**OS HW01 GROUP 18**

**Part 1:Trace Code Result**

1. **Flow Chart of Halt() System Call:**

**machine.h**

1. The GCC compiler compiles the user program into MIPS code, which is then executed by the MIPS Machine Simulator as it fetches instructions from the User Binary File.
2. This header file defines the core data structures and functions necessary for simulating the execution of user programs in Nachos, including the management of user memory, CPU registers, and exceptions. It provides the tools for emulating a simplified MIPS architecture, enabling virtual-to-physical address translation and handling exceptions during program execution.

**start.S**

When *Halt()* is called, the assembly case specifies that a system call must be made to the NachOS kernel.

Compile

**User Program**

# include <syscall.h>

Halt();

Compile

**syscall.h**

# define SC\_Halt 0;

Void Halt();

**mipssim.cc/machine.cc**

1. In the *Machine::Run()* for loop, the *Oneinstruction()* function is invoked to execute instructions from a user-level program.
2. In the *Oneinstruction()* switch statement, the `case OP\_SYSCALL` handles the scenario when a system call is invoked.
3. In the “case OP\_SYSCALL”, *RaiseException()* in machine.cc is executed. It stores the virtual address badVaddr into the special register BadVAddrReg, allowing the exception handler to identify the address where the problem occurred.
4. Prior to handling the exception, the system ensures that any pending or delayed operations are fully completed
5. The system switches from user mode to kernel mode, then passes the *SyscallException()* to the *ExceptionHandler()* function.
6. After *ExceptionHandler()* completes execution, the system switches from kernel mode back to user mode.

**userprog/ exception.cc**

1. *ExceptionHandler()* retrieves the system call type from the register and executes the *SC\_Halt()*
2. The specific handling of *SC\_Halt()* is performed in *SysHalt()*.
3. *ASSERTNOTREACHED()* ensures the program does not continue, as the simulator should have already been halted.

**ksyscall.h**

*SysHalt()* calls the *Halt()* function to complete the operation

**machine.h/interrupt.cc**

The system enters the *Halt()* function in interrupt.cc, which terminates the kernel and stops the program.

1. **Details of Trace Halt() Code**

0.text/Start.S

.globl \_\_start

.ent \_\_start

\_\_start:

jal main

move $4,$0

jal Exit /\* if we return from main, exit(0) \*/

.end \_\_start

1.syscall.h

viod Halt():

2.text/halt.c

int

main()

{

    Halt();

    /\* not reached \*/

}

3.text/Start.s

ccccccccccfd

.globl Halt

.ent Halt

Halt:

addiu $2,$0,SC\_Halt

syscall

j $31

.end Halt

.globl MSG

.ent MSG

4.machin.h

viod Run();

5.mipssim.cc

Machine::Run();

for (;;) {

OneInstruction(instr);

kernel->interrupt->OneTick();

if (singleStep && (runUntilTime <= kernel->stats>totalTicks)) Debugger();

}

6. mipssim.cc

viod Machine::OneInstruction(Instruction\*instr)

if (!ReadMem(registers[PCReg], 4, &raw))

return; // exception occurred

instr->value = raw;

instr->Decode();

int pcAfter = registers[NextPCReg] + 4;

int sum, diff, tmp, value;

unsigned int rs, rt, imm;

switch (instr->opCode) {

case OP\_ADD:

sum = registers[instr->rs] + registers[instr->rt];

if (!((registers[instr->rs] ^ registers[instr->rt]) & SIGN\_BIT) && ((registers[instr->rs] ^ sum) & SIGN\_BIT)) {

RaiseException(OverflowException, 0);

return;

}

registers[instr->rd] = sum;

break;

case OP\_SYSCALL:

RaiseException(SyscallException, 0);

return;

7. machine.cc

void Machine::RaiseException(ExceptionType which, int badVAddr)

registers[BadVAddrReg] = badVAddr;

DelayedLoad(0, 0); // finish anything in progress

kernel->interrupt->setStatus(SystemMode);

ExceptionHandler(which); // interrupts are enabled at this point

kernel->interrupt->setStatus(UserMode);

8.userprog/exception.cc

void ExceptionHandler(ExceptionType which)

switch (which) {

case SyscallException:

switch(type) {

case SC\_Halt:

DEBUG(dbgSys, "Shutdown, initiated by user program.\n");

SysHalt();

cout<<"in exception\n";

ASSERTNOTREACHED();

break;

}

9.ksyscall.h

void SysHalt() {

kernel->interrupt->Halt();

}

10.interrupt.cc

void Interrupt::Halt() {

cout << "Machine halting!\n\n";

cout << "This is halt\n"; kernel->stats->Print();

delete kernel; // Never returns. }

1. **Flow Chart of Create() System Call:**

**start.S**

When *Create ()* is called, the assembly case specifies that a system call must be made to the NachOS kernel.

**syscall.h**

# define SC\_Halt 0;

int Create(char \*name);

**User Program**

# include <syscall.h>

Create(“filename”);

Compile

Compile

**machine.h**

1. The GCC compiler compiles the user program into MIPS code, which is then executed by the MIPS Machine Simulator as it fetches instructions from the User Binary File.
2. This header file defines the core data structures and functions necessary for simulating the execution of user programs in Nachos, including the management of user memory, CPU registers, and exceptions. It provides the tools for emulating a simplified MIPS architecture, enabling virtual-to-physical address translation and handling exceptions during program execution.

**mipssim.cc / machine.cc**

1. In the *Machine::Run()* for loop, the *Oneinstruction()* function is invoked to execute instructions from a user-level program.
2. In the *Oneinstruction()* switch statement, it would fetch and decode the instruction. According to *Create*() defined in start.S, it would then perform the case “OP\_ADDIU ” , `OP\_SYSCALL` and ‘OP\_J’. In case “OP\_ADDIU”, it would load the value SC\_Create. In case “OP\_SYSCALL”, handles the scenario when a system call is invoked. ‘OP\_J’ would jump back the PC.
3. In the “case OP\_SYSCALL”, *RaiseException()* in machine.cc is executed. It stores the virtual address badVaddr into the special register BadVAddrReg, allowing the exception handler to identify the address where the problem occurred.
4. Prior to handling the exception, the system ensures that any pending or delayed operations are fully completed
5. The system switches from user mode to kernel mode, then passes the *SyscallException()* to the *ExceptionHandler()* function.
6. After *ExceptionHandler()* completes execution, the system switches from kernel mode back to user mode.

**exception.cc / machine.cc**

1. *ExceptionHandler()* deals with the system call type from the register and executes the switch case *SC\_Create*
2. In *SC\_Create* case, it would get string address through function *machine*::*ReadRegister()* and load the filename from main memory
3. The specific handling of *SC\_Create* is performed in *SysCreate ()*.
4. *ASSERTNOTREACHED()* ensures the program does not continue, as the simulator should have already been halted.

**ksyscall.h / kernel.cc**

1. *SysCreate()* calls the *CreateFile()* function in kernel.cc.
2. *CreateFile*() calls *Create*() defined in filesys.cc to complete the operation

**filesys.cc**

1. *FileSystem::Create()* would first instance directory object. And use the member function *Find()* to check is there exist the same file in this directory.
2. Use the *FindAndSet*()function from object freeMap to check if there is a sector to hold the file header.
3. use the member function *Find()* from object directory to save the filename in this directory.
4. Use the *Allocate*() function in object hdr to allocate disk for data
5. If all function mention before succeeds, it would write back all change back to disk
6. Lastly, delete all object.
7. **Details of Trace Create() Code**

0. text/start.S

.globl \_\_start

.ent \_\_start

\_\_start:

jal main

move $4,$0

jal Exit /\* if we return from main, exit(0) \*/

.end \_\_start

1. syscall.h

int Create(char \*name);

2. text/start.S

.globl Create

.ent Create

Create:

addiu $2,$0,SC\_Create

syscall

j $31

.end Create

3. machine.h

void Run();

4.mipssim.cc

void Machine::Run()

for (;;) {

OneInstruction(instr);

kernel->interrupt->OneTick();

if (singleStep && (runUntilTime <= kernel->stats->totalTicks))

Debugger();

}

5.mipssim.cc

void Machine::OneInstruction(Instruction \*instr)

if (!ReadMem(registers[PCReg], 4, &raw))

return; // exception occurred

instr->value = raw;

instr->Decode();

switch (instr->opCode) {

case OP\_ADDIU:

registers[instr->rt] = registers[instr->rs] + instr->extra;

break;

case OP\_J:

pcAfter = (pcAfter & 0xf0000000) | IndexToAddr(instr->extra);

break;

case OP\_SYSCALL:

RaiseException(SyscallException, 0);

return;

6.machine.cc

Void Machine::RaiseException(ExceptionType which, int badVAddr)

DEBUG(dbgMach, "Exception: " << exceptionNames[which]);

registers[BadVAddrReg] = badVAddr;

DelayedLoad(0, 0); // finish anything in progress

kernel->interrupt->setStatus(SystemMode);

ExceptionHandler(which); // interrupts are enabled at this point

kernel->interrupt->setStatus(UserMode);

7.exception.cc

void ExceptionHandler(ExceptionType which)

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

int val;

int status, exit, threadID, programID;

DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");

switch (which) {

case SyscallException:

switch(type) {

case SC\_Create:

val = kernel->machine->ReadRegister(4);

{

char \*filename = &(kernel->machine->mainMemory[val]);

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;

8. ksyscall.h

int SysCreate(char \*filename)

return kernel->interrupt->CreateFile(filename);

9. kernel.cc

int Kernel::CreateFile(char \*filename)

return fileSystem->Create(filename);

10. filesys.cc

bool FileSystem::Create(char \*name, int initialSize)

Directory \*directory;

PersistentBitmap \*freeMap;

FileHeader \*hdr;

int sector;

bool success;

DEBUG(dbgFile, "Creating file " << name << " size " << initialSize);

directory = new Directory(NumDirEntries);

directory->FetchFrom(directoryFile);

if (directory->Find(name) != -1)

success = FALSE;

else {

freeMap = new PersistentBitmap(freeMapFile,NumSectors);

sector = freeMap->FindAndSet();

if (sector == -1)

success = FALSE;

else if (!directory->Add(name, sector))

success = FALSE;

else {

hdr = new FileHeader;

if (!hdr->Allocate(freeMap, initialSize))

success = FALSE;

else {

success = TRUE;

hdr->WriteBack(sector);

directory->WriteBack(directoryFile);

freeMap->WriteBack(freeMapFile);

}

delete hdr;

}

delete freeMap;

}

delete directory;

return success;

1. **Details of Makefile**

1.text/Makefile

include Makefile.dep //

CC = $(GCCDIR)gcc //

AS = $(GCCDIR)as

LD = $(GCCDIR)ld

INCDIR =-I../userprog -I../lib

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

ifeq ($(hosttype),unknown)

PROGRAMS = unknownhost

else

PROGRAMS = halt

endif

Makefile.dep 是判斷執行的作業系統環境。CC , AS ,LD , INCDIR , CFLAGS是類似define指令，後面程式假如為這些被定義過字則會被會替代掉。

CC定義了Compile是使用gcc。

AS 定義了Assembler是使用as。

INCDIR(Include directory) 定義了 Compiler要找的directory。

CFLAGS(Complier Flags) 定義了其他complier的參數，例如: -G(用來控制資料分配), -c(只進行編譯然後不把.o檔連結再一起)

根據makefile.dep 定義的hosttype決定要執行的target，如果hosttype 成立那只會執行 unknownhost 這個target，不成立的話會執行PROGRAM指定的target(因為target all的dependence為$(PROGRAMS))。

在terminal 端打上 make會預設執行all這個target，其餘的function為target 與 dependency的關係，例如:以halt這個target 為例，halt.o 跟 start.o 是 halt 的dependency，然後halt.c又是halt.o 的 dependency，再加上start.S 和../userprog/syscall.h 是 start.o的dependency。所以只要有任一的dependency改變，他們所對應的target就會執行。

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

add.o: add.c

    $(CC) $(CFLAGS) -c add.c

add: add.o start.o

    $(LD) $(LDFLAGS) start.o add.o -o add.coff

    $(COFF2NOFF) add.coff add

shell.o: shell.c

    $(CC) $(CFLAGS) -c shell.c

shell: shell.o start.o

以halt target 說明complier的過程，所以halt 的$(LD) $(LDFLAGS)是最前面提到的替代字元會執行linker的動作，然後COFF2NOFF是定義Makefile.dep的替代字元，之後將.coff檔轉成halt 執行檔。

然後其餘的target 都是一樣執行上述說明的動作。（故沒有貼出所有makefile檔）

**Part 2:Implement System Call**

1. **Detail of your Console I/O system call implementation**

**#void PrintInt(int number)**

1. exception.cc

void ExceptionHandler( ExceptionType which )

case SC\_PrintInt:

val = kernel->machine->ReadRegister(4);

SysPrintInt(val);

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;

break;

在 ExceptionHandler() function中額外新增了 SC\_PrintInt case，並呼叫我們額外定義的 SysPrintInt() function．

2. ksyscall.h

void SysPrintInt(int val)

void SysPrintInt(int val){

kernel->synchConsoleOut->PutInt(val);

}

我們call另一個我們新定義的function PutInt fuction

3. synchconsole.cc

void SynchConsoleOutput::PutInt(int val)

void SynchConsoleOutput::PutInt(int val){

char str[20];

int idx = 0;

sprintf(str, "%d\n\0", val);

lock->Acquire();

consoleOutput->PutString(str);

waitFor->P();

lock->Release();

}

put char 新增成 put string

我們參考putchar()的形式去重新定義PutInt

然後我們將input int val包含換行 轉換成 string。

另外我們在呼叫另一個自己定義的Put-string function, 來input date string。

此PutInt function 是參考原本內建的SynchConsoleOutput::PutChar function 進行修改．

首先我們將input int val包含換行 轉換成 string。

另外我們在呼叫另一個自己定義的Putstring function, 來input date string。

其餘的操作都同 PutChar() function．

void ConsoleOutput::PutString(char \* ch){

int idx = 0;

do{

WriteFile(1, &ch[idx], sizeof(char));

idx++;

}while(ch[idx] !='\0');

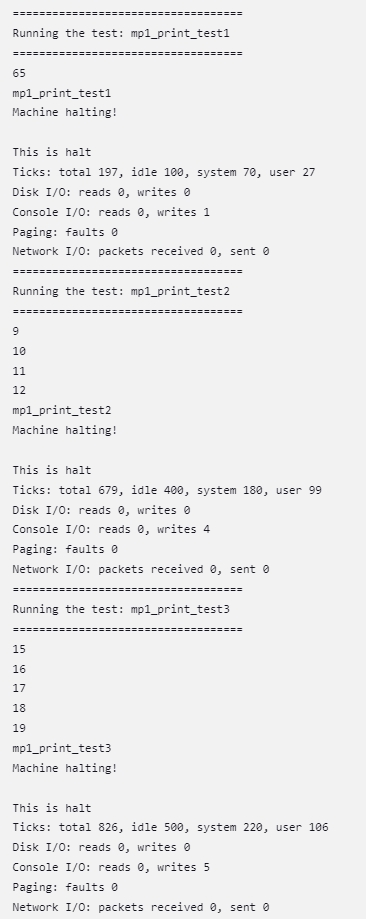
putBusy = TRUE; //

kernel->interrupt->Schedule(this, ConsoleTime, ConsoleWriteInt);

}

此 PutString function 也是參考 ConsoleOutput::PutChar function．只是在這邊我們直接使用 writeFile 印出整個string，之後也進同前做 schedule() 的動作．

**\*\*The final test for the PrintInt()\*\***



1. **Detail of your File I/O system call implementation**

Detail of File I/O system call

1. exception.cc

void ExceptionHandler(ExceptionType which){

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

int val;

int status, exit, threadID, programID;

DEBUG(dbgSys, "Received Exception " << which << " type: " << type << "\n");

switch (which) {

case SyscallException:

switch(type) {

case SC\_Halt:

case SC\_MSG: // case : 100

case SC\_Create:

case SC\_Add:

case SC\_Open: // case : 6

case SC\_Write: // case : 8

case SC\_Read: // case

case SC\_Close:

case SC\_Exit:

default:

}

}

ASSERTNOTREACHED();

}

void ExceptionHandler(ExceptionType which);

We add extra cases SC\_Open , SC\_Write , SC\_Read and SC\_Close in ExceptionHandler() to implement four file I/O system call. (*Open()*, *Write()*, *Read()* and *Close()*)

**# OpenField Open(char \*name);**

1. exception.cc

void ExceptionHandler(ExceptionType which)

case SC\_Open:

val=kernel->machine->ReadRegister(4);

{

char \*filename = &(kernel->machine->mainMemory[val]);

status = SysOpen(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;

break;

Since the filename was stored in register\_4, we do the similar operation in SC\_Create to load the filename from memory.

Next, we call *SysOpen()* and send the filename as argument, and this function would return status. (-1 if open fail)  
Last, we use WriteRegister to store status in register\_2, and it’s the return value of *Open()* function.

In addition, we also put some code to control the PC.

1. ksyscall.h

OpenFileId SysOpen(char \*name);

OpenFileId SysOpen(char \*name){

return kernel->fileSystem->OpenFile\_MP1\_(name);

}

Call the *OpenFile\_MP1\_(name)* function that we define additionally

1. filesys.h

OpenFileId FileSystem::OpenFile\_MP1\_(char \*name);

OpenFileId OpenFile\_MP1\_(char \*name){

OpenFile\* fd;

fd = Open(name);

if(fd == NULL) return -1;

for(int i = 0; i < 20; i++) {

if(fileDescriptorTable[i] == NULL){

// cout<<"open i="<<i<<"\n";

fileDescriptorTable[i] = fd;

return i;

}

}

return -1;

}

首先回傳一個指向由Open function所製造的物件的一個指標，如果這個物件的指標指向NuLL(代表物件沒有被製造)，會回傳-1，表示open 失敗。

假設open成功，接下來會對fileDescriptorTable做查表的動作，(其中fileDescriptorTable的定義為OpenFile \*fileDescriptorTable [20])，假如fileDescriptorTable []中存在指向NULL的指標(代表這個表是有多的空間)，會把目前指向NULL的指標指向前面Open function所製造得物件的那個指標，且回傳目前open 的file對應的fileDescriptorTable index。

當fileDescriptorTable沒有空間時，則會回傳-1 (實作 exceeding opened file limit)。

OpenFile\* Open(char \*name)

OpenFile\* Open(char \*name) {

int fileDescriptor = OpenForReadWrite(name, FALSE);

if (fileDescriptor == -1) return NULL;

return new OpenFile(fileDescriptor);

}

呼叫OpenForReadWrite()，他會回傳fd (nonegative，if open success ; -1 for otherwise)，如果打開成功則會製造一個OpenFile 的物件，並回傳指向這個物件的指標。

1. sysdep.cc

OpenForReadWrite(char \*name, bool crashOnError)

OpenForReadWrite(char \*name, bool crashOnError)

{

int fd = open(name, O\_RDWR, 0);

ASSERT(!crashOnError || fd >= 0);

return fd;

}

We use this function to open file, and it would return fd.

“fd” means the file descriptor for the opened file. It will be a non-negative integer if the file is successfully opened, or -1 if an error occurred.

1. openfile.h

class OpenFile;

1. openfile.h

class OpenFile

class OpenFile {

public:

OpenFile(int f) { file = f; currentOffset = 0; } // open the file

~OpenFile() { Close(file); } // close the file

private:

int file;

int currentOffset;

};

這個class 的Constructer會把fd 存在private parameter file裡面。

**# int Write(char \*buffer, int size, OpenFileId id)**

1. exception.cc

void ExceptionHandler( ExceptionType which )

case SC\_Write:

{

val = kernel->machine->ReadRegister(4);

char \*buffer = &(kernel->machine->mainMemory[val]);

int numChar = kernel->machine->ReadRegister(5);

int fileID = kernel->machine->ReadRegister(6);

status = SysWrite(buffer, 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;

break;

}

First, take the parameters from register\_4 and store them in val, then assign a buffer to remove the value stored in mainmory, and then take the values ​​from register\_5 and regester\_6 to numChar and fileID respectively. Then input numChar and fileID respectively through *SysWrite()* to return the status value. In addition, we also put some code to control the PC.

1. ksyscall.h

int SysWrite(char \*buffer, int size, OpenFileId id)

int SysWrite(char \*buffer, int size, OpenFileId id)

{

return kernel->fileSystem-> SysWrite \_MP1\_(buffer, size, id);

}

The *SysWrite ()* function passes the data buffer, size, and id to *SysWrite\_MP1\_()*, and then it returns the value from *SysWrite \_MP1\_()* to *SysWrite ()*.

1. filesys.h

int WriteFile\_MP1\_(char \*buffer, int size, OpenFileId id)

int WriteFile\_MP1\_(char \*buffer, int size, OpenFileId id){

if (id >= 20 || id < 0 || fileDescriptorTable[id] == NULL)

{

return -1;

}

return fileDescriptorTable[id]->Write(buffer, size); }

假如 id(table的index) 不符合定義的範圍，或是這個id指向是NULL，則會Return -1(Write file 失敗)。反之則執行這個id指向的物件，並執行這個物件裡的*Write()*。

1. filesys.h

int Write(char \*from, int numBytes)

int Write(char \*from, int numBytes)

{

int numWritten = WriteAt(from, numBytes, currentOffset);

currentOffset += numWritten;

return numWritten;

}

Write characters to an open file, and stop the operation if the write attempt fails.

呼叫*WriteAt()*並將要寫的內容(from)，長度(numBytes)和目前的位置(currentOffset)，然後將其位置加上寫入內容的長度做紀錄，最後再回傳該內容的長度。

1. openfile.h

int Write(char \*from, int numBytes)

int WriteAt(char \*from, int numBytes, int position) {

Lseek(file, position, 0);

WriteFile(file, from, numBytes);

return numBytes;

}

接下來呼叫*WriteFile()* 直接把內容寫進去。

# **int Read(char \*buffer, int size, OpenFileId id)**

1. exception.cc

void ExceptionHandler( ExceptionType which )

case SC\_Read:

{

val = kernel->machine->ReadRegister(4);

char \*buffer = &(kernel->machine->mainMemory[val]);

int numChar = kernel->machine->ReadRegister(5);

int fileID = kernel->machine->ReadRegister(6);

status = SysRead(buffer, 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;

break;

}

First, use register\_4 and mainMemory[] to load the input char to buffer. Secondly, take the values ​​from register\_5 and regester\_6 to get numChar and fileID respectively. Then , we call *SysRead ()* to do the reading process and it would return the status value. In addition, we also put some code to control the PC.

1. ksyscall.h

int SysRead(char \*buffer, int size, OpenFileId id)

int SysRead(char \*buffer, int size, OpenFileId id)

{

return kernel->fileSystem->ReadFile\_MP1\_(buffer, size, id);

}

The *SysRead()* function passes the data buffer, size, and id to *ReadFile\_MP1\_()*, and then it returns the value from *ReadFile\_MP1\_()* to *SysRead()*.

1. filesys.h

int ReadFile\_MP1\_(char \*buffer, int size, OpenFileId id)

int ReadFile\_MP1\_(char \*buffer, int size, OpenFileId id){

if (id >= 20 || id < 0 || fileDescriptorTable[id] == NULL)

{

return -1;

}

return fileDescriptorTable[id]->Read(buffer, size); }

{

假如 id(table的index) 不符合定義的範圍，或是這個id指向是NULL，則會Return -1(Read File 失敗)。反之則執行這個id指向的table物件，並執行這個物件裡的member function *Read()*。

1. openfile.cc

int Read(char \*into, int numBytes)

int Read(char \*into, int numBytes) {

int numRead = ReadAt(into, numBytes, currentOffset);

currentOffset += numRead;

return numRead;

}

呼叫*ReadAt()*並將要讀的內容 (from)，長度 (numBytes) 和目前的位置(currentOffset) 當作 arguement，然後將其位置加上讀入內容的長度做紀錄，最後再回傳該內容的長度。

1. openfile.h and sysdep.cc;

int Write(char \*from, int numBytes)

ReadPartial(int fd, char \*buffer, int nBytes)

int ReadAt(char \*from, int numBytes, int position) {

Lseek(file, position, 0);

return ReadPartial (file, from, numBytes);

}

int ReadPartial(int fd, char \*buffer, int nBytes)

{

return read(fd, buffer, nBytes);

}

接下來呼叫*ReadPartial()*直接把內容讀進去，其中的一個 argument file 為儲存在 table 物件裡的實際檔案名，而*ReadPartial()* 會呼叫*read()* function 做實際的 read 動作。

**# int Close(OpenField id);**

1. exception.cc

void ExceptionHandler(ExceptionType which)

case SC\_Close: {

int fileID = kernel->machine->ReadRegister(4);

status = SysClose(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;

break;

}

Since the id was stored in register\_4, we use ReadRegister(4) to get fileID.

Next, we call *SysClose()* function*.*

In addition, we also put some code to control the PC.

1. ksyscall.h

int SysClose(OpenFileId id)

int SysClose(OpenFileId id){

return kernel->fileSystem->CloseFile\_MP1\_(id);

}

Call the CloseFile\_MP1\_(id) function that we define additionally

1. filesys.h

int FileSystem::CloseFile\_MP1\_(OpenFileId id){

int CloseFile\_MP1\_(OpenFileId id){

if (id >= 20 || id < 0 || fileDescriptorTable[id] == NULL)

{

return 0;

}

delete fileDescriptorTable[id];

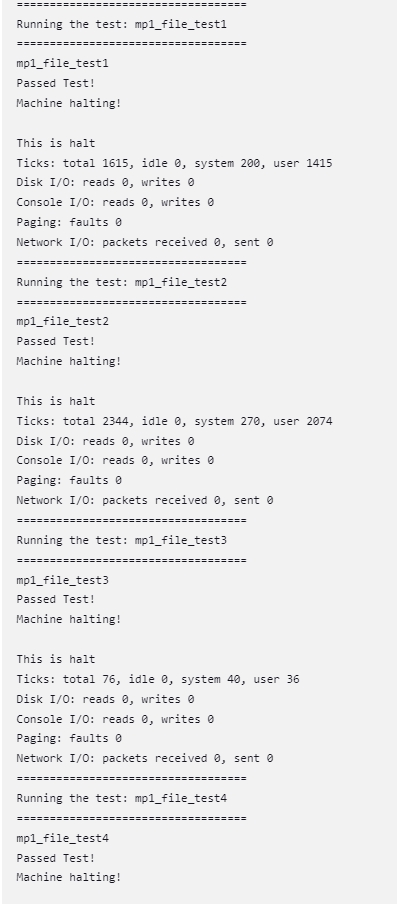
fileDescriptorTable[id] = NULL;

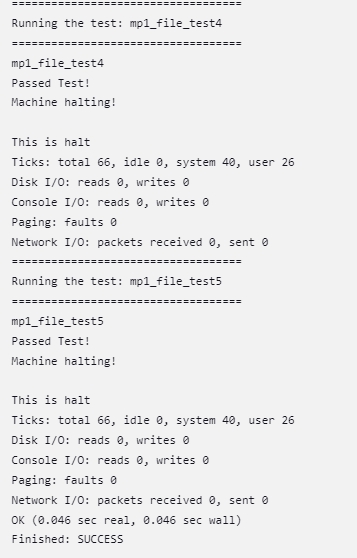
return 1;

}

假如 id(table的index) 不符合定義的範圍，或是這個id指向是NULL，則會Return 0(close file 失敗)。反之則刪除這個fileDescriptorTable[] (即這個id 我們決定close，故從我們處理檔案的table 裡刪掉 )，然後並讓她重新指向NULL，之後再return 1(close file 成功)。

**\*\*The final test for the Open(), Write(), Read(), Close()\*\***

****

****

**Part 3:Contribution**

|  |  |  |
| --- | --- | --- |
|  | 張世傑 | 吳孟儒 |
| Trace of Halt | ˇ | ˇ |
| Report of Halt | ˇ |  |
| Trace of Create | ˇ | ˇ |
| Report of Create |  | ˇ |
| Details of Makefile | ˇ | ˇ |
| Console I/O system call | ˇ | ˇ |
| File I/O system | ˇ | ˇ |