NTU Operating System Project 2

Motivation

- For the first task, We'd like to add sleep() function in system call that can help us call sleep in our program.
- For the second task, we'd like to implement CPU scheduling by FIFO(First-In-First-Out), SJF(Shortest-Job-First), Priority, RR(Round-Robin), and multi-level queue.

Implementation

Task1 - System Call

1. First of all, we need to define a new token, SC_Sleep, that compiler(scanner) can recognize in `code/userprog/syscall.h`.
2. So, we have to assign a number to SC_Sleep that it can return the value.

```
...
#define SC_ThreadFork   9
#define SC_ThreadYield  10
#define SC_PrintInt 11

/* ********************************************
  Self-defined
 ******************************************** */
#define SC_Sleep    12.
...
```

3. Then, observe the other define words like SC_PrintInt in the same file. You can see there're lots of declaration function for each system call such as `void PrintInt(int number);` or `void ThreadFork(void (*func)());`,etc.. So, we must declare a function for sleep system call.

```
...
/* ********************************************
  Self-defined
 ******************************************** */
void Sleep(int number);
#endif /* IN_ASM */
...
```

4. According to the assignment description file, another file should be checked is start.s in test/start.s.

   According to the comment above this file, it's a file to assist system calls to **NachOS kernel** by assembly language.

   > Start.s

- Assembly language **assist for user programs running on top of Nachos**. Since we don't want to pull in the entire C library, we define what we need for a user program here, namely Start and the system calls.

System call stubs:

- Assembly language assist to make system calls to the Nachos kernel. There is one stub per system call, that places the code for the system call into register r2, and leaves the arguments to the system call alone (in other words, arg1 is in r4, arg2 is in r5, arg3 is in r6, arg4 is in r7)
  The return value is in r2. This follows the standard C calling convention on the MIPS.

```
/* ****************************************
  Self-defined
 **************************************** */
    .globl  Sleep   /* Set Sleep to a global symbol that can be found in
real c++ code properly */
    .ent    Sleep   /* Set the entry point for Sleep */
Sleep:
    addiu   $2,$0,SC_Sleep
    syscall
    j       $31
    .end    Sleep
```

5. From assignment description file, we should catch the exception situation by userprog/exception.cc. So, we need to define a new case in this file.

   According to the comment above in exception.cc

> Entry point into the Nachos kernel from user programs. There are two kinds of things that can cause control to transfer back to here from user code:
>
> - **syscall** -- The user code explicitly requests to call a procedure in the Nachos kernel. Right now, the only function we support is "Halt".
> - **exceptions** -- The user code does something that the CPU can't handle. For instance, accessing memory that doesn't exist, arithmetic errors, etc.
>
> ...
>
> **ExceptionHandler**
>
> - Entry point into the Nachos kernel. Called when a user program is executing, and either does a syscall, or generates an addressing or arithmetic exception.
> - For system calls, the following is the calling convention:
>   system call code -- r2
>   arg1 -- r4
>   arg2 -- r5
>   arg3 -- r6
>   arg4 -- r7
>   The result of the system call, if any, must be put back into r2.
> - And don't forget to increment the pc before returning. **(Or else you'll loop making the same system call forever!**
> - "which" is the kind of exception. The list of possible exceptions are in machine.h.

So, observe the previous case, SC_PrintInt, we can guess the value stored in `kernel->machine->Readgister(4)`. The code is like this...

```
/* ****************************************
Self-defined
**************************************** */
case SC_Sleep:
    val=kernel->machine->ReadRegister(4);
    cout << "Sleep a while:" <<val << "(ms)" << endl;
    return;
```

But, this code **can not really stop the executing** instead of printing out a normal message. So, refer to the assignment note, we should check `kernel->alarm->WaitUntil()` function that'll really implement sleep function in **threads/alarm.h**. The result after adding the code is just like...

```
/* ****************************************
Self-defined
**************************************** */
case SC_Sleep:
    val=kernel->machine->ReadRegister(4);
    cout << "Sleep a while:" <<val << "(ms)" << endl;
    kernel->alarm->WaitUntil(val);
    return;
```

*More info about WainUntil() function and interrupt, you can check out here

6. **The most important part**

   Define a class named **sleepList** in alarm.h and implement the methods in alarm.cc. All the other description can check out OS-NachOS-HW1 and use VSode can be more clearly. Lazy to write it down.

7. Refer to OS-NachOS-HW1, you can write your own testing case or just use the ready-made test case on the web. Note that you must revise `Makefile` in `code/test/Makefile` like this...

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

sleep2: sleep2.o start.o
    $(LD) $(LDFLAGS) start.o sleep2.o -o sleep2.coff
    ../bin/coff2noff sleep2.coff sleep2
```

*Note that you must use **tab** in `Makefile`.

*The purpose of these manipulation is when you use `make` command in `code/Makefile`. It'll execute all `Makefile` that exist in each folder. It can be observed in `code/Makefile`

```
MAKE = make
```

```
LPR = lpr

all:
    cd threads; $(MAKE) depend
    cd threads; $(MAKE) nachos
    cd userprog; $(MAKE) depend
    cd userprog; $(MAKE) nachos
    cd filesys; $(MAKE) depend
    cd filesys; $(MAKE) nachos
    cd network; $(MAKE) depend
    cd network; $(MAKE) nachos
    cd bin; $(MAKE) all
    cd test; make all
```

*Error message I encountered:

```
...
../bin/coff2noff halt.coff halt
make[1]: execvp: ../bin/coff2noff: Permission denied
make[1]: *** [halt] Error 127
make[1]: Leaving directory `/home/sbk/NTU/Operating System/Project2/nachos-
4.0/code/test'
make: *** [all] Error 2
```

According to [Why do I get permission denied when I try use "make" to install something?] page on StackOverflow, just use `chmod 777 ./bin/coff2noff` in `./code` folder.

Task2 - CPU Scheduling

Testing

1. ***threads/thread.cc*** - create a test case

```
...
void threadBody()
{
    Thread *thread = kernel->currentThread;
    while (thread->getBurstTime() > 0)
    {
        thread->setBurstTime(thread->getBurstTime() - 1);
        kernel->interrupt->OneTick();
        printf("%s: remaining %d\n", kernel->currentThread->getName(),
kernel->currentThread->getBurstTime());
    }
}
void Thread::SchedulingTest()
{
    const int thread_num = 4;
    char *name[thread_num] = {"A", "B", "C", "D"};
    int thread_priority[thread_num] = {5, 1, 3, 2};
    int thread_burst[thread_num] = {3, 9, 7, 3};

    Thread *t;
```

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

2. ***threads/thread.h*** - define various method in header files and implement these method in c file

```
class Thread
{
    private:
        ...
    public:
        ...
        void setBurstTime(int t)    {burstTime = t;}
        int getBurstTime()            {return burstTime;}
        void setStartTime(int t)      {startTime = t;}
        int getStartTime()            {return startTime;}
        void setPriority(int t)     {execPriority = t;}
        int getPriority()               {return execPriority;}
        static void SchedulingTest();
        ...
    private:
        int burstTime;  // predicted burst time
        int startTime;  // the start time of the thread
        int execPriority;   // the execute priority of the thread
        ...
};
```

3. ***threads/kernel.cc*** - add `SchedulingTest()` function that we've defined in `thread.h in ThreadKernel`

```
void ThreadedKernel::SelfTest()
{
    ...
    LibSelfTest();              // test library routines
    currentThread->SelfTest();  // test thread switching
    Thread::SchedulingTest();   // Add this line
                                // test semaphore operation
    semaphore = new Semaphore("test", 0);
    ...
}
```

4. ***threads/main.cc*** - if we want to test all types of scheduler, then we need to set extra parameter to choose the type just shown as below

```
$ cd code/threads
$ ./nachos FCFS
$ ./nachos SJF
$ ./nachos Priority
$ ./nachos RR
```

```
int main(int argc, char **argv)
{
    ...
    DEBUG(dbgThread, "Entering main");
    SchedulerType type = RR;
    if(strcmp(argv[1], "FCFS") == 0)
        type = FIFO;
    else if (strcmp(argv[1], "SJF") == 0)
        type = SJF;
    else if (strcmp(argv[1], "PRIORITY") == 0)
        type = Priority;
    else if (strcmp(argv[1], "RR") == 0)
        type = RR;
    kernel = new KernelType(argc, argv);
    ...
}
```

Implement

5. ***machine/machine.h*** - let NumPhysPage bigger to avoid **segmentation fault (core dumped)** which is an error message about memory allocation **(note that this error is hard to debug)**.

```
const unsigned int PageSize = 128;      // set the page size equal to
                                        // the disk sector size, for
simplicity
const unsigned int NumPhysPages = 256;  // Change this line
const int MemorySize = (NumPhysPages * PageSize);
const int TLBSize = 4;                  // if there is a TLB, make it small
```

6. ***threads/scheduler.h*** - define additional scheduler type named FIFO(First-In-First-Out) and get type method in header file

```
enum SchedulerType
{
        RR,     // Round Robin
        SJF,
        Priority,
        FIFO    // Add this line
};
class Scheduler
{
    public:
        ...
        ~Scheduler();
```

```
        Scheduler(SchedulerType type);      // Initialize list of ready
threads
        SchedulerType getSchedulerType() {return schedulerType;}
        void setSchedulerType(SchedulerType t) {schedulerType = t;}
        ...
};
```

7. ***threads/scheduler.cc*** - implement each scheduler type here

```
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;
}
int FIFOCompare(Thread *a, Thread *b)
{
    return 1;
}
//----------------------------------------------------------------------
// Scheduler::Scheduler
//  Initialize the list of ready but not running threads.
//  Initially, no ready threads.
//----------------------------------------------------------------------
Scheduler::Scheduler()
{
    Scheduler(RR);
}
Scheduler::Scheduler(SchedulerType type)
{
    schedulerType = type;
    // readyList = new List<Thread *>;
    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 FIFO:
            readyList = new SortedList<Thread *>(FIFOCompare);
    }
    toBeDestroyed = NULL;
}
```

8. ***threads/alarm.cc*** - if you want to implement ***preemptive***, you need to determine whether call `interrupt->YieldOnReturn()` or not in `Alarm::CallBack()`

   And for the computing burst time, if you decide to use **running time estimation**, you need to compute in `Alarm::WaitUntil()` shown as below

```
void Alarm::CallBack()
{
    ...
    if (status == IdleMode && !woken && _sleepList.IsEmpty())
        ...
    else    // there's someone to preempt
    {
        if(kernel->scheduler->getSchedulerType() == RR || kernel->scheduler-
>getSchedulerType() == Priority )
        {
            cout << "=== interrupt->YieldOnReturn ===" << endl;
            interrupt->YieldOnReturn();
        }
    }
}
void Alarm::WaitUntil(int x)
{
    ...
    Thread* t = kernel->currentThread;

    int worktime = kernel->stats->userTicks - t->getStartTime();
    t->setBurstTime(t->getBurstTime() + worktime);
    t->setStartTime(kernel->stats->userTicks);
    ...
}
```

Revise Some files

You have to include `scheduler.h` to each header file that we'll use that including `kernel.h`, `userkernel.h`, and `netkernel.h`. Then initial the scheduler type in each c file such as `kernel.cc`, `userkernel.cc`, and `netkernel.cc`. Most of them are very similar.

9. ***threads/kernel.h***

```
...
class ThreadedKernel
{
    public:
        ...
        void Initialize();
        void Initialize(SchedulerType type);
        ...
};
```

10. ***threads/kernel.cc***

```
void ThreadedKernel::Initialize()
```

```
{
    Initialize(RR);
}
void ThreadedKernel::Initialize(SchedulerType type)//
{
    stats = new Statistics();       // collect statistics
    interrupt = new Interrupt;      // start up interrupt handling
    scheduler = new Scheduler(type);    // initialize the ready queue
    alarm = new Alarm(randomSlice); // start up time slicing

    // We didn't explicitly allocate the current thread we are running in.
    // But if it ever tries to give up the CPU, we better have a Thread
    // object to save its state.
    currentThread = new Thread("main");
    currentThread->setStatus(RUNNING);

    interrupt->Enable();
}
```

11. ***userprog/userkernel.h***

```
#include "../threads/scheduler.h"

class UserProgKernel : public ThreadedKernel
{
    public:
        ...
        void Initialize();
        void Initialize(SchedulerType type);
        ...
};
```

12. ***userprog/userkernel.cc***

```
void UserProgKernel::Initialize()
{
    Initialize(RR);
}
void UserProgKernel::Initialize(SchedulerType type)//
{
    ThreadedKernel::Initialize(type);   // init multithreading
    machine = new Machine(debugUserProg);
    fileSystem = new FileSystem();
    #ifdef FILESYS
        synchDisk = new SynchDisk("New SynchDisk");
    #endif // FILESYS
}
```

13. ***network/netkernel.h***

```
#include "../threads/scheduler.h"

class NetKernel : public UserProgKernel
{
    public:
        ...
        void Initialize();
        void Initialize(SchedulerType type);
        ...
};
```

14. ***network/netkernel.cc***

```
void NetKernel::Initialize()
{
    Initialize(RR);
}
void NetKernel::Initialize(SchedulerType type)//
{
    UserProgKernel::Initialize(type);   // init other kernel data structs
    postOfficeIn = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
}
```

Result

- Task 1 Result
  I create another test case named Sleep3.c and aim to test the sleep time **10 times longer** than Sleep1.c and Sleep2.c is aim to test the time that **100 times shorter** than Sleep1.c.
  - sleep1

```
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 5; i++) {
        Sleep(1000000);
        PrintInt(2222);
    }
    return 0;
}
```

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/userprog$ ./nachos -e ../test/sleep1
Total threads number is 1
Thread ../test/sleep1 is executing.
Sleep a while:1000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:2222
Sleep a while:1000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:2222
Sleep a while:1000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:2222
Sleep a while:1000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:2222
Sleep a while:1000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:2222
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 500000100, idle 499999828, system 130, user 142
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
```

- Sleep2

```c
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 5; i++) {
        Sleep(10000);
        PrintInt(99999);
    }
    return 0;
}
```

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/userprog$ ./nachos -e ../test/sleep2
Total threads number is 1
Thread ../test/sleep2 is executing.
Sleep a while:10000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:99999
Sleep a while:10000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:99999
Sleep a while:10000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:99999
Sleep a while:10000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:99999
Sleep a while:10000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:99999
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 5000100, idle 4999828, system 130, user 142
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
```

- Sleep3

```c
#include "syscall.h"
main() {
    int i;
    for(i = 0; i < 5; i++) {
        Sleep(10000000);
        PrintInt(666);
    }
    return 0;
}
```

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/userprog$ ./nachos -e ../test/sleep3
Total threads number is 1
Thread ../test/sleep3 is executing.
Sleep a while:10000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:666
Sleep a while:10000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:666
Sleep a while:10000000(ms)
Alarm::WaitUntil go sleep
sleepList::PutToReady Thread woken
Print integer:666
Sleep a while:10000000(ms)
Alarm::WaitUntil go sleep
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2147483643, idle 2147483465, system 90, user 88
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
```

In Sleep1.c, you can feel the sleep function working clearly that compare with a normal code without sleep function or compare with a shorter sleep time such as Sleep2.c

And in Sleep3.c, you can feel the sleeping time much more longer that what we expected but just execute 3 times PrintInt function.**(No idea why)**

- Task 2 Result
  - Result of real multi thread testing

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/userprog$ ./nachos -e ../test/test1 -e ../test/test2
Total threads number is 2
Thread ../test/test1 is executing.
Thread ../test/test2 is executing.
Print integer:9
Print integer:8
Print integer:7
=== interrupt->YieldOnReturn ===
Print integer:20
Print integer:21
Print integer:22
Print integer:23
Print integer:24
=== interrupt->YieldOnReturn ===
Print integer:6
return value:0
Print integer:25
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 300, idle 8, system 70, user 222
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
```

- Result of FCFS

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/threads$ ./nachos 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
A: remaining 2
A: remaining 1
A: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
B: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
D: remaining 2
D: remaining 1
D: remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2700, idle 160, system 2540, 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
```

- Result of RR

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/nachos-4.0/code/threads$ ./nachos 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 ===
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
=== interrupt->YieldOnReturn ===
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
=== interrupt->YieldOnReturn ===
A: remaining 2
A: remaining 1
A: remaining 0
B: remaining 0
D: remaining 2
D: remaining 1
=== interrupt->YieldOnReturn ===
D: remaining 0
```

- Result of SJF

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/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 0 looped 4 times
*** thread 1 looped 4 times
A: remaining 2
A: remaining 1
A: remaining 0
D: remaining 2
D: remaining 1
D: remaining 0
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
B: remaining 2
B: remaining 1
B: remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2600, idle 60, system 2540, 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
```

- Result of priority

```
sbk@sbk-virtual-machine:~/NTU/Operating System/Project2/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
=== interrupt->YieldOnReturn ===
*** thread 1 looped 4 times
*** thread 0 looped 4 times
=== interrupt->YieldOnReturn ===
B: remaining 8
B: remaining 7
B: remaining 6
B: remaining 5
B: remaining 4
B: remaining 3
=== interrupt->YieldOnReturn ===
B: remaining 2
B: remaining 1
B: remaining 0
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
D: remaining 2
D: remaining 1
D: remaining 0
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
C: remaining 6
C: remaining 5
C: remaining 4
C: remaining 3
C: remaining 2
C: remaining 1
C: remaining 0
=== interrupt->YieldOnReturn ===
=== interrupt->YieldOnReturn ===
A: remaining 2
A: remaining 1
A: remaining 0
=== interrupt->YieldOnReturn ===
```