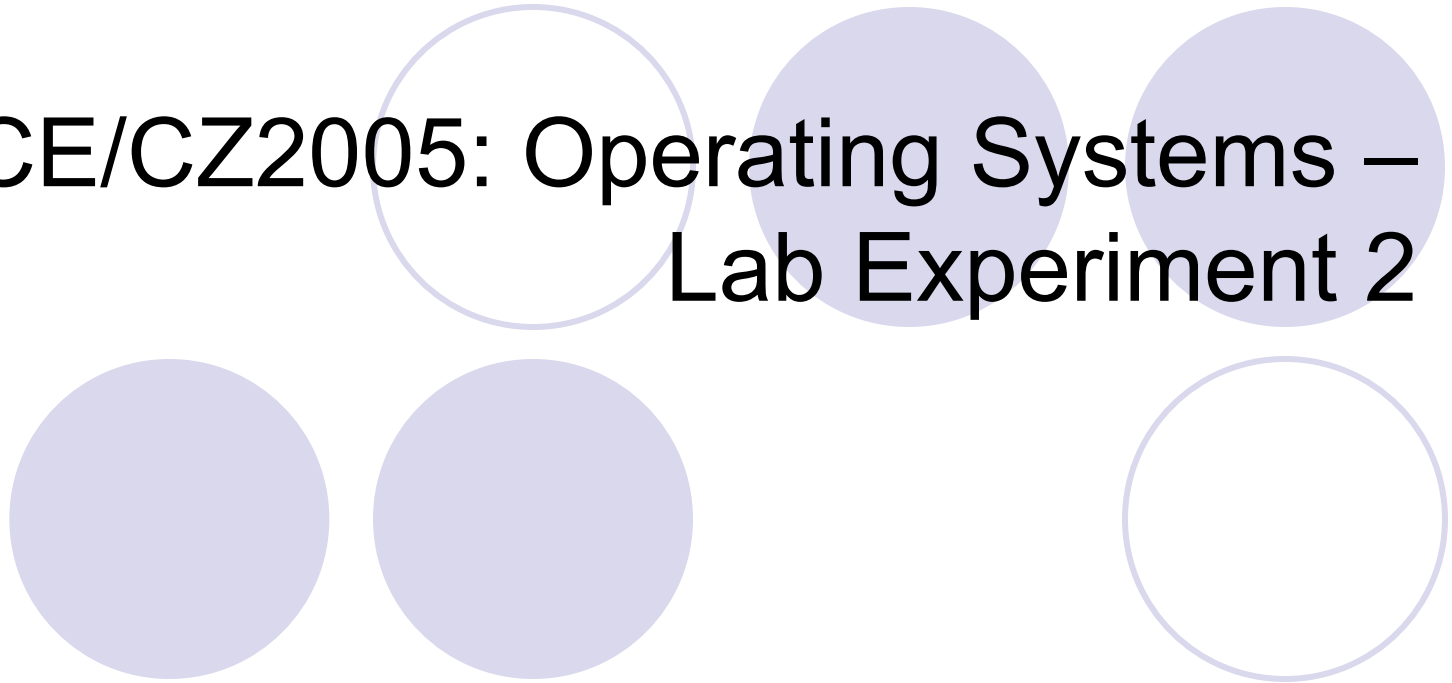
CE/CZ2005: Operating Systems – Lab Experiment 2

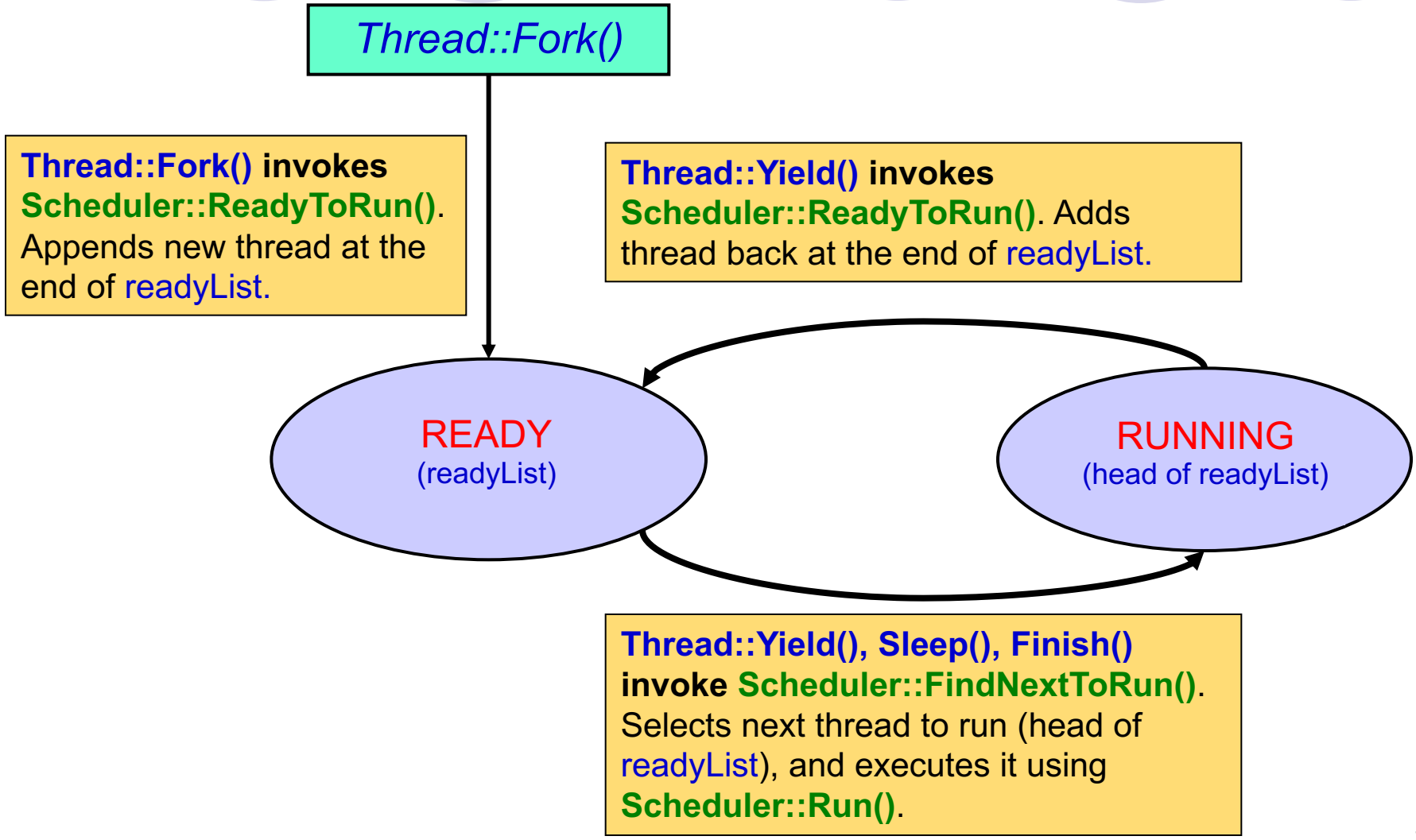


Outline

- CPU Scheduling in NachOS
- Threads, Timers and Interrupts in NachOS
- Discussion of Experiment 2

CPU Scheduling in NachOS

Non-preemptive FIFO Scheduling



Threads

- **Thread()**
  - Constructor: sets the thread as JUST_CREATED status
- **Fork()**
  - Allocate stack, initialize registers.
  - Call Scheduler::ReadyToRun() to put the thread into readyList, and set its status as READY.
- **Yield()**
  - Suspend the calling thread and put it into readyList.
  - Call Scheduler::FindNextToRun() to select another thread from readyList.
  - Execute selected thread by Scheduler::Run(), which sets its status as RUNNING and call SWITCH() (in code/threads/switch.s) to exchange the running thread.
- **Finish()**
  - Mark current thread for destruction.
  - Call **Sleep()** to find next thread to run and execute it.

Threads (Cont.)

- **void Yield()**
  - Suspend the calling thread and select a new one for execution
    - Find next ready thread by calling *Scheduler::FindNextToRun()*.
    - Put current thread into ready list (waiting for rescheduling).
    - Execute the next ready thread by *invoking Scheduler::Run()*.
      - If no other threads are ready to execute, continue running the current thread.

Threads (Cont.)

- **void Finish()**
  - Terminate the currently running thread.
    - Call *Sleep()* and never wake up
    - De-allocate the data structures of a terminated thread
    - The newly scheduled thread examines the *toBeDestroyed* variable and finishes this thread.

The same as “terminated” in our lecture

Threads (Cont.)

- **void Sleep ()**
  - Suspend the current thread and change its state to BLOCKED
    - Run next ready thread
    - Invoke ***interrupt->Idle()*** to wait for the next interrupt when ***readyList*** is empty
  - *Sleep* is called when the current thread needs to be blocked until some future event takes place.
    - Eg. Waiting for a disk read interrupt
    - It is called by Semaphore::P() in code/threads/synch.cc.
    - Semaphore::V() will wake up one of the thread in the waiting queue (sleeping threads queue).

Timers

Timer can be used to trigger an interrupt (i.e., after a fixed number of time ticks)

- **void TimerInterruptHandler ()**
  - Interrupt handler that is called when timer expires.
- **void TimerExpired ()**
  - Function that executes when the timer expires.
- **int TimeOfNextInterrupt ()**
  - Function returns the next interrupt time tick.

Timers (Cont.)

- **void TimerInterruptHandler ()**
  - Function defined in code/threads/system.cc.
  - Executes whenever the associated timer expires and the interrupt is triggered.
  - Timer is initialized in code/threads/system.cc using the constructor for class Timer which is defined in code/machine/timer.cc.
- **void TimerExpired ()**
  - Function defined in code/machine/timer.cc.
  - Executes whenever the timer expires. It in turn invokes the interrupt handler which is defined in previous slide.

Timers (Cont.)

- **int TimeOfNextInterrupt ()**
  - Defined in code/machine/timer.cc.
  - Returns an integer denoting number of time ticks.
  - Used to schedule an interrupt using the timer. The interrupt will be triggered after this number of time ticks from the current time.
  - Can be used to make the timer periodic as required for round-robin scheduling.

Interrupt

- The timer uses several functions from the Interrupt class.
- Pending timer interrupts in the system are maintained in a list called pending, comprising objects of the class PendingInterrupt.
- This list is sorted in increasing order of the time tick when the interrupt will be triggered.
- Defined in code/machine/interrupt.cc.

Interrupt (Cont.)

- **void Schedule()**
  - Function schedules/inserts a new interrupt to the pending list.
  - Insertion is in sorted order; sorted by the pending time ticks for the interrupt to be triggered.
  - Used in Timer to initialize a timer interrupt.

Interrupt (Cont.)

- **void OneTick()**
  - Function to process a single time tick.
  - Updates global variable stats→totalTicks.
  - Calls ***Interrupt::CheckIfDue()*** (defined below) to process any pending interrupt that would be triggered now.
  - If variable yieldOnReturn is true, then triggers a context switch through a call to ***Thread::Yield().***

Interrupt (Cont.)

- **bool CheckIfDue()**
  - Function to process interrupts and invoke handler.
  - Checks if pendingInterrupt at the head of pending list should be triggered at current tick.
  - If yes, corresponding handler is invoked.
  - Handler for Timer is ***Timer::TimerExpired().***

- **void YieldOnReturn()**
  - Function that is called by the Timer handler ***TimerInterruptHandler().***
  - Sets the variable yieldOnReturn to true.
  - Force ***Interrupt::OneTick()*** to trigger context switch.

Timer::Timer()

↓

Interrupt::Schedule(
TimerInterruptHandler,
TimeOfNextInterrupt)

Interrupt::OneTick()

↓

Interrupt::CheckIfDue()

↓

Timer::TimerExpired()

↓

Timer::TimerInterruptHandler()

↓

Interrupt::YieldOnReturn()

↓

Interrupt::OneTick() will trigger context switch
Thread::yield()

Discussion of Experiment 2

Experiment 2 – Overview

- Objective
  - Understand how to schedule processes/threads using round-robin strategy with a fixed time quantum.
  - Understand how to create and reset timer interrupts to implement the fixed time quantum.
- Tasks
  - Initialize the timer interrupt with a fixed time quantum of 40 time ticks.
  - Make the timer interrupt periodic.
  - Reset the timer interrupt if a thread finishes in the middle of a time quantum.
  - Look for code comments /* Experiment 2 */

Directory Structure

bin	For generating NachOS format files, DO NOT CHANGE!
filesys	NachOS kernel related to file system, DO NOT CHANGE!
exp1	Experiment 1, nachos threads.
exp2	Experiment 2, CPU scheduling.
machine	MIPS H/W simulation. Experiment 2 modifications for Timer, Interrupt.
Makefile.common Makefile.dep	For compilation of NachOS, DO NOT CHANGE!
network	NachOS kernel related to network, DO NOT CHANGE!
port	NachOS kernel related to port, DO NOT CHANGE!
readme	Short description of OS labs and assessments
test	NachOS format files for testing virtual memory, DO NOT CHANGE!
threads	NachOS kernel related to thread management. Experiment 2 modifications for System, Thread.
userprog	NachOS kernel related to running user applications, DO NOT CHANGE!

Experiment 2 – User program

- User program for Experiment 2 can be found in `exp2/threadtest.cc`
  - ThreadTest() ← this is the test procedure called from within main()
  - You will use it to evaluate your round-robin implementation. PLEASE DO NOT MODIFY.

Experiment 2 – Summary

- Objective:
  - Understand how to schedule processes/threads using round-robin strategy with a fixed time quantum.
  - Understand how to create and reset timer interrupts to implement the fixed time quantum.
- Assessment:
  - Assessment of your implementation. Please leave your code, the output files **output_1.txt** and **output_2.txt**, as well as **Table1.csv** and **Table2.csv** in the **exp2** folder for TA/Supervisor to review. Deadline is 1 week after your lab session (e.g., if lab session is from 10AM-12PM on a Monday, then deadline is 9:59AM on the next Monday).
  - Lab Quiz 1, which is an online multiple-choice quiz, will be administered through NTULearn during recess week.
- Documents:
  - Can be found in NTULearn

Experiment 2 – Tasks 1 & 2

- Initialize the timer interrupt with a fixed time quantum of 40 time ticks.
  - Activate Timer in code/threads/system.cc.
  - Initialize the timer with the fixed time quantum in code/machine/timer.cc.

- Make the timer interrupt periodic.
  - Modify function Timer::TimerExpired() to make the above timer periodic.
  - It should trigger a timer interrupt every 40 time ticks.

- Test your implementation.
  - Change working directory to Experiment 2 by typing **cd ~/nachos-exp1-2/exp2**.
  - Compile Nachos by typing **make**. If you see **"ln -sf arch/intel-i386-linux/bin/nachos nachos"** at the end of the compiling output, your compilation is successful. If you encounter any anomalies, type **make clean** to remove all object and executable files and then type **make** again for a clean compilation.
  - Trace a run of this Nachos test program by typing **./nachos -d > output_1.txt**. Option **–d** is to display Nachos debugging messages.
  - Populate the table (as instructed in the manual for Experiment 2) based on the generated output.

Task 1 and 2

- threads/system.cc
- machine/timer.cc

Initialize the timer interrupt with a fixed time quantum of 40 time ticks.

1(a) Activate Timer in code/threads/system.cc.

```
//
static void
TimerInterruptHandler(int dummy)
{
    if (interrupt->getStatus() != IdleMode)
        interrupt->YieldOnReturn();
}

//----------------------------------------------------------------------
// Initialize
//      Initialize Nachos global data structures.  Interpret command
//      line arguments in order to determine flags for the initialization.
//
//      "argc" is the number of command line arguments (including the name
//              of the command) -- ex: "nachos -d +" -> argc = 3
//      "argv" is an array of strings, one for each command line argument
//              ex: "nachos -d +" -> argv = {"nachos", "-d", "+"}
//----------------------------------------------------------------------
void
Initialize(int argc, char **argv)
{
  /* Experiment 2 */
  /*Identify where the timer is initialized in this file. Activate the initialization of the timer by updating appropriate
variables.*/

  int argCount;
    char* debugArgs = "";
    bool randomYield = FALSE;
```

```
DebugInit(debugArgs);                   // initialize DEBUG messages
stats = new Statistics();               // collect statistics
interrupt = new Interrupt;              // start up interrupt handling
scheduler = new Scheduler();            // initialize the ready queue
if (randomYield) {                      // start the timer (if needed)
    timer = new Timer(TimerInterruptHandler, 0, randomYield);

    /* Experiment 2*/
    /* Debug message to denote scheduling of timer interrupt*/
    DEBUG('i',"*** Timer interrupt scheduled at %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());
}

threadToBeDestroyed = NULL;

// 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);
```

No change here

- Initialize the timer with the fixed time quantum in code/machine/timer.cc.

Syntax

The syntax for creating a **constant** using #define in the C language is:

```
#define CNAME value
```

```
// timer.cc
//      Routines to emulate a hardware timer device.
//
//      A hardware timer generates a CPU interrupt every X milliseconds.
//      This means it can be used for implementing time-slicing.
//
//      We emulate a hardware timer by scheduling an interrupt to occur
//      every time stats->totalTicks has increased by TimerTicks.
//
//      In order to introduce some randomness into time-slicing, if "doRandom"
//      is set, then the interrupt is comes after a random number of ticks.
//
//      Remember -- nothing in here is part of Nachos.  It is just
//      an emulation for the hardware that Nachos is running on top of.
//
//  DO NOT CHANGE -- part of the machine emulation
//
// Copyright (c) 1992-1993 The Regents of the University of California.
// All rights reserved.  See copyright.h for copyright notice and limitation
// of liability and disclaimer of warranty provisions.

#include "copyright.h"
#include "timer.h"
#include "system.h"

/* Experiment 2 */
// Fixed quantum size definition.
```

```
//----------------------------------------------------------------------
// Timer::TimeOfNextInterrupt
//      Return when the hardware timer device will next cause an interrupt.
//      If randomize is turned on, make it a (pseudo-)random delay.
//----------------------------------------------------------------------

int
Timer::TimeOfNextInterrupt()
{

  /* Experiment 2 */
  /* Update below code so that it returns a fixed time quantum of 40 time ticks */

    if (randomize)
        return 1 + (Random() % (TimerTicks * 2));
    else
        return TimerTicks;
}
```

- 2. Make the timer interrupt periodic.
  - Modify function Timer::TimerExpired() to make the above timer periodic.
  - It should trigger a timer interrupt every 40 time ticks.

machine/timer.cc

```
//----------------------------------------------------------------------
// Timer::TimerExpired
//      Routine to simulate the interrupt generated by the hardware
//      timer device.
//----------------------------------------------------------------------
void
Timer::TimerExpired()
{
    /* Experiment 2 */
    /* Add code below to make the timer periodic. */

    /* Experiment 2*/
       /* Debug message to denote scheduling of timer interrupt*/
    DEBUG('i',"*** Timer interrupt scheduled at %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());

    // invoke the Nachos interrupt handler for this device
    (*handler)(arg);
}

//----------------------------------------------------------------------
```

threads/system.cc

```
DebugInit(debugArgs);                           // initialize DEBUG messages
stats = new Statistics();                       // collect statistics
interrupt = new Interrupt;                      // start up interrupt handling
scheduler = new Scheduler();                    // initialize the ready queue
if (randomYield) {                              // start the timer (if needed)
    timer = new Timer(TimerInterruptHandler, 0, randomYield);

    /* Experiment 2*/
    /* Debug message to denote scheduling of timer interrupt*/
    DEBUG('i',"*** Timer interrupt scheduled at %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());
}

threadToBeDestroyed = NULL;

// 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");
```

```
//----------------------------------------------------------------------
// Timer::Timer
//      Initialize a hardware timer device.  Save the place to call
//      on each interrupt, and then arrange for the timer to start
//      generating interrupts.
//
//      "timerHandler" is the interrupt handler for the timer device.
//              It is called with interrupts disabled every time the
//              the timer expires.
//      "callArg" is the parameter to be passed to the interrupt handler.
//      "doRandom" -- if true, arrange for the interrupts to occur
//              at random, instead of fixed, intervals.
//----------------------------------------------------------------------

Timer::Timer(VoidFunctionPtr timerHandler, _int callArg, bool doRandom)
{
    randomize = doRandom;
    handler = timerHandler;
    arg = callArg;

    // schedule the first interrupt from the timer device
    interrupt->Schedule(TimerHandler, (_int) this, TimeOfNextInterrupt(),
                TimerInt);
}
```

Example of calling interrupt :: Schedule()

output_1.txt

Tick	ready list	current thread	Timer Interrupt triggered	Thread completion	Context switch
40	child1, child2, child3	main	Timer interrupt scheduled at 80		main -> child1
50	child2, child3, main	child1		Thread 1 Completed	child1 -> child2

```
== Tick 10 ==
        interrupts: on -> off
Time: 10, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ed1098)
Interrupt handler timer, scheduled at 40
End of pending interrupts
        interrupts: off -> on
Entering SimpleTestForking thread child1 #0 with func=0x804a640, arg=1, join=NO
        interrupts: on -> off
Putting thread child1 #0 on ready list.
        interrupts: off -> on

== Tick 20 ==
        interrupts: on -> off
Time: 20, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ed1098)
Interrupt handler timer, scheduled at 40
End of pending interrupts
        interrupts: off -> on
Forking thread child2 #0 with func=0x804a640, arg=2, join=NO
        interrupts: on -> off
Putting thread child2 #0 on ready list.
        interrupts: off -> on

== Tick 30 ==
        interrupts: on -> off
Time: 30, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ed1098)
Interrupt handler timer, scheduled at 40
End of pending interrupts
        interrupts: off -> on
Forking thread child3 #0 with func=0x804a640, arg=3, join=NO
        interrupts: on -> off
Putting thread child3 #0 on ready list.
        interrupts: off -> on
```

```
== Tick 40 ==
        interrupts: on -> off
Time: 40, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ed1098)
Interrupt handler timer, scheduled at 40
End of pending interrupts
Invoking interrupt handler for the timer at time 40
Scheduling interrupt handler the timer at time = 80
*** Timer interrupt scheduled at 80
Time: 40, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ee33c0)
Interrupt handler timer, scheduled at 80
End of pending interrupts
        interrupts: off -> on
        interrupts: on -> off
Yielding thread main #0
Putting thread main #0 on ready list.
Switching from thread main #0 to thread child1 #0
        interrupts: off -> on
```

```
== Tick 50 ==
        interrupts: on -> off
Time: 50, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b167(9ee33c0)
Interrupt handler timer, scheduled at 80
End of pending interrupts
        interrupts: off -> on
Thread 1 Completed.
        interrupts: on -> off
Finishing thread child1 #0
Sleeping thread child1 #0
Switching from thread child1 #0 to thread child2 #0
        interrupts: off -> on
```

Experiment 2 – Task 3

- Reset the timer interrupt if a thread finishes in the middle of a time quantum.
  - When the current thread finishes, remove the pending timer interrupt from the pending list, and insert a new timer interrupt with the time quantum of 40 time ticks.
  - You would need to modify files/functions Threads::Finish(), timer.cc, timer.h, interrupt.cc and interrupt.h.
  - Note: For this experiment, to keep things simple, we will assume that no other interrupts are pending in the list, except the timer interrupts created by us.
  - Compile and execute NachOS as in Tasks 1 & 2 in the previous slide (use filename output_2.txt to store your results).
  - Populate the table (as instructed in the manual for Experiment 2) based on the generated output.

Task 3

- machine/timer.cc
- machine/interrupt.cc
- machine/timer.h
- machine/interrupt.h
- threads/thread.cc

Create new function

machine/timer.cc

```
/* Experiment 2 */
//----------------------------------------------------------------------
// Timer::getHandler
//      Return TimerHandler.
//----------------------------------------------------------------------
VoidFunctionPtr
Timer::getHandler()
{

}
```

Create new function

machine/interrupt.cc

```
/* Experiment 2 */
/* Add code to remove pending interrupt from head of pending list. */
//----------------------------------------------------------------------
// RemoveFromPendingQueue
//      Remove the pending interrupt at the head of pending list.
//----------------------------------------------------------------------

void Interrupt::RemoveFromPendingQueue()
{
    Hint: similar syntax as below, but not SortedRemove(). Refers to
    threads/list.cc for replacement.
}
```

machine/interrupt.cc

```
//----------------------------------------------------------------------
bool
Interrupt::CheckIfDue(bool advanceClock)
{
    MachineStatus old = status;
    int when;

    ASSERT(level == IntOff);            // interrupts need to be disabled,
                                        // to invoke an interrupt handler
    if (DebugIsEnabled('i'))
        DumpState();
    PendingInterrupt *toOccur = (PendingInterrupt *)pending->SortedRemove(&when);

    if (toOccur == NULL)                // no pending interrupts
        return FALSE;
```

Header file

Just add the new function to the class public attribute.

machine/timer.h

```
#ifndef TIMER_H
#define TIMER_H

#include "copyright.h"
#include "utility.h"

// The following class defines a hardware timer.
class Timer {
  public:
    Timer(VoidFunctionPtr timerHandler, _int callArg, bool doRandom);
                              // Initialize the timer, to call the interrupt
                              // handler "timerHandler" every time slice.
    ~Timer() {}

// Internal routines to the timer emulation -- DO NOT call these

    void TimerExpired();      // called internally when the hardware
                              // timer generates an interrupt

    int TimeOfNextInterrupt(); // figure out when the timer will generate
                              // its next interrupt

  /* Experiment 2 */
  // get handler for timer to reset it


  private:
    bool randomize;           // set if we need to use a random timeout delay
    VoidFunctionPtr handler;  // timer interrupt handler
    _int arg;                 // argument to pass to interrupt handler

};

#endif // TIMER_H
```

machine/interrupt.h

```
// The following class defines the data structures for the simulation
// of hardware interrupts.  We record whether interrupts are enabled
// or disabled, and any hardware interrupts that are scheduled to occur
// in the future.

class Interrupt {
  public:
    Interrupt();                      // initialize the interrupt simulation
    ~Interrupt();                     // de-allocate data structures

    IntStatus SetLevel(IntStatus level);// Disable or enable interrupts
                                      // and return previous setting.

    void Enable();                    // Enable interrupts.
    IntStatus getLevel() {return level;}// Return whether interrupts
                                      // are enabled or disabled

    void Idle();                      // The ready queue is empty, roll
                                      // simulated time forward until the
                                      // next interrupt

    void Halt();                      // quit and print out stats

    void YieldOnReturn();             // cause a context switch on return
                                      // from an interrupt handler

    MachineStatus getStatus() { return status; } // idle, kernel, user
    void setStatus(MachineStatus st) { status = st; }

    void DumpState();                 // Print interrupt state


    // NOTE: the following are internal to the hardware simulation code.
    // DO NOT call these directly.  I should make them "private",
    // but they need to be public since they are called by the
    // hardware device simulators.

    void Schedule(VoidFunctionPtr handler,// Schedule an interrupt to occur
        _int arg, int when, IntType type);// at time ``when''.  This is called
                                      // by the hardware device simulators.

    void OneTick();                   // Advance simulated time

  /* Experiment 2 */
  // Function to remove pending interrupt from head of pending list.
```

threads/thread.cc

```
void
Thread::Finish ()
{
    if (will_joinP == 0) {              // this thread will not be joined
        (void) interrupt->SetLevel(IntOff);
        ASSERT(this == currentThread);

        DEBUG('t', "Finishing thread %s #%i\n", getName(), pid);

        threadToBeDestroyed = currentThread;

        /* Experiment 2 */
        /* Add code here to reset the timer interrupt so that the next
           interrupt is triggered after 40 time ticks from now.
        */
        // Get handler for the timer.


        // Remove pending timer interrupt from pending list. This is the interrupt based on fixed time quantum.


        // Reschedule a new timer interrupt so that it is triggered after a fixed time quantum from current time.


        // Debug message to denote resetting of timer interrupt
        DEBUG('i',"*** Timer interrupt reset to %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());

        Sleep();                                    // invokes SWITCH
    }
    else {                              // this thread will be joined
        DEBUG('j', "Thread %s #%i is here to revive the thread that "
              "called it\n", getName(), pid);

        join_thereP->P();               // make sure the Join proc has
                                          // been called
        join_wait->V();                 // tell that Join proc that you
                                          // are in finish and done

        (void) interrupt->SetLevel(IntOff);
        ASSERT(this == currentThread);

        DEBUG('t', "Finishing thread %s #%i\n", getName(), pid);

        threadToBeDestroyed = currentThread;

        /* Experiment 2 */
        /* Add code here to reset the timer interrupt so that the next
           interrupt is triggered after 40 time ticks from now.
         */
        // Get handler for the timer.


        // Remove pending timer interrupt from pending list. This is the interrupt based on fixed time quantum.


        // Reschedule a new timer interrupt so that it is triggered after a fixed time quantum from current time.


        // Debug message to denote resetting of timer interrupt
        DEBUG('i',"*** Timer interrupt reset to %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());

        Sleep();                                    // invokes SWITCH
    }
    // not reached
}
```

threads/system.cc

```
DebugInit(debugArgs);                       // initialize DEBUG messages
stats = new Statistics();                   // collect statistics
interrupt = new Interrupt;                  // start up interrupt handling
scheduler = new Scheduler();                // initialize the ready queue
if (randomYield) {                          // start the timer (if needed)
   timer = new Timer(TimerInterruptHandler, 0, randomYield);

   /* Experiment 2*/
   /* Debug message to denote scheduling of timer interrupt*/
   DEBUG('i',"*** Timer interrupt scheduled at %d\n",stats->totalTicks+timer->TimeOfNextInterrupt());
}

threadToBeDestroyed = NULL;

// 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");
```

Output_2.txt

Tick	ready list	current thread	Timer Interrupt triggered	Thread completion	Context switch
40	child1, child2, child3	main	Timer interrupt scheduled at 80		main -> child1
50	child2, child3, main	child1	Timer interrupt reset to 90	Thread 1 Completed	child1 -> child2

```
== Tick 40 ==
        interrupts: on -> off
Time: 40, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b28d(8f49098)
Interrupt handler timer, scheduled at 40
End of pending interrupts
Invoking interrupt handler for the timer at time 40
Scheduling interrupt handler the timer at time = 80
*** Timer interrupt scheduled at 80
Time: 40, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b28d(8f5b3c0)
Interrupt handler timer, scheduled at 80
End of pending interrupts
        interrupts: off -> on
        interrupts: on -> off
Yielding thread main #0
Putting thread main #0 on ready list.
Switching from thread main #0 to thread child1 #0
        interrupts: off -> on

== Tick 50 ==
        interrupts: on -> off
Time: 50, interrupts off
Pending interrupts:
In mapcar, about to invoke 804b28d(8f5b3c0)
Interrupt handler timer, scheduled at 80
End of pending interrupts
        interrupts: off -> on
Thread 1 Completed.
        interrupts: on -> off
Finishing thread child1 #0
Scheduling interrupt handler the timer at time = 90
*** Timer interrupt reset to 90
Sleeping thread child1 #0
Switching from thread child1 #0 to thread child2 #0
        interrupts: off -> on
```

Acknowledgement

- The slides are created with assistance from Ankita Samaddar.