Nachos

- Nachos is an instructional operating system developed at UC Berkeley

- Nachos consists of two main parts:
  - Operating system
    - This is the part of the code that you will study and modify — specifically the code in the threads directory
    - We will not study the code in the userprog, filesys, or network directories
  - Machine emulator — simulates a (slightly old) MIPS CPU, registers, memory, timer (clock), console, disk drive, and network
    - You will study this code, but will not be allowed to modify it
    - This code is in the machine directory

- The OS and machine emulator run together as a single UNIX process

Preparing for the First Project

- Reading assignment:
  - Read about Nachos, & skim the material on the emulated machine and threads
    - Don’t worry about synchronization, user programs, the file system, or networking
  - Read old Appendix A of the text (online as “Overview Paper”)
  - Skim material on threads in Kalra’s “Salsa — An OS Tutorial” (online)
  - Skim Section 2 “Nachos Machine” and Section 3 “Nachos Threads” in Narten’s “A Road Map Through Nachos” (online)
  - Start looking at the code in the threads and machine directories

- If you are not familiar with C++ classes, see the class web page

Preparation for the First Project (cont.)

- Compiling the code
  - Alternative 1:
    - Nachos source code is available in ~walker/pub (read ~walker/pub/README)
    - Get an account on intrepid if you don’t have one (tell the system folks you’re in my Operating Systems class)
      - Note: these instructions work for intrepid, but do not work for the other CS HP Unix machines (trident, aegis)
    - Read “Project 1 — Getting an Early Start” on the class web page to find out how to copy the necessary files to your account, and compile an executable copy of Nachos into the threads directory
  - Alternative 2:
    - Read instructions on Prof. Nesterenko’s web page or Prof. Khan’s web page, and follow their installation instructions to compile Nachos on one of the CS Linux machines (forrestal, iowa)

Nachos — The Emulated Machine

- Code is in the machine directory

  - machine.h, machine.cc — emulates the part of the machine that executes user programs: main memory, processor registers, etc.

  - mipssim.cc — emulates the integer instruction set of a MIPS R2/3000 CPU.

  - interrupt.h, interrupt.cc — manages enabling and disabling interrupts as part of the machine emulation.

  - timer.h, timer.cc — emulates a clock that periodically causes an interrupt to occur.

  - stats.h — collects interesting statistics.
Nachos — The Operating System

- For now, we will mostly be concerned with code in the **threads** directory
- **system.h**, **system.cc** — Nachos startup/shutdown routines.
- **main.cc**, **threadtest.cc** — a simple test of the thread routines.
- **thread.h**, **thread.cc** — thread data structures and thread operations such as thread fork, thread sleep and thread finish.
- **scheduler.h**, **scheduler.cc** — manages the list of threads that are ready to run.
- **list.h**, **list.cc** — generic list management.
- **utility.h**, **utility.cc** — some useful definitions and debugging routines.

Nachos Threads

- As distributed, Nachos does not support multiple processes, only threads
  - All threads share / execute the same code (the Nachos source code)
  - All threads share the same global variables (have to worry about synch.)
- Threads can be in one of 4 states:
  - JUST_CREATED — exists, has not stack, not ready yet
  - READY — on the ready list, ready to run
  - RUNNING — currently running (variable currentThread points to currently running thread)
  - BLOCKED — waiting on some external even, probably should be on some event waiting queue

Scheduling in Nachos

- The Nachos scheduler is non-preemptive FCFS — chooses next process when:
  - Current thread calls Thread::Sleep() (to block (wait) on some event)
  - Current thread calls Thread::Yield() to explicitly yield the CPU
- **main( )** (in threads/main.cc) calls Initialize( ) (in threads/system.cc)
  - which starts scheduler, an instance of class Scheduler (defined in threads/scheduler.h and scheduler.cc)
- Interesting functions:
  - Mechanics of running a thread:
    - Scheduler::ReadyToRun( ) — puts a thread at the tail of the ready queue
    - Scheduler::FindNextToRun( ) — returns thread at the head of the ready queue
    - Scheduler::Run( ) — switches to thread

Scheduling in Nachos (cont.)

```c
Scheduler::Scheduler ( )
{
    readyList = new List;
}

void
Scheduler::ReadyToRun (Thread *thread)
{
    DEBUG("t",
        "Putting thread %s on ready list.\n",
        thread->getName());
    thread->setStatus(READY);
    readyList->Append((void *)thread);
}

Thread *
Scheduler::FindNextToRun ( )
{
    return (Thread *)readyList->Remove();
}
```
Scheduling in Nachos
(cont.)

```c
void
Scheduler::Run (Thread *nextThread)
{
    Thread *oldThread = currentThread;
    oldThread->CheckOverflow();
    currentThread = nextThread;
    currentThread->setStatus(RUNNING);
    DEBUG('t', "Switching from thread "%s" to thread "%s"
          oldThread->getName(),
          nextThread->getName());
    SWITCH(oldThread, nextThread);
    DEBUG('t', "Now in thread "%s"
          currentThread->getName());
    if (threadToBeDestroyed != NULL) {
        delete threadToBeDestroyed;
        threadToBeDestroyed = NULL;
    }
}
```

Working with a Non-Preemptive Scheduler

- The Nachos scheduler is non-preemptive FCFS — chooses next process when:
  - Current thread calls Thread::Sleep( ) (to block (wait) on some event)
  - Current thread calls Thread::Yield( ) to explicitly yield the CPU

- Some interesting functions:
  - Thread::Fork( ) — create a new thread to run a specified function with a single argument, and put it on the ready queue
  - Thread::Yield( ) — if there are other threads waiting to run, suspend this thread and run another
  - Thread::Sleep( ) — this thread is waiting on some event, so suspend it, and hope someone else wakes it up later
  - Thread::Finish( ) — terminate the currently running thread

Manipulating Threads in Nachos

```c
void
Thread::Fork(VoidFunctionPtr func, int arg)
{
    DEBUG('t',"Forking thread "%s" with
           func = 0x%x, arg = %d\n",
           name, (int) func, arg);
    StackAllocate(func, arg);
    IntStatus oldLevel = interrupt->
                        SetLevel(IntOff);
    scheduler->ReadyToRun(this);
    (void) interrupt->SetLevel(oldLevel);
}
```