NachOS Introduction & MP1 Homework Spec

LECTURE: JERRY CHOU

NachOS
Not Another Completely Heuristic Operating System

- **What is NachOS?**
  - Nachos is instructional software for **teaching undergraduate**, and potentially graduate, level operating systems courses.
  - Illustrate and explore all areas of modern operating systems, including threads and concurrency, multiprogramming, system calls, virtual memory, software-loaded TLB's, file systems, network protocols, remote procedure call, and distributed systems.

- **How NachOS works?**
  - written in C++ for MIPS
  - Nachos runs as a user-process on a host operating system
  - A MIPS simulator executes the code for any user programs running on top of the Nachos operating system.

- Website: https://homes.cs.washington.edu/~tom/nachos/

NachOS Architecture
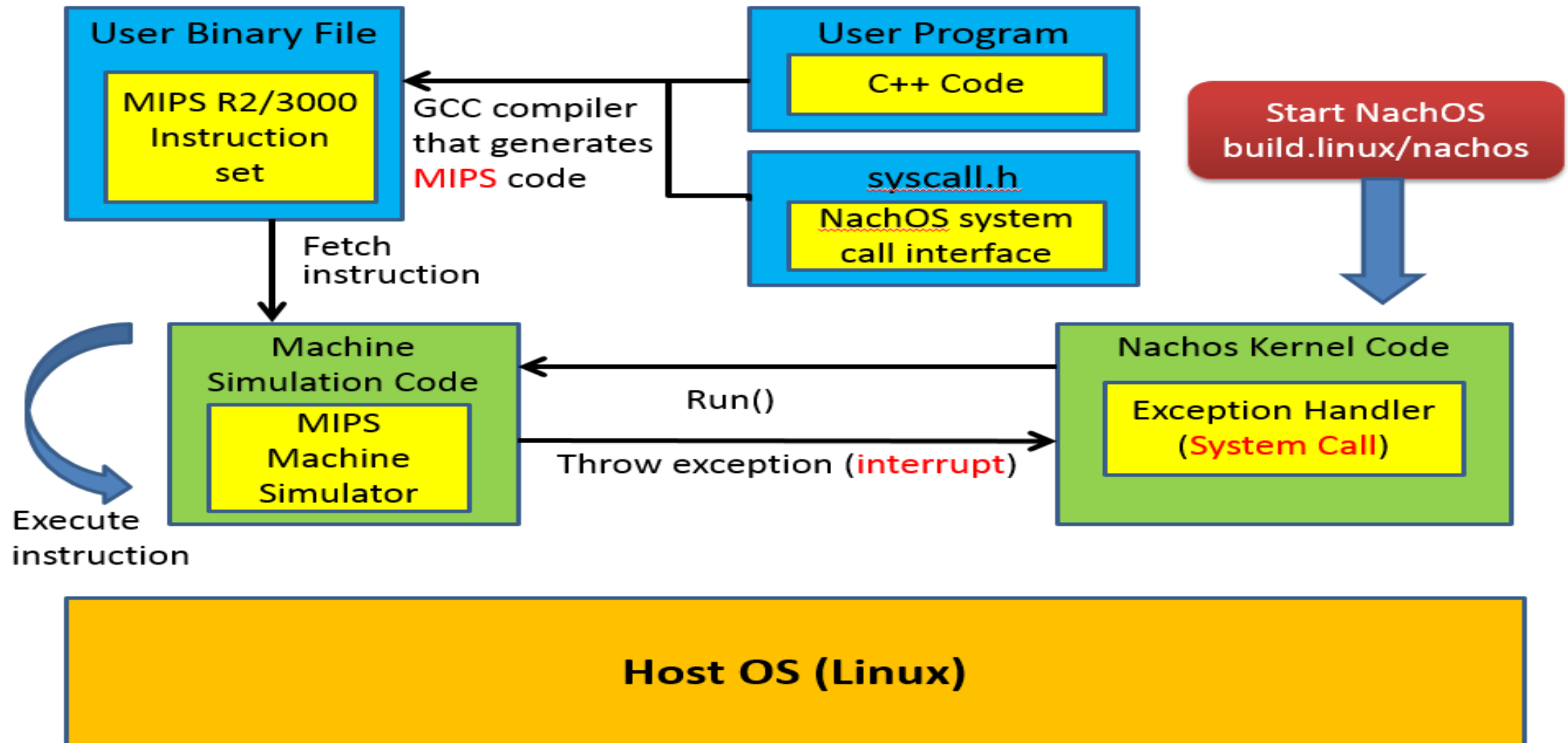
NachOS Directory Structure

lib/
◦ Utilities used by the rest of the Nachos code

machine/
◦ The **machine simulation**.
◦ **All files here CANNOT be modified for any homework assignments**

threads/
◦ **Nachos is a multi-threaded program**. Thread support is found here. This directory also contains the **main() routine of the nachos program in main.cc**.

NachOS Directory Structure

test/

- **User test programs** to run on the simulated machine. This directory contains **its own Makefile**.

- **This is where you can write your own test programs**

userprog/

- **Nachos operating system code** to support the creation of address spaces, loading of user (test) programs, and execution of test programs on the simulated machine.

- You might need to modify the kernel code here

NachOS Directory Structure

network/

◦ Nachos operating system support for networking. Several independent simulated Nachos machines can talk to each other through a simulated network.

◦ We don't need to touch the code in this course.

filesys/

◦ Two different file system implementations are here. The "**real**" file system uses the simulated workstation's simulated disk to hold files. A "**stub**" file system translates Nachos file system calls into UNIX file system calls.

◦ Some files need to be modified in MP1 and MP4

◦ MP1 uses the **stub** file system; MP4 uses the **real** file system

Setup NachOS Environment

- SSH to our server
  - Download putty or MobaXterm
  - Set VPN: https://reurl.cc/NZpGOQ
  - IP address: 10.121.187.197 port:22
  - Username: os21team + your teamID (e.g. os21team01)
  - Password: You are required to reset the password once you login
  - If you have problems, email to os@lsalab.cs.nthu.edu.tw

- Installation (under your home directory)
  $ cp -r /home/os2021/share/NachOS-4.0_MP1 .
  $ cd NachOS-4.0_MP1/code/build.linux
  $ make clean
  $make

Build NachOS kernel

● You must rebuild NachOS every time after you modify anything in NachOS (files under any folder, except test/), otherwise you won't change the execution results.

$ cd NachOS-4.0_MP1/code/build.linux

$ make clean ← If you don't do this, changes to "*.h" files won't be detected during compilation

$ make

Build & Run Test Programs

● You can build any test program under test/ folder to test your NachOS kernel implementation

● Example to build the **halt** test program:

$ cd NachOS-4.0_MP1/code/test

$ make clean

$ make halt

● Example to build the **halt** test program:

$ ../build.linux/nachos -e halt

"-e" means to execute a binary code in NachOS

Makefile

- Make is Unix utility that is designed **to start execution of a makefile**.
- A Makefile is a special file, **containing shell commands**
- Most often, the *makefile* directs *make* on how to compile and link a program.
- How Makefile (test/Makefile) make test programs ?

```
CC = $(GCCDIR)gcc
AS = $(GCCDIR)as
LD = $(GCCDIR)ld

INCDIR =-I../userprog -I../lib
CFLAGS = -G 0 -c $(INCDIR) -B/usr/bin/local/nachos/lib/gcc-lib/decstation-ultrix/2.95.2/ -B/usr/bin/local/nachos/decstation-ultrix/bin/

PROGRAMS = add halt createFile fileIO_test1 fileIO_test2

all: $(PROGRAMS)

start.o: start.S ../userprog/syscall.h
	$(CC) $(CFLAGS) $(ASFLAGS) -c start.S

halt.o: halt.c
	$(CC) $(CFLAGS) -c halt.c
halt: halt.o start.o
	$(LD) $(LDFLAGS) start.o halt.o -o halt.coff
	$(COFF2NOFF) halt.coff halt

clean:
	$(RM) -f *.o *.ii
	$(RM) -f *.coff
```

You may follow the rules for your own new test program

NachOS Debug Message

● NachOS provides different types of debug message that only be printed on screen by the debug message flag in your execution command

```
const char dbgAll = '+';
const char dbgSys = 'u';
const char dbgTraCode = 'c';
```

➢ Type definitions can be seen from "lib/debug.h".

"dbgSys" and "dbgTraCode" can be helpful to you.

➢ Messages type is specified in the code
DEBUG(dbgTraCode, "In ExceptionHandler(), Received Exception " << which << " type: " << type << ", " << kernel->stats->totalTicks);
DEBUG(dbgSys, "Shutdown, initiated by user program.\n");

➢ To show the debug message
$ ../build.linux/nachos -e halt -d u
$ ../build.linux/nachos -e halt -d c

MP1: System Call

- Spec & Deadline: Posted on course website

- Goal:
  - ➢ Understand how to work under **Linux** platform.
  - ➢ Understand how **system calls** are implemented by OS.
  - ➢ Understand the difference between **user mode** and **kernel mode**.

Part1: Trace code

● **Working items**

1. **SC_Halt (halt.c)**

2. **SC_Create (**createFile.c**)**

3. **SC_PrintInt (**add.c**)**

● **Requirements**

➢ Explain the purposes and details of each function call listed in the code path above in **report**.

➢ Explain how the arguments of a system call is passed from user program to kernel in **report**.

userprog/exeception.cc **ExceptionHandler()**

userprog/ksyscall.h **SysPrintInt()**

userprog/synchconsole.cc **SynchConsoleOutput::PutInt()** **SynchConsoleOutput::PutChar()**

machine/console.cc **ConsoleOutput::PutChar()**

machine/interrupt.cc **Interrupt::Schedule()**

machine/mipssim.cc **Machine::Run()**

machine/interrupt.cc **Machine::OneTick()**

machine/interrupt.cc **Interrupt::CheckIfDue()**

machine/console.cc **ConsoleOutput::CallBack()**

userprog/synchconsole.cc **SynchConsoleOutput::CallBack()**

Part2: Implementations

●Working items

1. OpenFileId Open(char *name)
2. int Write(char *buffer, int size, OpenFileId id);
3. int Read(char *buffer, int size, OpenFileId id);
4. int Close(OpenFileId id);

Hint: Files to be modified
➢ test/start.S
➢ userprog/syscall.h, exception.cc, ksyscall.h
➢ filesys/filesys.h

●**Requirements**

➢Must maintain OpenFileTable and use the table entry number of OpenFileTable as the OpenFileId.

➢Must handle invalid file open requests, including the non-existent file, exceeding opened file limit (at most 20 files), etc.

➢All valid file open requests must be accepted if the opened file limit (at most 20 files) is not reached.

➢Must handle invalid file read, write, close requests, including invalid id, etc.

➢……………

Part3: Report

● Working items:

1. Cover page, including team members, Team member contribution

2. Explain how system calls work in NachOS

3. Explain your implementation

Grading

1. Implementation correctness – 50%
   - Pass all the test cases.
   - You DO NOT need to upload NachOS code to iLMS.
   - Your working folder will be locked after deadline.

2. Report – 30%
   - Upload it to iLMS with the Filename: MP1_report_[GroupNumber].pdf.

3. Demo– 20%
   - Answer questions during demo.
   - Demo will take place on our server, so you are responsible to make sure your code works on our server.

*Refer to syllabus for late submission penalty.

Code Trace: userprog/syscall.h

/* syscalls.h

 *   Nachos system call interface. These are Nachos kernel operations

 *   that can be invoked from user programs, by trapping to the kernel

 *   via the "syscall" instruction.

    /* system call codes
    #define SC_Create   4
    #define SC_Remove   5
    #define SC_Open   6
    #define SC_Read   7
    #define SC_Write   8
    #define SC_PrintInt   16

/* The system call interface.
 * **an assembly language stub** stuffs the system call
 * code into a register, and traps to the kernel.

/* Print Integer */
void PrintInt(int number);

/* Return 1 on success, negative error code on failure */
int Create(char *name);

/* Open the Nachos file "name", and return an
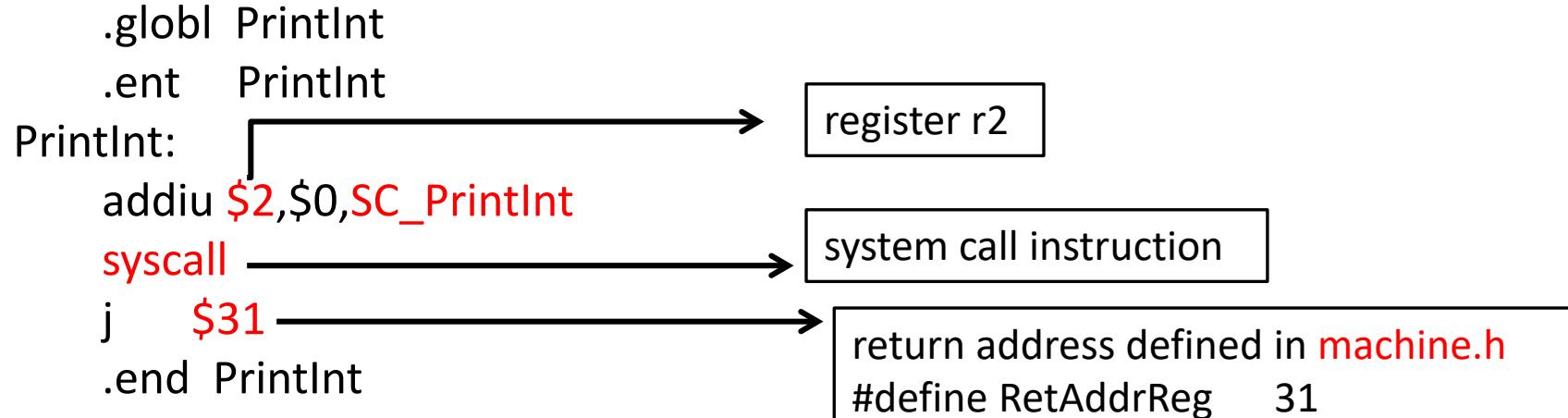 * "OpenFileId" that can be used to R/W the file. */
OpenFileId Open(char *name);

/* A unique identifier for an open Nachos file. */
typedef int OpenFileId;

Code Trace: test/start.S

```
/* System call stubs:
*     Assembly language assist to make system calls to the Nachos kernel.
*     There is one stub per system call, that places the code for the
*     system call into register r2, and leaves the arguments to the
*     system call alone (in other words, arg1 is in r4, arg2 is in r5)
*
*
*     The return value is in r2. This follows the standard C calling
*     convention on the MIPS.
```

```
        .globl  PrintInt
        .ent    PrintInt
PrintInt:
        addiu $2,$0,SC_PrintInt
        syscall
        j       $31
        .end  PrintInt
```

register r2

system call instruction

return address defined in machine.h
#define RetAddrReg   31

ADDIU -- *Add immediate unsigned (no overflow)*

Description:	Adds a register and a sign-extended immediate value and stores the result in a register
Operation:	$t = $s + imm; advance_pc (4);
Syntax:	addiu $t, $s, imm

Code Trace: machine/mipssim.cc

```
//      Simulate the execution of a user-level program on Nachos.
//      Called by the kernel

void Machine::Run()
{
    Instruction *instr = new Instruction;  // storage for decoded instruction

    if (debug->IsEnabled('m')) {
        cout << "Starting program in thread: " << kernel->currentThread->getName();
            cout << ", at time: " << kernel->stats->totalTicks << "\n";
    }
    kernel->interrupt->setStatus(UserMode);
    for (;;) {
        OneInstruction(instr);
        kernel->interrupt->OneTick();
        if (singleStep && (runUntilTime <= kernel->stats->totalTicks))
            Debugger();
    }
}
```

Leave **kernel level program**

Execute **one instruction from user level**

Code Trace: File System Stub

- A "stub" file system translates Nachos file system calls into UNIX file system calls.
- It is enabled by the compiler directive flag "-DFILESYS_STUB"
  - The flag is pre-configured (enabled) in NachOS's makefile (build.linux/Makefile)

```
DEFINES = -DFILESYS_STUB -DRDATA -DSIM_FIX
```

- The flag determines what part of the code will be compiled

```
#ifdef FILESYS_STUB
    //code that will be compiled when FILESYS_STUB is defined
#elseif
    //code that will be compiled when FILESYS_STUB is NOT defined
#endif
```

Code Trace: File System Stub

```cpp
#ifdef FILESYS_STUB                          // Temporarily implement file system calls as
                                             // calls to UNIX, until the real file system
                                             // implementation is available
typedef int OpenFileId;

class FileSystem {
  public:
    FileSystem() {
        for (int i = 0; i < 20; i++) fileDescriptorTable[i] = NULL;
    }

    bool Create(char *name) {
        int fileDescriptor = OpenForWrite(name);

        if (fileDescriptor == -1) return FALSE;
        Close(fileDescriptor);
        return TRUE;
    }
};

#else // FILESYS
class FileSystem {
  public:
    FileSystem(bool format);                 // Initialize the file system.
};

#endif // FILESYS
```

Code Trace: Call Back Function

```
/* machine/interrupt.cc */
void Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type)
{
    int when = kernel->stats->totalTicks + fromNow;
    PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);
    pending->Insert(toOccur);
}
```

Register interrupt callback function in pending queue

```
bool Interrupt::CheckIfDue(bool advanceClock) {
    ……………..
    do {
        next = pending->RemoveFront();
        next->callOnInterrupt->CallBack();
        delete next;
    } while (!pending->IsEmpty() && (pending->Front()->when <= stats->totalTicks));
}
```

Pull interrupt from pending queue

Call interrupt service routine (callback function)

References

- Text editor: vim
  - https://www.radford.edu/~mhtay/CPSC120/VIM_Editor_Commands.htm

- Shell script tutorial
  - https://www.shellscript.sh/

- Linux command
  - Summary of common Unix Commands
  - Common Unix Commands
  - SystemV Commands Pocket Guide

- Makefile
  - Mr. Opengate
  - GUN make: Introduction