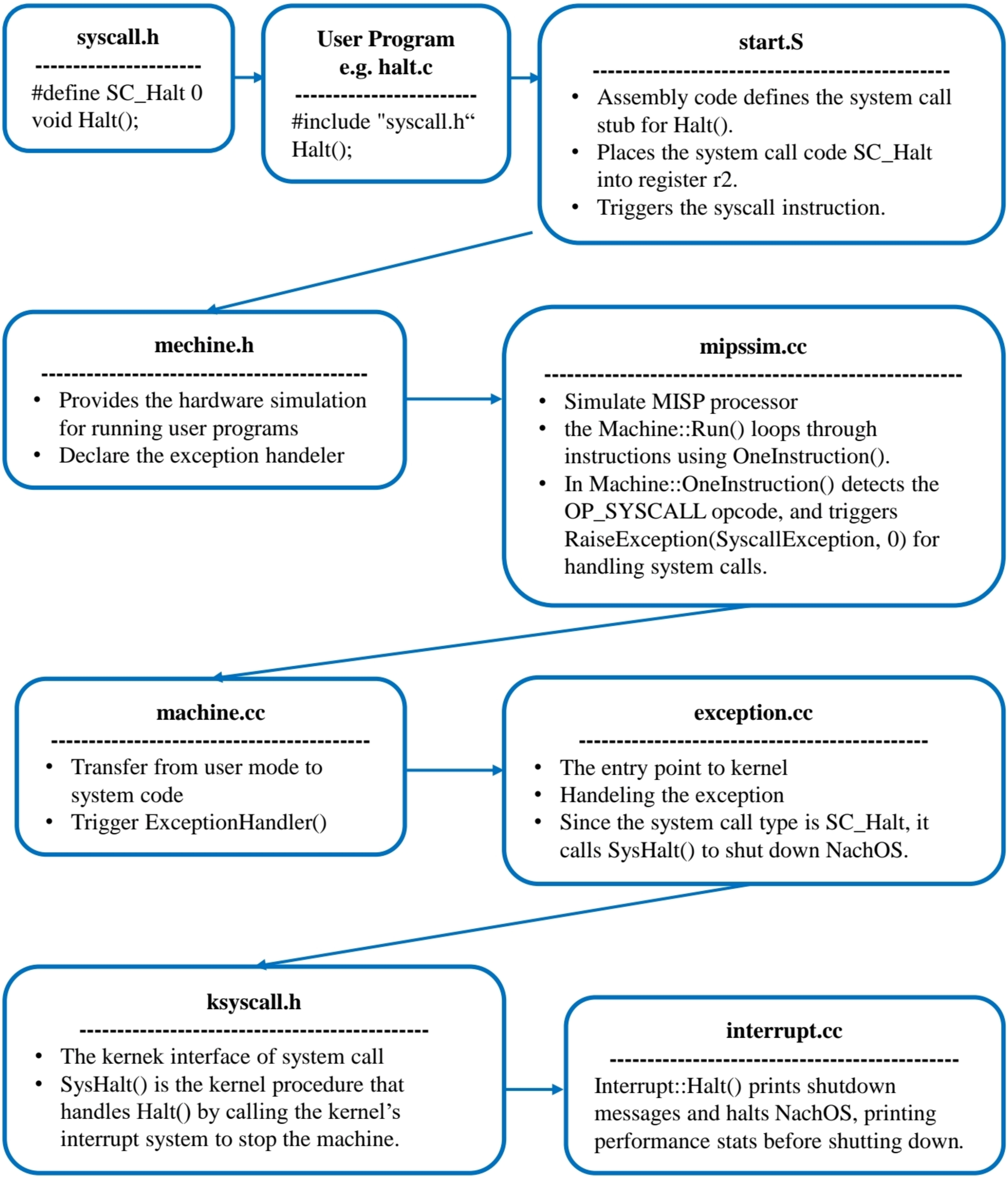
OS HW01 GROUP 21

Part 1:Trace Code Result

1. Flow Chart of Halt() System Call:



syscall.h

```
#define SC_Halt 0
void Halt();
```

User Program e.g. halt.c

```
#include "syscall.h“
Halt();
```

start.S

- Assembly code defines the system call stub for Halt().
- Places the system call code SC_Halt into register r2.
- Triggers the syscall instruction.

mechine.h

- Provides the hardware simulation for running user programs
- Declare the exception handeler

mipssim.cc

- Simulate MISP processor
- the Machine::Run() loops through instructions using OneInstruction().
- In Machine::OneInstruction() detects the OP_SYSCALL opcode, and triggers RaiseException(SyscallException, 0) for handling system calls.

machine.cc

- Transfer from user mode to system code
- Trigger ExceptionHandler()

exception.cc

- The entry point to kernel
- Handeling the exception
- Since the system call type is SC_Halt, it calls SysHalt() to shut down NachOS.

ksyscall.h

- The kernek interface of system call
- SysHalt() is the kernel procedure that handles Halt() by calling the kernel’s interrupt system to stop the machine.

interrupt.cc

Interrupt::Halt() prints shutdown messages and halts NachOS, printing performance stats before shutting down.

2. Details of Trace Halt() Code

(1) userprog/ syscall.h

Define the Halt() in Nachos system call interface.

```
#define SC_Halt     0

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

(2) User application, for example: test/halt.c

Call the function in user application.

```
#include "syscall.h"

Halt();
```

(3) Test/Start.S

The assembly language stub places the system call code into a register.

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

(4) Error checking

i) machine/ mipssim.cc

The function Run() is called when the program starts up

```
Void Machine::Run() {
    Instruction *instr = new Instruction;  // storage for decoded instruction
    for (;;) {
        OneInstruction(instr);
    }
}
```

Invoke the exception handler after interrupt, and after it returns, will return to Run().

```
Void Machine::OneInstruction(Instruction *instr) {
switch (instr->opCode) {
     case OP_SYSCALL:
         RaiseException(SyscallException, 0);
         return;
```

ii) machine/ machine.cc

Transfer from user mode to kernel mode

```
Void Machine::RaiseException(ExceptionType which, int badVAddr)
{
    kernel->interrupt->setStatus(SystemMode);
    ExceptionHandler(which);        // interrupts are enabled at this point
    kernel->interrupt->setStatus(UserMode);
}
```

iii) userprog/ exception.cc

Do the exception handeling, and call SysHalt().

```
Void ExceptionHandler(ExceptionType which){
    int type = kernel->machine->ReadRegister(2);
    switch (which) {
    case SyscallException:
        switch(type) {
        case SC_Halt:
            DEBUG(dbgSys, "Shutdown, initiated by user program.\n");
            SysHalt();
                        cout<<"in exception\n";
            ASSERTNOTREACHED();
            break;

```

(5) Kernel procedures

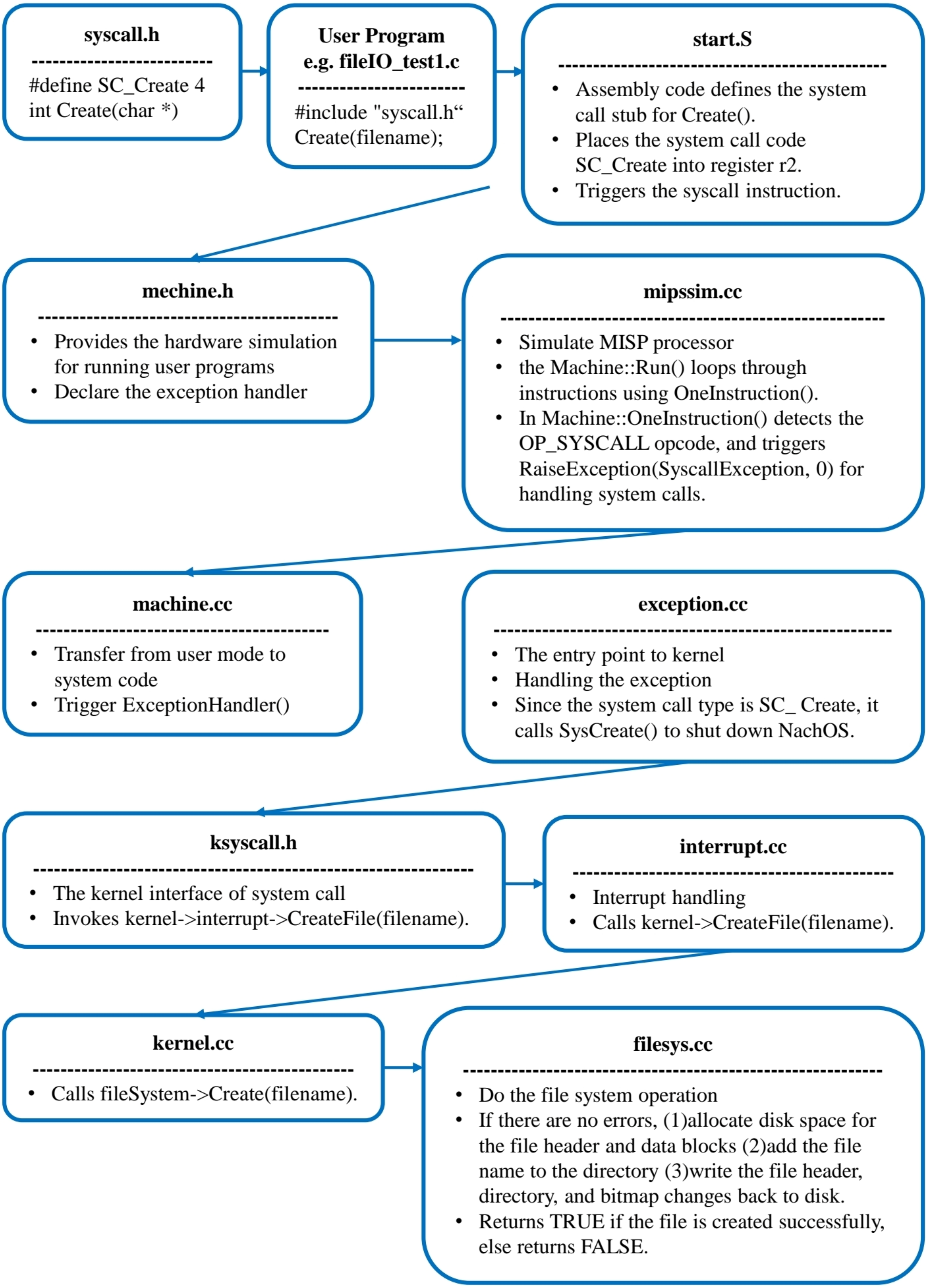
i) userprog/ ksyscall.h

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

ii) machine/ interrupt.cc

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

3. Flow Chart of Create() System Call:



syscall.h

#define SC_Create 4
int Create(char *)

User Program
e.g. fileIO_test1.c

#include "syscall.h“
Create(filename);

start.S

- Assembly code defines the system call stub for Create().
- Places the system call code SC_Create into register r2.
- Triggers the syscall instruction.

mechine.h

- Provides the hardware simulation for running user programs
- Declare the exception handler

mipssim.cc

- Simulate MISP processor
- the Machine::Run() loops through instructions using OneInstruction().
- In Machine::OneInstruction() detects the OP_SYSCALL opcode, and triggers RaiseException(SyscallException, 0) for handling system calls.

machine.cc

- Transfer from user mode to system code
- Trigger ExceptionHandler()

exception.cc

- The entry point to kernel
- Handling the exception
- Since the system call type is SC_ Create, it calls SysCreate() to shut down NachOS.

ksyscall.h

- The kernel interface of system call
- Invokes kernel->interrupt->CreateFile(filename).

interrupt.cc

- Interrupt handling
- Calls kernel->CreateFile(filename).

kernel.cc

- Calls fileSystem->Create(filename).

filesys.cc

- Do the file system operation
- If there are no errors, (1)allocate disk space for the file header and data blocks (2)add the file name to the directory (3)write the file header, directory, and bitmap changes back to disk.
- Returns TRUE if the file is created successfully, else returns FALSE.

4. Details of Trace Create() Code:

(1) userprog/ syscall.h

Define the Create() in Nachos system call interface.

```
#define SC_Create   4

/* Create a Nachos file, with name "name" */
/* Return 1 on success, negative error code on failure */
int Create(char *name);
```

(2) User application, for example: test/fileIO_test1.c

Call the function in user application.

```
#include "syscall.h"

int success = Create("file1.test");
```

(3) test/Start.S

The assembly language stub places the system call code into a register.

```
.globl Create
.entCreate
Create:
addiu $2,$0,SC_Create
syscall
j   $31
.end Create
```

(4) Error checking

i) machine/ mipssim.cc

The function Run() is called when the program starts up

```
Void Machine::Run() {
    Instruction *instr = new Instruction;  // storage for decoded instruction
    for (;;) {
        OneInstruction(instr);
    }
}
```

Invoke the exception handler after interrupt, and after it returns, will return to Run().

```
Void Machine::OneInstruction(Instruction *instr) {
switch (instr->opCode) {
     case OP_SYSCALL:
         RaiseException(SyscallException, 0);
         return;
```

ii) machine/ machine.cc

Transfer from user mode to kernel mode

```
Void Machine::RaiseException(ExceptionType which, int badVAddr)
{
    kernel->interrupt->setStatus(SystemMode);
    ExceptionHandler(which);        // interrupts are enabled at this point
    kernel->interrupt->setStatus(UserMode);
}
```

iii) **userprog/ exception.cc**

Do the exception handling, and call SysCreate().

```
Void ExceptionHandler(ExceptionType which){
    int type = kernel->machine->ReadRegister(2);
    switch (which) {
    case SyscallException:
        switch(type) {
        case SC_Create:
            {
            char *filename = &(kernel->machine->mainMemory[val]);
            status = SysCreate(filename);
           kernel->machine->WriteRegister(2, (int) status);
            }
```

(2) Kernel procedures

i) userprog/ ksyscall.h

```
int SysCreate(char *filename)
{
    return kernel->interrupt->CreateFile(filename);
}
```

ii) machine/ interrupt.cc

```
Int Interrupt::CreateFile(char *filename)
{
    return kernel->CreateFile(filename);
}
```

iii) Threads/kernel.cc

```
int Kernel::CreateFile(char *filename)
{
    return fileSystem->Create(filename);
}
```

iv) filesys/ filesys.cc

```
Bool FileSystem::Create(char *name, int initialSize){
    if(theres any error){success = FALSE;}
    else {
        success = TRUE;
         hdr->WriteBack(sector);
        directory->WriteBack(directoryFile);
        freeMap->WriteBack(freeMapFile);
        }
    delete hdr;
    return success;
}
```

5. Details of Makefile

(1) Define variables

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

INCDIR =-I [...]
CFLAGS = -G 0 -c [...] -B [...]
```

Code explanation:

CC is the C compiler, AS is the assembler, and LD is the linker.

INCDIR defines the include directories for the compiler.

CFLAGS defines the compiler flags.

(2) Assign what programs to build

```
ifeq ($(hosttype),unknown)
PROGRAMS = unknownhost
else
PROGRAMS = halt
Endif

all: $(PROGRAMS)
```

(3) Compiling the program

```
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
```

Code explanation:

1. generate start.o by compiling start.S.
2. compiles halt.c into an object file halt.o.
3. links halt.o and start.o into the final executable halt.coff, and transform to Nachos Object File Format.

(4) Remove the file

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

distclean: clean
    $(RM) -f $(PROGRAMS)
```

(5) Error handeling

```
unknownhost:
    @echo Host type could not be determined.
    @echo make is terminating.
```

```
	@echo If you are on an MFCF machine, contact the instructor to report this problem
	@echo Otherwise, edit Makefile.dep and try again.
```

Part 2:Implement System Call

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

   (1) Implement PrintInt

   a. **syscall.h**

      Define the system call code for PrintInt() to tell the kernel which systemcall is being asked for.

```
#define SC_PrintInt   16
void PrintInt(int number);
```

   b. **Start.S**

      The assembly language stub places the system call code into a register.

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

   c. **exception.cc**

      In the ExceptionHandler(), add a case to handle the system call "PrintInt".

      Read the integer from register 4.

      Invoke SysPrintInt(), which is defined in ksyscall.h.

```
case SC_PrintInt:
            SysPrintInt(kernel->machine->ReadRegister(4));
            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;
```

   d. **ksyscall.h**

```
void SysPrintInt(int number ){
    return kernel->interrupt->PrintInt(number);
}
```

   e. **interrupt.h**

```
void PrintInt(int number);
```

   f. **interrupt.cc**

```
void
Interrupt::PrintInt(int number)
{
    return kernel->PrintInt(number);
}
```

g. kernel.h

```
void PrintInt(int number);
```

h. kernel.cc

Transfer the integer into string (buffer[16]) and save the length of the string (len).

```
void Kernel::PrintInt(int number) {
    char buffer[16];
    int index = 0;
    int len=0;
    if (number < 0) {
        buffer[index++] = '-';
        number = -number;
        len++;
    }
    if (number == 0) {
        buffer[index++] = '0';
        len++;
    }
    else {
        int start = index;
        while (number > 0) {
            buffer[index++] = '0' + (number % 10);
            number /= 10;
            len++;
        }
        int end = index - 1;
```

Reverse the string because the number in buffer is stored backward.

```
        while (start < end) {
            char temp = buffer[start];
            buffer[start] = buffer[end];
            buffer[end] = temp;
            start++;
            end--;
        }
    }
    buffer[index++] = '\n';
    buffer[index] = '\0';
    len=len+2;
```

Call the PutInt function in synchconsole.cc to write the number.

```
    synchConsoleOut->PutInt(buffer,len);
}
```

i. synchconsole.h

Define the PutInt() in class SynchConsoleOutput.

```
void PutInt(char *ch,int len);
```

j. synchconsole.cc

Call the PutInt() in console.cc.

```
void
SynchConsoleOutput::PutInt(char *ch,int len)
{
    lock->Acquire();
    consoleOutput->PutInt(ch,len);
    waitFor->P();
    lock->Release();
}
```

k. console.h

```
void PutInt(char *ch,int len);
```

l. console.cc

Use the WriteFile function to write the integer on the console.

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

2. Detail of your File I/O system call implementation

(2) Implement Open

a. exception.cc

In the ExceptionHandler(), add a case to handle the system call "Open"

Read the filename from register 4.

Invoke SysOpen(), which is defined in ksyscall.h

```
case SC_Open:
    val = kernel->machine->ReadRegister(4);
    {
    char *filename = &(kernel->machine->mainMemory[val]);
    OpenFileId fid = SysOpen(filename);
    kernel->machine->WriteRegister(2, fid);
    }
    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;
```

b. **ksyscall.h**

```
OpenFileId SysOpen(char *filename) {
    // return value
    // fileId: sucess
    // -1: failed
    return (OpenFileId) kernel->interrupt->Open(filename);
}
```

c. **interrupt.h**

```
OpenFileId Open(char *filename);
```

d. **interrupt.cc**

```
OpenFileId
Interrupt::Open(char *filename)
{
    return kernel->Open(filename);
}
```

e. **kernel.h**

```
OpenFileId Open(char *filename);
```

f. **kernel.cc**

Use a table to store the pointer of each file.

```
#define MAX_OPEN_FILES 20  // Set a limit for open files
OpenFile *openFileTable[MAX_OPEN_FILES];  // Array to hold open files

Kernel::Kernel(int argc, char **argv)
{
    /*Open ---------------------------------------------------------*/
    // Initialize open file table
    for (int i = 0; i < MAX_OPEN_FILES; i++) {
        openFileTable[i] = NULL;
    }
```

Call the function Open() in filesystem. If successfully open the file, and the total opened file is less than 20, store the file pointer into the table.

```
OpenFileId Kernel::Open(char *filename) {
    OpenFile *openFile = fileSystem->Open(filename);
    if (openFile == NULL) {
        return -1;  // Failed to open the file
    }
    for (int i = 0; i < MAX_OPEN_FILES; i++) {
        if (openFileTable[i] == NULL) {
            openFileTable[i] = openFile;
            return i;
        }
    }
    // No free file descriptors
    delete openFile;
    return -1;
}
```

(3) Implement Write

a. exception.cc

In the ExceptionHandler(), add a case to handle the system call “Write”.
Read the arguments (text, size, fid) from register 4, 5, and 6.
Invoke SysWrite(), which is defined in ksyscall.h

```
case SC_Write:
    val = kernel->machine->ReadRegister(4);
    {
        char *text = &(kernel->machine->mainMemory[val]);
        int size = kernel->machine->ReadRegister(5);
        OpenFileId fid = kernel->machine->ReadRegister(6);
        status = SysWrite(text,size,fid);
        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;
```

b. ksyscall.h

```
int SysWrite(char *text, int size, OpenFileId fid){
    return kernel->interrupt->WriteFile(text,size,fid);
}
```

c. interrupt.h

```
int WriteFile(char *text, int size, OpenFileId fid);
```

d. interrupt.cc

```
int
Interrupt::WriteFile(char *text, int size, OpenFileId fid)
{
    return kernel->WriteFile(text,size,fid);
}
```

e. kernel.h

```
int WriteFile(char *text, int size, OpenFileId fid);
```

f. kernel.cc

Given the file index, find the pointer of the corresponding file from the table, then call the function Write() in openfile.cc

```
int Kernel::WriteFile(char *text, int size, OpenFileId fid) {
    if(0<= fid && fid<= MAX_OPEN_FILES){
        OpenFile *openFile = openFileTable[fid];
        return openFile->Write(text,size);
    }
    return -1;
}
```

(4) Implement Read

a. exception.cc

In the ExceptionHandler(), add a case to handle the system call “Read”.
Read the arguments (text, size, fid) from register 4, 5, and 6.
Invoke SysRead(), which is defined in ksyscall.h

```
case SC_Read:
    val = kernel->machine->ReadRegister(4);
    {
        char *text = &(kernel->machine->mainMemory[val]);
        int size = kernel->machine->ReadRegister(5);
        OpenFileId fid = kernel->machine->ReadRegister(6);
        status = SysRead(text,size,fid);
        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;
```

b. ksyscall.h

```
int SysRead(char *text, int size, OpenFileId fid){
    return kernel->interrupt->ReadFile(text,size,fid);
}
```

c. interrupt.h

```
int ReadFile(char *text, int size, OpenFileId fid);
```

d. interrupt.cc

```
int
Interrupt::ReadFile(char *text, int size, OpenFileId fid)
{
    return kernel->ReadFile(text,size,fid);
}
```

e. kernel.h

```
int ReadFile(char *text, int size, OpenFileId fid);
```

f. kernel.cc

Given the file index, find the pointer of the corresponding file from the table, then call the function Read() in openfile.cc

```
int Kernel::ReadFile(char *text, int size, OpenFileId fid) {
    if(0<= fid && fid<= MAX_OPEN_FILES){
        OpenFile *openFile = openFileTable[fid];
         if (openFile == nullptr) {
            return -1;
        }
        return openFile->Read(text,size);
    }
    return -1;
}
```

(5) Implement Close

a. exception.cc

In the ExceptionHandler(), add a case to handle the system call “Close”

Read the FileID from register 4.

Invoke SysClose(), which is defined in ksyscall.h

```
case SC_Close:
    val = kernel->machine->ReadRegister(4);
    status = SysClose(val);
    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;
```

b. ksyscall.h

```
int SysClose(OpenFileId fid){
    return kernel->interrupt->Close(fid);
}
```

c. interrupt.h

```
int Close(OpenFileId fid);
```

d. interrupt.cc

```
int
```

```
Interrupt::Close( OpenFileId fid)
{
    return kernel->Close(fid);
}
```

e. **kernel.h**

```
int Close(OpenFileId fid);
```

f. **kernel.cc**

Check whether the id is valid or not, then use the delete function to invoke the destructor of OpenFile object. Last, assign the null value to table to reset the state of it.

```
int Kernel::Close(OpenFileId fid) {
     if (0 <= fid && fid < MAX_OPEN_FILES) {
        OpenFile *openFile = openFileTable[fid];
        if (openFile != nullptr) {
            delete openFile;
            openFileTable[fid] = nullptr;
            return 1;
        }
    }
    return 0;
}
```

Part 3:Contribution

1. **Describe details and percentage of each member’s contribution.**

313551047 陳以瑄, contribution (50%):

(1) Trace code

(2) Implement Open()

(3) Implement Write()

111550150 俞祺譯, contribution (50%):

(1) Trace code

(2) Implement PrintInt()

(3) Implement Read()

(4) Implement Close()