didn’t prevent interrupt when running the test case. The result of the FCFS case shows the same result as RR.
    - Since we should only run the other thread after the current thread is finished for the case FCFS and SJF, we modify threads/alarm.cc.
    - The result then be correct.