Operating Systems
Project 2 -
System Call & CPU Scheduling

機械所 吳冠甫 R10522532

November 18th, 2022

Contents

Part I: System Call

Motivation

In this part, we have to implement a system call `Sleep()`.

Part I: System Call continued on next page...

Implementation

1. Implement system call

   (a) Define system call number and function prototype for `Sleep()`

```
#define SC_PrintInt   11
#define SC_Sleep      12
```

Listing 1: `/code/userprog/syscall.h`

```
void PrintInt(int number);
void Sleep(int number);
```

Listing 2: `/code/userprog/syscall.h`

   (b) Prepare register for `Sleep()`.

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

Listing 3: `/code/test/start.s`

   (c) Add new case for `Sleep()` in exception

```
switch (which) {
    ...
    case SC_PrintInt:
        val = kernel->machine->ReadRegister(4);
        cout << "Print integer:" << val << endl;
        return;
    case SC_Sleep:
        val = kernel->machine->ReadRegister(4);
        cout << "Sleep time: " << val << "(ms)" << endl;
        kernel->alarm->WaitUntil(val);
        return;
    ...
}
```

Listing 4: `/code/userprog/exception.cc`

- When `Sleep()` is called, `WaitUntil` will be called.
- Thus, we need to implement `Alarm::WaitUntil()`.

Part I: System Call [Implementation] continued on next page…

2. Modify `Alarm`

   (a) Define `SleepThread` for the thread need to be slept by given input time *x*.

```
#include <list>
#include "thread.h"
...
// ---Add
class SleepThread {
public:
    Thread* threadToSleep;
    int duration;

public:
    SleepThread(Thread* t, int x):
        threadToSleep(t), duration(x) {};
};
```

Listing 5: `/code/threads/alarm.h`

(b) Define `SleepPool` for a list structure to save sleeping threads, and take out the ready queue after sleep.

```
// ---Add
class SleepPool {
public:
    int interruptCount;
    std::list<SleepThread> sleepThreadList;

public:
    SleepPool():
        interruptCount(0) {};

    void PutToSleep(Thread* t, int x);

    bool PutReadyQueue();

    bool IsEmpty();
};
```

Listing 6: `/code/threads/alarm.h`

```
// ---Add
void SleepPool::PutToSleep(Thread* t, int x) {
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    sleepThreadList.push_back(SleepThread(t, interruptCount + x));
    t->Sleep(false);
}


// ---Add
bool SleepPool::IsEmpty() {
    return sleepThreadList.size() == 0;
}


// ---Add
bool SleepPool::PutReadyQueue() {
    typedef std::list<SleepThread>::iterator ListIt_t;

    bool woken;
    interruptCount++;

    for (ListIt_t it = sleepThreadList.begin();
         it !=sleepThreadList.end(); ++it) {
        if (interruptCount >= it->duration) {
            woken = true;
            cout << "SleepPool::PutReadyQueue, a thread is taken out" << endl;
            kernel->scheduler->ReadyToRun(it->threadToSleep);
            it = sleepThreadList.erase(it);
        }
    }

    return woken;
}
```

Listing 7: `/code/threads/alarm.cc`

(c) Add member variable `SleepPool` in `Alarm`

```
class Alarm : public CallBackObj {
public:
    ...
private:
    ...
    // ---Add
    SleepPool sleepPool;
    ...
}
```

Listing 8: `/code/threads/alarm.h`

(d) Modify `Alarm::CallBack()`

```
// ---Modify
void Alarm::CallBack() {
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = sleepPool.PutReadyQueue();   // ---Add

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

Listing 9: `/code/threads/alarm.cc`

(e) Add `Alarm::WaitUntil()`

```
// ---Add
void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;

    cout << "Alarm::WaitUntil sleep" << endl;
    sleepPool.PutToSleep(t,x);
    kernel->interrupt->SetLevel(oldLevel);
}
```

Listing 10: `/code/threads/alarm.cc`

3. Test

   (a) Test case I

```c
#include "syscall.h"

main() {
    int i;
    for (i = 0; i < 5; ++i) {
        Sleep(300000);
        PrintInt(111111);
    }
    return 0;
}
```

Listing 11: `/code/test/sleep1.c`

   (b) Test case II

```c
#include "syscall.h"

main() {
    int i;
    for (i = 0; i < 10; ++i) {
        Sleep(100000);
        PrintInt(222222);
    }
    return 0;
}
```

Listing 12: `/code/test/sleep2.c`

Results

1. Execute `sleep1`

```
./nachos -e ../test/sleep1
```

```
Total threads number is 1
Thread ../test/sleep1 is executing.
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 150000100, idle 149999823, system 130, user 147
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
```

Listing 13: Result of `sleep1`

2. Execute `sleep2`

```
./nachos -e ../test/sleep2
```

```
Total threads number is 1
Thread ../test/sleep2 is executing.
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 100000100, idle 99999603, system 230, user 267
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
```

Listing 14: Result of `sleep2`

Part I: System Call [Results] continued on next page. . .

3. Execute `sleep1` and `sleep2` simultaneously.

```
./nachos -e ../test/sleep1 -e ../test/sleep2
```

```
Total threads number is 2
Thread ../test/sleep1 is executing.
Thread ../test/sleep2 is executing.
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
Sleep time: 100000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:222222
return value:0
```

```
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
Sleep time: 300000(ms)
Alarm::WaitUntil sleep
SleepPool::PutReadyQueue, a thread is taken out
Print integer:111111
return value:0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 150000100, idle 149999336, system 350, user 414
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
```

Listing 15: Result of `sleep1` and `sleep2`

Part II: CPU Scheduling

Motivation

In this part, I implement different CPU scheduling algorithm as follows

- First-Come-First-Service (FCFS)
- Shortest-Job-First (SJF)
- Priority

Part II: CPU Scheduling continued on next page. . .

Implementation

1. Test Case

   (a) Create `ThreadInfo()` for printing current running thread and remaining CPU burst time.

```
// ---Add
void
ThreadInfo() {
    Thread* thread = kernel->currentThread;
    while (thread->getCpuBurstTime() > 0) {
        thread->setCpuBurstTime(thread->getCpuBurstTime() - 1);
        kernel->interrupt->OneTick();
        cout << "Running thread "
             << kernel->currentThread->getName()
             << ": cpu burst time remaining "
             << kernel->currentThread->getCpuBurstTime()
             << endl;
    }
}
```

Listing 16: `/code/threads/thread.cc`

   (b) Add `Thread::SchedulingTest()`

```
class Thread {
private:
    ...
    // ---Add
    static void SchedulingTest();
    ...
};
```

Listing 17: `/code/threads/thread.h`

```
// ---Add
void Thread::SchedulingTest() {
    // Test case 1
    const int THREAD_NUM = 4;
    char *name[THREAD_NUM] = {"A", "B", "C", "D"};
    int threadPriority[THREAD_NUM] = {4, 2, 6, 5};
    int cpuBurst[THREAD_NUM] = {5, 8, 7, 1};

    //
    Thread *t;
    for (int i = 0; i < THREAD_NUM; ++i) {
        t = new Thread(name[i]);
        t->setThreadPriority(threadPriority[i]);
        t->setCpuBurstTime(cpuBurst[i]);
        t->Fork((VoidFunctionPtr) ThreadInfo, (void*)NULL);
    }
    kernel->currentThread->Yield();
}
```

Listing 18: `/code/threads/thread.cc`

Part II: CPU Scheduling [Implementation] continued on next page...

2. Add function in class `Thread` for CPU burst time or thread priority.

```
class Thread {
private:
    ...
    // ---Add
    void setCpuBurstTime(int t)    {cpuBurstTime = t;}
    int  getCpuBurstTime()         {return cpuBurstTime;}
    void setThreadStartTime(int t) {threadStartTime = t;}
    int  getThreadStartTime()      {return threadStartTime;}
    void setThreadPriority(int t)  {threadPriority = t;}
    int  getThreadPriority()       {return threadPriority;}
    ...

private:
    ...
    // ---Add
    int cpuBurstTime;
    int threadStartTime;
    int threadPriority;
    ...
};
```

Listing 19: `/code/threads/thread.h`

3. Modify constructor for receiving different `SchedulerType` initializer in

   (a) class `ThreadedKernel`

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

Listing 20: `/code/threads/kernel.h`

```
// ---Modify
void ThreadedKernel::Initialize() {
    Initialize(RR);
}


// ---Add
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   // ---
    Modify
    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();
}


void ThreadedKernel::SelfTest() {
    ...
    // ---Add
    Thread::SchedulingTest();
    ...
}
```

Listing 21: `/code/threads/kernel.cc`

Part II: CPU Scheduling [Implementation] continued on next page...

(b) class `UserProgKernel`

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

Listing 22: `/code/userprog/userkernel.h`

```
void UserProgKernel::Initialize() {
    Initialize(RR);
}
void UserProgKernel::Initialize(SchedulerType type) {
    ThreadedKernel::Initialize(type);   // ---Modify
    ...
}
```

Listing 23: `/code/userprog/userkernel.cc`

(c) class `NetKernel`

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

Listing 24: `/code/network/netkernel.h`

```
// ---Modify
void NetKernel::Initialize() {
    Initialize(RR);
}


// ---Add
void NetKernel::Initialize(SchedulerType type) {
    UserProgKernel::Initialize(type);   // init other kernel data structs

    postOfficeIn  = new PostOfficeInput(10);
    postOfficeOut = new PostOfficeOutput(reliability, 10);
}
```

Listing 25: `/code/network/netkernel.cc`

Part II: CPU Scheduling [Implementation] continued on next page. . .

4. Modify the main function to select the scheduling algorithm in `SchedulerType` through input arguments.

   (a) main function

```
int main(int argc, char **argv) {
    ...
    // ---Add
    SchedulerType type;
    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 {
        type = RR;
    }

    kernel = new KernelType(argc, argv);
    kernel->Initialize(type);   // ---Modify
    ...
}
```

Listing 26: `/code/threads/main.cc`

- Default scheduling algorithm is `RR`

   (b) Add `FIFO` in `SchedulerType`

```
enum SchedulerType {
    RR,      // Round Robin
    SJF,
    Priority,
    FIFO     // ---Add
};
```

Listing 27: `/code/threads/scheduler.h`

- `FIFO` (First-Input-First-Output) for `FCFS`

5. Modify constructor of class `Scheduler` for constructing by `SchedulerType`

```
class Scheduler {
public:
    ...
    Scheduler(SchedulerType type);  // ---Add
    ...
    // ---Add
    SchedulerType getSchedulerType() {return schedulerType;}
    void setSchedulerType(SchedulerType t) {schedulerType = t;}
    ...
};
```

Listing 28: `/code/threads/scheduler.h`

```
// ---Modify
Scheduler::Scheduler() {
    Scheduler(RR);
}


// ---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 FIFO:
        readyList = new SortedList<Thread*>(FIFOCompare);
        break;
    }
    toBeDestroyed = NULL;
}
```

Listing 29: `/code/threads/scheduler.cc`

6. Define compare functions for different `Scheduler`

```
// ---Add
int SJFCompare(Thread* a, Thread* b) {
    if (a->getCpuBurstTime() == b->getCpuBurstTime()) {
        return 0;
    }
    else if (a->getCpuBurstTime() > b->getCpuBurstTime()) {
        return 1;
    }
    else {
        return -1;
    }
}


// ---Add
int PRIORITYCompare(Thread* a, Thread* b) {
    if (a->getThreadPriority() == b->getThreadPriority()) {
        return 0;
    }
    else if (a->getThreadPriority() > b->getThreadPriority()) {
        return 1;
    }
    else {
        return -1;
    }
}


// ---Add
int FIFOCompare(Thread* a, Thread* b) {
    return 1;
}
```

Listing 30: `/code/threads/scheduler.cc`

Part II: CPU Scheduling [Implementation] continued on next page. . .

7. Modify `Alarm::CallBack()` to make `PRIORITY` preemptive.

```
// ---Modify
void Alarm::CallBack() {
    Interrupt *interrupt = kernel->interrupt;
    MachineStatus status = interrupt->getStatus();
    bool woken = sleepPool.PutReadyQueue();   // ---Add

    // ---Modify
    if (status == IdleMode && !woken && sleepPool.IsEmpty()) {  // is it time to quit?
        if (!interrupt->AnyFutureInterrupts()) {
            timer->Disable();   // turn off the timer
        }
    }
    else {   // there's someone to preempt
        if (kernel->scheduler->getSchedulerType() == Priority) {
            cout << "Preemptive scheduling: interrupt->YieldOnReturn" << endl;
            interrupt->YieldOnReturn();
        }
        else if (kernel->scheduler->getSchedulerType() == RR) {
            interrupt->YieldOnReturn();
        }
    }
}


// ---Add
void Alarm::WaitUntil(int x) {
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
    Thread* t = kernel->currentThread;

    // ---Add
    // Count burst time
    int duration = kernel->stats->userTicks - t->getThreadStartTime();
    t->setCpuBurstTime(t->getCpuBurstTime() + duration);
    t->setThreadStartTime(kernel->stats->userTicks);

    cout << "Alarm::WaitUntil sleep" << endl;
    sleepPool.PutToSleep(t, x);
    kernel->interrupt->SetLevel(oldLevel);
}
```

Listing 31: `/code/threads/alarm.cc`

Part II: CPU Scheduling continued on next page. . .

Results

1. Test FCFS

```
./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
Running thread A: cpu burst time remaining 4
Running thread A: cpu burst time remaining 3
Running thread A: cpu burst time remaining 2
Running thread A: cpu burst time remaining 1
Running thread A: cpu burst time remaining 0
Running thread B: cpu burst time remaining 7
Running thread B: cpu burst time remaining 6
Running thread B: cpu burst time remaining 5
Running thread B: cpu burst time remaining 4
Running thread B: cpu burst time remaining 3
Running thread B: cpu burst time remaining 2
Running thread B: cpu burst time remaining 1
Running thread B: cpu burst time remaining 0
Running thread C: cpu burst time remaining 6
Running thread C: cpu burst time remaining 5
Running thread C: cpu burst time remaining 4
Running thread C: cpu burst time remaining 3
Running thread C: cpu burst time remaining 2
Running thread C: cpu burst time remaining 1
Running thread C: cpu burst time remaining 0
Running thread D: cpu burst time remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2700, idle 170, system 2530, 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
```

- Expected result: $A \to B \to C \to D$

2. Test SJF

```
./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
Running thread D: cpu burst time remaining 0
Running thread A: cpu burst time remaining 4
Running thread A: cpu burst time remaining 3
Running thread A: cpu burst time remaining 2
Running thread A: cpu burst time remaining 1
Running thread A: cpu burst time remaining 0
Running thread C: cpu burst time remaining 6
Running thread C: cpu burst time remaining 5
Running thread C: cpu burst time remaining 4
Running thread C: cpu burst time remaining 3
Running thread C: cpu burst time remaining 2
Running thread C: cpu burst time remaining 1
Running thread C: cpu burst time remaining 0
Running thread B: cpu burst time remaining 7
Running thread B: cpu burst time remaining 6
Running thread B: cpu burst time remaining 5
Running thread B: cpu burst time remaining 4
Running thread B: cpu burst time remaining 3
Running thread B: cpu burst time remaining 2
Running thread B: cpu burst time remaining 1
Running thread B: cpu burst time remaining 0
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 2600, idle 70, system 2530, 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
```

- Expected result: $D \rightarrow A \rightarrow C \rightarrow B$

Part II: CPU Scheduling [Results] continued on next page. . .

3. Test `Priority`

```
./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
Preemptive scheduling: interrupt->YieldOnReturn
*** thread 1 looped 4 times
*** thread 0 looped 4 times
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Running thread B: cpu burst time remaining 7
Running thread B: cpu burst time remaining 6
Running thread B: cpu burst time remaining 5
Running thread B: cpu burst time remaining 4
Running thread B: cpu burst time remaining 3
Running thread B: cpu burst time remaining 2
Running thread B: cpu burst time remaining 1
Running thread B: cpu burst time remaining 0
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Running thread A: cpu burst time remaining 4
Running thread A: cpu burst time remaining 3
Running thread A: cpu burst time remaining 2
Running thread A: cpu burst time remaining 1
Running thread A: cpu burst time remaining 0
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Running thread D: cpu burst time remaining 0
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Running thread C: cpu burst time remaining 6
Running thread C: cpu burst time remaining 5
Running thread C: cpu burst time remaining 4
Running thread C: cpu burst time remaining 3
Running thread C: cpu burst time remaining 2
Running thread C: cpu burst time remaining 1
Running thread C: cpu burst time remaining 0
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
Preemptive scheduling: interrupt->YieldOnReturn
No threads ready or runnable, and no pending interrupts.
```

```
Assuming the program completed.
Machine halting!

Ticks: total 2800, idle 130, system 2670, 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
```

- Expected result: $B \to A \to D \to C$