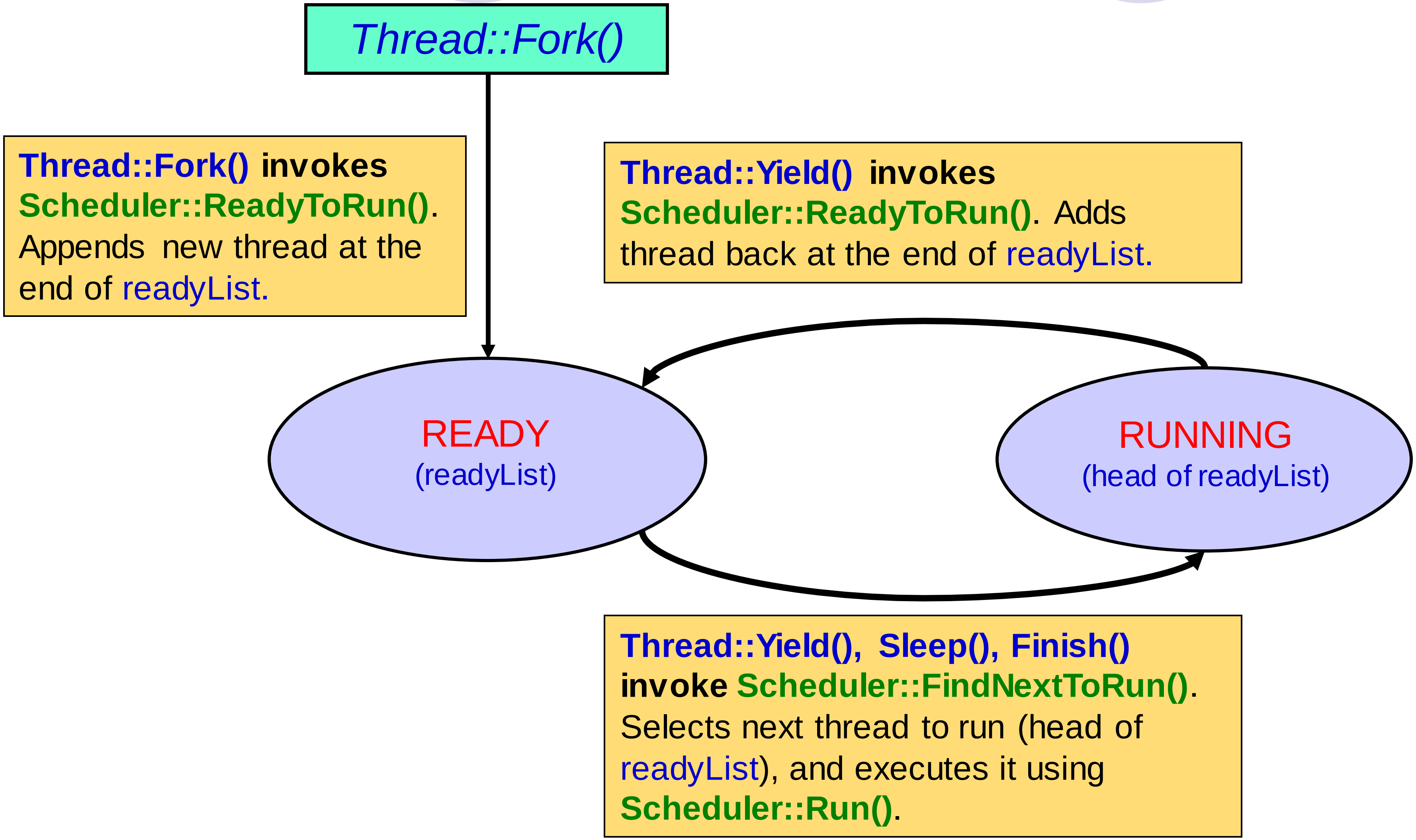
SC2005: Operating Systems – Lab Experiment 2

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.

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 – 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.

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.

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.
- Documents:
  - Can be found in NTULearn

Acknowledgement

- The slides are created with assistance from Ankita Samaddar.