113-1 Operating System MP1

Group 45

111062332 朱誼學, 111062333 高英耀

Contributions	朱誼學	高英耀
Trace code	50%	50%
Implement	50%	50%
Report	50%	50%
Explenation	We’ve done all our work in discord vc	

1. Explain how system calls work in NachOS as requested in PartII-1

a. SC_Halt

SC_Halt is called to shut down the whole process.

```
/* Stop Nachos, and print out performance stats */
void Halt();
```

When Nachos starts up, it will call **`void Machine::Run()`**, and this function will never stop. Then, this function will call
`void Machine::OneInstruction()`,
when PCReg point to the address of Halt, SC_Halt will be stored in r2 in start.S and than call **`SYSCALL`**:

```
    .globl Halt
    .ent    Halt
Halt:
    addiu $2,$0,SC_Halt
    syscall
    j   $31
    .end Halt
```

so **`OneInstruction()`** will call **`RaiseException(SyscallException, 0);`**

```
case OP_SYSCALL:
    DEBUG(dbgTraCode, "In Machine::OneInstruction, RaiseException(SyscallException, 0), " << kernel->stats->totalTicks);
    RaiseException(SyscallException, 0);
    return;
```

Then **`RaiseException`** will call **`ExceptionHandler(SyscallException)`**,
in **`ExceptionHandler`**, it will read r2

```
int type = kernel->machine->ReadRegister(2);
```

then execute SC_Halt and call SysHalt()

```
case SC_Halt:
    DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
    SysHalt();
    cout << "in exception\n";
    ASSERTNOTREACHED();
    break;
```

SysHalt() then call **`kernel->interrupt->Halt();`**

```
void SysHalt() {
    kernel->interrupt->Halt();
}
```

Then delete the kernel to shut down nachos.

```
void Interrupt::Halt() {
#ifndef NO_HALT_STAT
    cout << "Machine halting!\n\n";
    cout << "This is halt\n";
    kernel->stats->Print();
#endif
    delete kernel;  // Never returns.
}
```

b. SC_Create

Argument pointer in **`Create()`** will be stored in r4(MIPS conventions),SC_Create will be stored in r2 in start.S and than call **`SYSCALL`**, in **`ExceptionHandler`**, it will read r2 then execute SC_Create, also read r4 to get pointer and find the argument(filename) in memory with the pointer.

```
case SC_Create:
    val = kernel->machine->ReadRegister(4);
    {
        char *filename = &(kernel->machine->mainMemory[val]);
        // cout << filename << endl;
        status = SysCreate(filename);
        kernel->machine->WriteRegister(2, (int)status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

Then passes the filename to **`SysCreate`**,

and **`SysCreate()`** passes to **`FileSystem::Create()`**, then **`FileSystem::Create()`** will Create a file in Disk.

```
int SysCreate(char *filename) {
    // return value
    // 1: success
    // 0: failed
    return kernel->fileSystem->Create(filename);
}
```

c. SC_PrintInt

Argument pointer in **`PrintInt()`** will be stored in r4(MIPS conventions),SC_PrintInt will be stored in r2 in start.S and than call **`SYSCALL,`** in **`ExceptionHandler,`** it will read r2 then executeSC_PrintInt, also read r4 to get pointer and find the argument(int) in memory with the pointer.

```
case SC_PrintInt:
    DEBUG(dbgSys, "Print Int\n");
    val = kernel->machine->ReadRegister(4);
    DEBUG(dbgTraCode, "In ExceptionHandler(), into SysPrintInt, " << kernel->stats->totalTicks);
    SysPrintInt(val);
    DEBUG(dbgTraCode, "In ExceptionHandler(), return from SysPrintInt, " << kernel->stats->totalTicks);
    // Set Program Counter
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

Then passes the int value to **`SysPrintInt`**,

```
void SysPrintInt(int val) {
    DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, into synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
    kernel->synchConsoleOut->PutInt(val);
    DEBUG(dbgTraCode, "In ksyscall.h:SysPrintInt, return from synchConsoleOut->PutInt, " << kernel->stats->totalTicks);
}
```

and **`SysPrintInt()`** passes to **`SynchConsoleOutput::PutInt()`**,this function change the value to string, then passes the chars in the string to **`SynchConsoleOutput::PutChar()`** one by one.

```
void SynchConsoleOutput::PutInt(int value) {
    char str[15];
    int idx = 0;
    // sprintf(str, "%d\n\0", value);  the true one
    sprintf(str, "%d\n\0", value);  // simply for trace code
    lock->Acquire();
    do {
        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into consoleOutput->PutChar, " << kernel->stats->totalTicks);
        consoleOutput->PutChar(str[idx]);
        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return from consoleOutput->PutChar, " << kernel->stats->totalTicks);
        idx++;

        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, into waitFor->P(), " << kernel->stats->totalTicks);
        waitFor->P();
        DEBUG(dbgTraCode, "In SynchConsoleOutput::PutChar, return form  waitFor->P(), " << kernel->stats->totalTicks);
    } while (str[idx] != '\0');
    lock->Release();
}
```

Then **`SynchConsoleOutput::PutChar()`** passes the char to **`ConsoleOutput::PutChar(char ch)`**.

```
void SynchConsoleOutput::PutChar(char ch) {
    lock->Acquire();
    consoleOutput->PutChar(ch);
    waitFor->P();
    lock->Release();
}
```

Then this function checks if hardware is ready for putchar, changes the state of the console to busy, schedules an interrupt and also displays the char.

```
void ConsoleOutput::PutChar(char ch) {
    ASSERT(putBusy == FALSE);
    WriteFile(writeFileNo, &ch, sizeof(char));
    putBusy = TRUE;
    kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);
}
```

`Interrupt::Schedule()` will record the CallBackObject and when to call back in the interrupt, and insert an interrupt to the pending list.

```
void Interrupt::Schedule(CallBackObj *toCall, int fromNow, IntType type) {
    int when = kernel->stats->totalTicks + fromNow;
    PendingInterrupt *toOccur = new PendingInterrupt(toCall, when, type);

    DEBUG(dbgInt, "Scheduling interrupt handler the " << intTypeNames[type] << " at time = " << when);
    ASSERT(fromNow > 0);

    pending->Insert(toOccur);
}
```

After the simulated time in kernel is skipped by the function **`OneTick()`** in **`Run()`**,

```
void Interrupt::OneTick() {
    MachineStatus oldStatus = status;
    Statistics *stats = kernel->stats;

    // advance simulated time
    if (status == SystemMode) {
        stats->totalTicks += SystemTick;
        stats->systemTicks += SystemTick;
    } else {
        stats->totalTicks += UserTick;
        stats->userTicks += UserTick;
    }
    DEBUG(dbgInt, "== Tick " << stats->totalTicks << " ==");
```

`OneTick()` will call **`CheckIfDue()`** to handle interrupts.

```
// check any pending interrupts are now ready to fire
ChangeLevel(IntOn, IntOff);  // first, turn off interrupts
                             // (interrupt handlers run with
                             // interrupts disabled)
CheckIfDue(FALSE);           // check for pending interrupts
ChangeLevel(IntOff, IntOn);  // re-enable interrupts
```

`CheckIfDue()` will call the CallBack function of the CallBackObject, which is **`ConsoleOutput::CallBack()`.**

```
inHandler = TRUE;
do {
    next = pending->RemoveFront();  // pull interrupt off list
    DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, into callOnInterrupt->CallBack, " << stats->totalTicks);
    next->callOnInterrupt->CallBack();  // call the interrupt handler
    DEBUG(dbgTraCode, "In Interrupt::CheckIfDue, return from callOnInterrupt->CallBack, " << stats->totalTicks);
    delete next;
} while (!pending->IsEmpty() && (pending->Front()->when <= stats->totalTicks));
inHandler = FALSE;
return TRUE;
```

`ConsoleOutput::CallBack()` will update the state of hardware and call **`SynchConsoleOutput::CallBack()`**

```
void ConsoleOutput::CallBack() {
    DEBUG(dbgTraCode, "In ConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
    putBusy = FALSE;
    kernel->stats->numConsoleCharsWritten++;
    callWhenDone->CallBack();
}
```

`SynchConsoleOutput::CallBack()` is to tell nachos it is ready to print next char.

```
void SynchConsoleOutput::CallBack() {
    DEBUG(dbgTraCode, "In SynchConsoleOutput::CallBack(), " << kernel->stats->totalTicks);
    waitFor->V();
}
```

2. Explain our implementation as requested in PartII-2

- Working items:

(a). OpenFileId Open(char *name);

Open a file with the name, and return its corresponding OpenFileId.

Return -1 if it fails to open the file.

(b). int Write(char *buffer, int size, OpenFileId id);

Write "size" characters from the buffer into the file, and return the number of characters actually written to the file.

Return -1, if it fails to write the file.

(c). int Read(char *buffer, int size, OpenFileId id);

Read "size" characters from the file to the buffer, and return the number of characters actually read from the file.

Return -1, if it fails to read the file.

(d). int Close(OpenFileId id);

Close the file with id.

Return 1 if successfully close the file. Otherwise, return -1.

Need to delete the OpenFile after you close the file

Explanation:

In **test/start.s**, we add a few assembly code for Open, Write, Read, Close, so that the function define in syscall.h can link to start.S, and then be translated into assembly code.

```
    .globl Open
    .ent    Open
Open:
    addiu $2, $0, SC_Open
    syscall
    j   $31
    .end Open
```

In **userprog/syscall.h**, we uncommented the constant define, which is also the offset for 'addiu' in start.S

```
#define SC_Open 6
#define SC_Read 7
#define SC_Write 8
#define SC_Seek 9
#define SC_Close 10
```

In **userprog/exception.cc**, we add 4 new cases into the switch structure.

SC_Open: Read the address of filename from register 4 and use it to get the filename from mainMemory, then call **`OpenFileID SysOpen(char *name)`**, after that save the returned file id into register2, and update the PC register at the end.

```
case SC_Open:
    val = kernel->machine->ReadRegister(4);
    {
        char *filename = &(kernel->machine->mainMemory[val]);
        fileID = SysOpen(filename);
        kernel->machine->WriteRegister(2, (int)fileID);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

SC_Read: Get the attributes from register 4, 5, 6 respectively. gain the buffer content from mainMemory by address, and pass all the attributes into **`int SysRead(char *buffer, int size, OpenFileId id)`**. The returned status shows how many characters are read into the buffer. Likely, then save the value into register 2, and update the PC register.

```
case SC_Read:
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    {
        char *buf = &(kernel->machine->mainMemory[val]);
        status = SysRead(buf, numChar, fileID);
        kernel->machine->WriteRegister(2, (int)status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

SC_Write: Same as SC_Read but using **`int SysWrite(char *buffer, int size, OpenFileId id)`**. And the status stands for how many characters are written into the buffer instead of read.

```
case SC_Write:
    val = kernel->machine->ReadRegister(4);
    numChar = kernel->machine->ReadRegister(5);
    fileID = kernel->machine->ReadRegister(6);
    {
        char *buf = &(kernel->machine->mainMemory[val]);
        status = SysWrite(buf, numChar, fileID);
        kernel->machine->WriteRegister(2, (int)status);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

SC_Close: Read the file id from register 4 and save into val, passing it into int SysClose(OpenFileId id), and get the returned value called success. Later, save the result into register 2. At last, update the PC register as well.

```
case SC_Close:
    val = kernel->machine->ReadRegister(4);
    {
        int success;
        success = SysClose(val);
        kernel->machine->WriteRegister(2, (int)success);
    }
    kernel->machine->WriteRegister(PrevPCReg, kernel->machine->ReadRegister(PCReg));
    kernel->machine->WriteRegister(PCReg, kernel->machine->ReadRegister(PCReg) + 4);
    kernel->machine->WriteRegister(NextPCReg, kernel->machine->ReadRegister(PCReg) + 4);
    return;
    ASSERTNOTREACHED();
    break;
```

In **userprog/ksyscall.h**:

Define the functions that we called in exception.cc, which simply pass the return value from kernel->fileSystem to exceptionHandler.

```
OpenFileId SysOpen(char *name)
{
        return kernel->fileSystem->OpenAFile(name);
}

int SysRead(char *buffer, int size, OpenFileId id){
    return kernel->fileSystem->ReadFile(buffer, size, id);
}

int SysWrite(char *buffer, int size, OpenFileId id){
    return kernel->fileSystem->WriteFile(buffer, size, id);
}

int SysClose(OpenFileId id){
    return kernel->fileSystem->CloseFile(id);
}
```

In **userprog/filesys.h**:

We have to define a 2d char array in order to store the opened file's name.

```
// An array to memorize the name of opened file.
char OpenFileName[20][100];
```

In **`OpenAFile`**, at first we will call a predefined function **`inline OpenFile *FileSystem::Open(char *name)`** which will return an OpenFile instance. If it fails to open a file, **`OpenAFile`** will return -1. Moreover, we adopt a brute force search for handling duplicate file opening. If it matches the same filename, return -1. Next, we parse over the OpenFileTable, finding an index with nothing is in the corresponding memory space. When found, put the OpenFile instance in, and memorize the filename. Then return the index.

```
OpenFileId OpenAFile(char *name) {
    OpenFile* f = Open(name);
    if(f == NULL) return -1;
    for(int i=0; i<20; i++)
        if(strcmp(OpenFileName[i], name) == 0) return -1;
    int ans = -1;
    for(int i=0; i<20; i++)
        if(OpenFileTable[i] == NULL){
            OpenFileTable[i] = f;
            strcpy(OpenFileName[i], name);
            ans = i;
            break;
        }
    return (OpenFileId)ans;
}
```

In **`WriteFile`**, we get the corresponding file with file id as the index. If it fails to open the file or id is out of range, the function returns -1, else calls the predefined function **`inline int OpenFile::Write(char *from, int numBytes)`**, this function returns the number of written characters.

```
int WriteFile(char *buffer, int size, OpenFileId id){
    OpenFile* f = OpenFileTable[id];
    if(f != NULL && id >= 0 && id < 20){
        return f->Write(buffer, size);
    }
    else return -1;
}
```

`ReadFile` is similar to WriteFile but called **`inline int OpenFile::Read(char *into, int numBytes)`** instead of **`f->Write()`**.

```
int ReadFile(char *buffer, int size, OpenFileId id){
    OpenFile* f = OpenFileTable[id];
    if(f != NULL && id >= 0 && id < 20){
        return f->Read(buffer, size);
    }
    else return -1;
}
```

In **`CloseFile`**, we get the file we're going to close by its file id. Also, check whether we get the OpenFile pointer successfully, and the id is also in [0, 20). Then, clear the corresponding row in the name array. Delete the file by its pointer and return 1.

```
int CloseFile(OpenFileId id){
    OpenFile* f = OpenFileTable[id];
    if(f != NULL && id >= 0 && id < 20){
        OpenFileTable[id] = NULL;
        for(int i=0; i<100; i++) OpenFileName[id][i] = '\0';
        delete f;
        return 1;
    }
    else return -1;
}
```

3. The difficulties we encounter when implementing this assignment:

高英耀: At the very beginning, I did not know where to start, and I was afraid that changing something would cause the crush of nachos. So I asked Chatgpt for an example, and got an answer which seems to be very reliable. Anyway, now I have implemented some functions, so I should be less afraid to do some implementations in future, even though no one told me how to do it.

朱誼學: When I am tracing the codes, I am very confused about why, there are many function such as halt() didn’t be defined but still callable. After reading the problems and responses in the discussion list in eeclass, I have totally understood how start.S and the files in userprog interact with each other. Moreover, there isn't any TODO labeled for us, so I took a little bit more time to figured out where should I start my implementation.

4. Feedback:

高英耀: 老師教得很好，上課的內容都能使我輕易的理解與吸收，認真上課使我在trace code時能順利理解，也感謝討論區的同學與李秉綸助教在討論區的提問語回答，在實作的過程遇到的問題大多都能在討論區找到答案，李秉綸助教的回答都十分熱情與迅速，值得五星好評，實在感謝這堂OS的教授與助教們，為我們清華CS的學生建造出一個十分良好的學習環境。

朱誼學: Thanks for my classmates and TAs asking and answering question in eeclass, so that I can find the solution of most of my problems in there. And I really learn a lot in the professor's lesson and in this lab. Combining the detailed teaching and practical implementation, I feel like the Operating system is not that abstract anymore.