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
    - This code is in the **threads**, **userprog**, and **filesys** directories
    - We will not study networking, so the **network** directory has been removed
  - 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, or the file system
  - Read old Appendix A of the text (online as “Overview Paper”)
  - Skim Section 2 “Nachos Machine” and Section 3 “Nachos Threads” in Narten’s “A Road Map Through Nachos” (online)
  - Skim material on threads in Kalra’s “Salsa — An OS Tutorial” (online)
  - Start looking at the code in the **threads** and **machine** directories
  - Road Map plus printouts of all code are available in the MCS office for $4.50
- If you are not familiar with C++ or the gdb debugger, see the class web page

Preparing for the First Project (cont.)

- Compiling the code
  - Nachos source code is available in ~walker/pub
  - Read ~walker/pub/README
  - Decide where you want to work, so you can copy files from the appropriate directory into your account
    - ~walker/pub/nachos-3.4-hp
      - For HP workstations (aegis, intrepid)
      - Recommended
    - ~walker/pub/nachos-3.4-sparc
      - For Sun workstations (nimitz)
    - ~walker/pub/nachos-3.4-orig
      - The original, unmodified version
  - 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

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
- **main.cc**, **threadtest.cc** — a simple test of the thread routines.
- **system.h**, **system.cc** — Nachos startup/shutdown 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.)
- 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

```
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);
}
```


Manipulating Threads in Nachos (cont.)

```
void
Thread::Yield ()
{
    Thread *nextThread;

    IntStatus oldLevel = interrupt->
        SetLevel(IntOff);

    ASSERT(this == currentThread);
    DEBUG('t', "Yielding thread \"%s\"\n",
        getName());

    nextThread = scheduler->
        FindNextToRun();
    if (nextThread != NULL) {
        scheduler->ReadyToRun(this);
        scheduler->Run(nextThread);
    }
    (void) interrupt->SetLevel(oldLevel);
}
```

Manipulating Threads in Nachos (cont.)

```
void
Thread::Sleep ()
{
   Thread *nextThread;

   ASSERT(this == currentThread);
   ASSERT(interrupt->getLevel() == IntOff);
   DEBUG('t', "Sleeping thread \"%s\"\n",
      getName());

   status = BLOCKED;
   while ((nextThread = scheduler->
      FindNextToRun()) == NULL)
      interrupt->Idle();

   scheduler->Run(nextThread);
}
```