

Q4.1

Predefined:

```
FILE *file[5];
Thread *thread[5];
int fileC = 2;
int threadC = 1;
Thread *joining;
int jointed;
int joinState = 0;
```

SC_create:

```
case SC_Create:
{
    char* name = (char*) malloc(sizeof(char)*20);
    int buf;
    char temp[1];
    int i = 0;
    while(true)
    {
        kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
        sprintf(temp, "%s",&buf);
        if (strcmp(temp, "") == 0)
        {
            name[i] = '\\0';
            break;
        }
        name[i] = *temp;
        i++;
    }
    FILE *fp = fopen(name, "w");
    if (fp == NULL)
    {
        kernel->machine->WriteRegister(2, -1);
        break;
    }
    else{
        kernel->machine->WriteRegister(2, 1);
    }
}
```

```

        break;
    }
}

```

```

case SC_Create:
{
    char* name = (char*) malloc(sizeof(char)*20);
    int buf;
    char temp[1];
    int i = 0;
    while(true)
    {
        kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
        sprintf(temp,"%s",&buf);
        if(strcmp(temp,"") == 0)
        {
            name[i] = '\0';
            break;
        }
        name[i] = *temp;
        i++;
    }
    FILE *fp = fopen(name, "w");
    if(fp == NULL)
    {
        kernel->machine->WriteRegister(2, -1);
        break;
    }
    else{
        kernel->machine->WriteRegister(2, 1);
        break;
    }
}
}

```

SC_open:

```

case SC_Open:
{
    char* name = (char*) malloc(sizeof(char)*20);
    int buf;
    char temp[1];
    int i = 0;
    while(true)
    {
        kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);

```

```
    sprintf(temp, "%s", &buf);  
    if (strcmp(temp, "") == 0)  
    {  
        name[i] = '\\0';  
        break;  
    }  
    name[i] = *temp;  
    i++;  
}  
FILE *open = fopen(name, "r+");  
if (open == NULL)  
{  
    kernel->machine->WriteRegister(2, -1);  
    break;  
}  
*(file+fileC) = open;  
kernel->machine->WriteRegister(2, fileC);  
fileC++;  
break;  
}
```

```

..... case SC_Open:
..... {
.....     char *name = (char *) malloc(sizeof(char)*20);
.....     int buf;
.....     char temp[1];
.....     int i = 0;
.....     while(true)
.....     {
.....         kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
.....         sprintf(temp, "%s",&buf);
.....         if (strcmp(temp, "") == 0)
.....         {
.....             name[i] = '\0';
.....             break;
.....         }
.....         name[i] = *temp;
.....         i++;
.....     }
.....     FILE *open = fopen(name, "r+");
.....     if (open == NULL)
.....     {
.....         kernel->machine->WriteRegister(2, -1);
.....         break;
.....     }
.....     *(file+fileC) = open;
.....     kernel->machine->WriteRegister(2, fileC);
.....     fileC++;
.....     break;
..... }

```

SC_write:

```

case SC_Write:
{
    int buf;
    int k;
    int id;
    k = kernel->machine->ReadRegister(5);
    id = kernel->machine->ReadRegister(6);

    //printf("In ExceptionHandler, Write System call is made.The
first parameter is %d the second parameter is %d the third parameter is

```

```
%d\n", (int)kernel->machine->ReadRegister(4), (int)kernel->machine->ReadRegister(5), (int)kernel->machine->ReadRegister(6));

    if (id != 1){
        if (*(file+id) == NULL)
        {
            kernel->machine->WriteRegister(2, -1);
            break;
        }
        for (int i = 0; i < k; i++){

kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1, &buf);
            fputc(buf, *(file+id));
        }
        kernel->machine->WriteRegister(2, k);
    }
    else{
        for (int i = 0; i < k; i++) {

kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1, &buf);
            printf("%s", &buf);
        }
        kernel->machine->WriteRegister(2, k);
    }
    break;
}
```

```

.....case SC_Write:
.....{
.....    int buf;
.....    int k;
.....    int id;
.....    k = kernel->machine->ReadRegister(5);
.....    id = kernel->machine->ReadRegister(6);
.....
.....    //printf("In ExceptionHandler, Write System call is made. The first parameter is %d the ..
.....    if (id != 1){
.....        if (*(file+id) == NULL)
.....        {
.....            kernel->machine->WriteRegister(2, -1);
.....            break;
.....        }
.....        for (int i = 0; i < k; i++){
.....            kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i, 1, &buf);
.....            fputc(buf, *(file+id));
.....        }
.....        kernel->machine->WriteRegister(2, k);
.....    }
.....    else{
.....        for (int i = 0; i < k; i++) {
.....            kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i, 1, &buf);
.....            printf("%s", &buf);
.....        }
.....        kernel->machine->WriteRegister(2, k);
.....    }
.....    break;
.....}
.....case SC_Create:
.....{

```

SC_read:

```

case SC_Read:
{
    int buf;
    int k;
    k = kernel->machine->ReadRegister(5);
    int id = kernel->machine->ReadRegister(6);
    for (int i = 0; i < k; i++)
    {
        buf = fgetc(*(file+id));
        kernel->machine->WriteMem(kernel->machine->ReadRegister(4)+i,
1, buf);
    }
    kernel->machine->WriteRegister(2, k);

    break;

```

```
}
```

```
91
92     case SC_Read:
93     {
94         int buf;
95         int k;
96         k = kernel->machine->ReadRegister(5);
97         int id = kernel->machine->ReadRegister(6);
98         for (int i = 0; i < k; i++)
99         {
100             buf = fgetc(*(file+id));
101             kernel->machine->WriteMem(kernel->machine->ReadRegister(4)+i, 1, buf);
102         }
103         kernel->machine->WriteRegister(2, k);
104
105         break;
106     }
107
```

SC_exec:

```
case SC_Exec:
{
    char* name = (char*) malloc(sizeof(char)*26);
    int buf;
    char temp[1];
    int i = 0;
    while(true)
    {
        kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
        sprintf(temp, "%s",&buf);
        if (strcmp(temp, "") == 0)
        {
            name[i] = '\\0';
            break;
        }
        name[i] = *temp;
        i++;
    }
    kernel->machine->WriteRegister(2, threadC);
    Thread * t = new Thread(name);
    t->Fork((VoidFunctionPtr)RunUserProg1, name);
    *(thread+threadC) = t;
}
```

```

        threadC++;

        break;
    }

```

```

..... case SC_Exec:
..... {
.....     char* name = (char*) malloc(sizeof(char)*26);
.....     int buf;
.....     char temp[1];
.....     int i = 0;
.....     while(true)
.....     {
.....         kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
.....         sprintf(temp, "%s",&buf);
.....         if (strcmp(temp, "") == 0)
.....         {
.....             name[i] = '\0';
.....             break;
.....         }
.....         name[i] = *temp;
.....         i++;
.....     }
.....     kernel->machine->WriteRegister(2, threadC);
.....     Thread *t = new Thread(name);
.....     t->Fork((VoidFunctionPtr)RunUserProg1, name);
.....     *(thread+threadC) = t;
.....     threadC++;
.....     break;
..... }

```

SC_close:

```

case SC_Close:
{
    int id = kernel->machine->ReadRegister(4);
    if (*(file+id) == NULL){
        kernel->machine->WriteRegister(2, -1);
        break;
    }
    fclose(*(file+id));
    kernel->machine->WriteRegister(2, 1);
}

```



```

        break;
    }

    default:
        cerr << "Unexpected system call " << type << "\n";
        break;
}

```

```

..... case SC_Close:
..... {
.....     int id = kernel->machine->ReadRegister(4);
.....     if (*(file+id) == NULL){
.....         kernel->machine->WriteRegister(2, -1);
.....         break;
.....     }
.....     fclose(*(file+id));
.....     kernel->machine->WriteRegister(2, 1);
.....     break;
..... }
.....
..... default:
.....     cerr << "Unexpected system call " << type << "\n";
.....     break;
..... }

```

Q4.2

SC_exec:

```

    case SC_Exec:
    {
        char* name = (char*) malloc(sizeof(char)*26);
        int buf;
        char temp[1];
        int i = 0;
        while(true)
        {
            kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
            sprintf(temp, "%s",&buf);
            if (strcmp(temp, "") == 0)
            {
                name[i] = '\0';
                break;
            }

```

```

    }
    name[i] = *temp;
    i++;
}
kernel->machine->WriteRegister(2, threadC);
Thread * t = new Thread(name);
t->Fork((VoidFunctionPtr)RunUserProg1, name);
*(thread+threadC) = t;

threadC++;

break;
}

```

```

.....case SC_Exec:
.....{
.....char*.name=(char*) malloc(sizeof(char)*26);
.....int buf;
.....char temp[1];
.....int i=0;
.....while(true)
.....{
.....kernel->machine->ReadMem(kernel->machine->ReadRegister(4)+i,1,&buf);
.....sprintf(temp,"%s",&buf);
.....if(strcmp(temp,"")==0)
.....{
.....name[i]='\0';
.....break;
.....}
.....name[i]=*temp;
.....i++;
.....}
.....kernel->machine->WriteRegister(2, threadC);
.....Thread *t=new Thread(name);
.....t->Fork((VoidFunctionPtr)RunUserProg1, name);
.....*(thread+threadC) = t;
.....
.....threadC++;
.....
.....break;
.....}
.....case SC_Close:

```

SC_join:

```

case SC_Join:
{
    jointed = kernel->machine->ReadRegister(4);
    joining = kernel->currentThread;
    kernel->machine->WriteRegister(2, 0);
    kernel->interrupt->SetLevel(IntOff);
    kernel->currentThread->Sleep(FALSE);
    break;
}

```

```

.....case SC_Join:
.....{
.....    jointed = kernel->machine->ReadRegister(4);
.....    joining = kernel->currentThread;
.....    kernel->machine->WriteRegister(2, 0);
.....    kernel->interrupt->SetLevel(IntOff);
.....    kernel->currentThread->Sleep(FALSE);
.....    break;
.....}

```

SC_exit:

```

case SC_Exit:
{
    int k = kernel->machine->ReadRegister(4);
    if (k == 0)
    {
        printf("Process %s exited normally\n",
kernel->currentThread->getName());
    }
    else{
        joinState = -1;
        printf("Process %s exited abnormally\n",
kernel->currentThread->getName());
    }
    if (joining != NULL && kernel->currentThread == *(thread +
jointed))
    {
        kernel->interrupt->SetLevel(IntOff);
        kernel->scheduler->ReadyToRun(joining);
        joining = NULL;
    }
}

```

```

        jointed = 0;
    }
    kernel->currentThread->Finish();
    break;
}

```

```

.....case SC_Exit:
.....{
.....    int k = kernel->machine->ReadRegister(4);
.....    if (k == 0)
.....    {
.....        printf("Process %s exited normally\n", kernel->currentThread->getName());
.....    }
.....    else{
.....        joinState = -1;
.....        printf("Process %s exited abnormally\n", kernel->currentThread->getName());
.....    }
.....    if (joining != NULL && kernel->currentThread == *(thread + jointed))
.....    {
.....        kernel->interrupt->SetLevel(IntOff);
.....        kernel->scheduler->ReadyToRun(joining);
.....        joining = NULL;
.....        jointed = 0;
.....    }
.....    kernel->currentThread->Finish();
.....    break;
.....}

```

Exception.cc/UserProg

```

void
RunUserProg1(void *filename) {
    AddrSpace *space = new AddrSpace;
    ASSERT(space != (AddrSpace *)NULL);
    if (space->Load((char*)filename)) { // load the program into the
space
        space->Execute(); // run the program
    }
    ASSERTNOTREACHED();
}

```

```

void
RunUserProg1(void *filename) {
    AddrSpace *space = new AddrSpace;
    ASSERT(space != (AddrSpace *)NULL);
    if(space->Load((char *)filename)) { //load the program into the space
        space->Execute(); //run the program
    }
    ASSERTNOTREACHED();
}

```

TTesting

This program create a file file-test0.txt, and write "Hello from file-test0" into file-test0.txt five times.

Thus the result is that program create a file file-test0.txt with content of 5 lines of "Hello from file-test0". This program calls Create("file-test0.txt") to create file file-test0.txt, Open("filetest0.txt") to open file file-test0.txt. If the program open normally, the program calls Write(str, 22, output); to write "Hello from file-test0" to file file-test0.txt. Finally the program calls Close(output) to close the file descriptor. The following screenshot shows the output of the program.

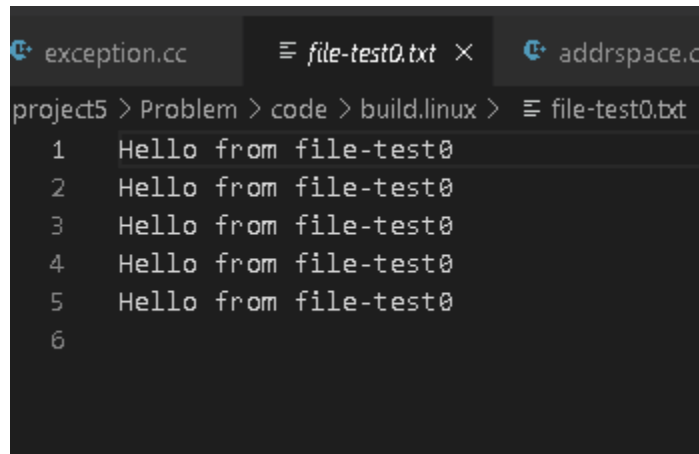
The console output is

```

zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/file-test0 -x ../test2/file-test1
Process ../test2/file-test0 exited normally
File doesn't exist
Process ../test2/file-test1 exited abnormally

```

The file-test0.txt



```

exception.cc  file-test0.txt  addrspace.c
project5 > Problem > code > build.linux > file-test0.txt
1  Hello from file-test0
2  Hello from file-test0
3  Hello from file-test0
4  Hello from file-test0
5  Hello from file-test0
6

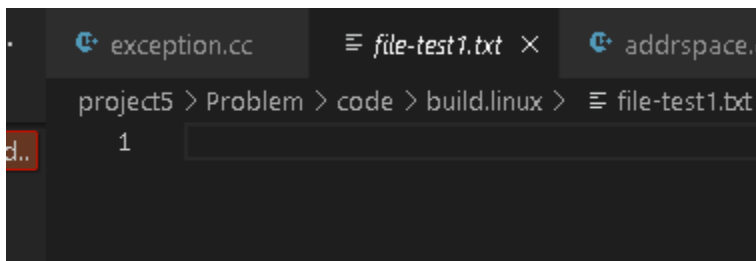
```

This program creates file file-test1.txt by calling Create("file-test1.txt") Then the program opens the file file-test1.txt by calling Open("file-test1.txt") if create file filetest1.txt normally, or call Write("Problem Creating a file\n", 24, output) to write a line of "Problem Creating a file". Then

the program calls `Open("nofile.txt")`, if the this file can not open correctly, the program prints "File doesn't exist". After that, if the file `nofile.txt`, the program will call `Write(str, 22, outfile0)` to write "Hello from file-test1" five times.

From the observation of the code, the output of the program should be "File doesn't exist" due to file `nofile.txt` is not exist. And there is a empty file `file-test1.txt` because the program will exit abnormally for not existence of file `nofile.txt`. The following screenshot displays the output of the program.

```
zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/file-test0 -x ../test2/file-test1
Process ../test2/file-test0 exited normally
File doesn't exist
Process ../test2/file-test1 exited abnormally
□
```



From the code reading of the program, the following file system calls are called:

`Create()`: The program calls `Create("file-test2.txt")` to create file `filetest2.txt` and to create("file-test2a.txt")

```
char buffer[100];
char* str = "Hello from file-test2\n";
int i,j;

if (Create("file-test2.txt") == -1) {
    Exit(1);
}
```

`Open()`: The program calls `Open("file-test2.txt")` to open file `file-test2.txt` and `Open("file-test2a.txt")` to open file `file-test2a.txt`.

```
outfile0 = Open("file-test2.txt");
o o
```

```
if ((outfile1 = Open("file-test2a.txt")) == -1)
    Exit(1);z
```

If fail to open file `file-test2a.txt`, the program will be exit abnormally.

`Write()`: The program calls `Write(str, 22, outfile0)` to write "Hello from file-test2" to file `file-test2.txt`.

```

for (i = 0; i < 5; i++)
{
    Write(str, 22, outfile0);
    for (j = 0; j < 10000; j++);
}

Close(outfile0);

```

Close() will calls to close(outfile0);

As the result, the program will output nothing related to file system and a line about exist abnormally For file, and create file file-test2.txt with content of 5 line "Hello from filetest2" and empty file file-test2a.txt The following screenshot displays the output of the program.

```

zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/file-test2
Process ../test2/file-test2 exited normally

```

```

≡ file-test0.txt
≡ file-test1.txt
≡ file-test2.txt
≡ file-test2a.txt
≡ file-test3.txt

```

Test2a.txt:

```

exception.cc  ≡ file-test2a.txt ×
project5 > Problem > code > build.linux > ≡ file-test2a.txt
1  Hello from file-test2
2  Hello from file-test2
3  Hello from file-test2
4  Hello from file-test2
5  Hello from file-test2
6

```

From the code reading of the program, the following file system calls are called:

Create(): The program calls Create("file-test3.txt") to create file filetest3.txt.

```

if (Create("file-test3.txt") == -1) {
    Exit(1);
}

```

Open(): The program calls Open("file-test3.txt") to open file file-test3.txt.

```
outfile0 = Open("file-test3.txt");
```

Write():

```
for (i = 0; i < 5; i++)
{
    numWritten = Write(str, 22, outfile0);
    Write("Number of Characters Written is ", 32, ConsoleOutput);
    length = toString(&num2stringBuffer, numWritten);
    Write(&num2stringBuffer, length, ConsoleOutput);
    Write("\n", 1, ConsoleOutput);
}
```

Write(str, 22, outfile0) to write a line "Hello from file-test3" to file file-test3.txt.

Write("Number of Characters Written is ", 32, ConsoleOutput) to write "Number of Characters Written is " to the standard output stream which is terminal here.

Write(&num2stringBuffer, length, ConsoleOutput) to write the size of string buffer have written into the file to the standard output stream which is terminal here.

Write("\n", 1, ConsoleOutput) to write the a '\n' char to the standard output stream.

Close(): The program calls Close(outfile0) to close the file file-test3.txt.

After the analysis of the program, it should output the "Number of Characters Written is 22" five time which are related to the file system call. A file file-test3.txt will be created with 5 line of "Hello from file-test3".

```
zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/file-test3
Number of Characters Written is 22
Number of Characters Written is 22
Number of Characters Written is 22
Number of Characters Written is 22
Number of Characters Written is 22
Process ../test2/file-test3 exited normally
^C
Cleaning up after signal 2
```

```
= exception.o
≡ file-test0.txt
≡ file-test1.txt
≡ file-test2.txt
≡ file-test2a.txt
≡ file-test3.txt
= filehdr.o
```



```
exception.cc  file-test3.txt X
project5 > Problem > code > build.linux > file-test3.txt
1 Hello from file-test3
2 Hello from file-test3
3 Hello from file-test3
4 Hello from file-test3
5 Hello from file-test3
6
```

Question2:

Test-file4.c: code

```
#include "syscall.h"

int
main()
{
    OpenFileId input;
    char buffer[100];
    int i,j;
    input = Open("file-test0.txt");

    for (i = 0; i < 2; i++)
    {
        Read(&buffer, 22, input);
        Write(buffer, 22, ConsoleOutput);
        for (j = 0; j < 10000; j++);
    }
    if(Close(input)){
        Write("close correctly\n",22,ConsoleOutput);
    }else{
        Write("close wrongly \n",22,ConsoleOutput);
    }
    Exit(0);
}
```

```

#include "syscall.h"

int
main()
{
    int OpenFileId = input;
    char buffer[100];
    int i, j;
    input = Open("file-test0.txt");

    for (i = 0; i < 2; i++)
    {
        Read(&buffer, 22, input);
        Write(buffer, 22, ConsoleOutput);
        for (j = 0; j < 10000; j++);
    }
    if (Close(input)) {
        Write("close correctly\n", 22, ConsoleOutput);
    } else {
        Write("close wrongly\n", 22, ConsoleOutput);
    }
    Exit(0);
}

```

In this program, it will use `Open("file-test0.txt")` to existed file `file-test0.txt`. In the loop(only 2 lines data), each iteration will read one line of the file with maximum size of 22 by calling `Read()` and print out the content to the console. Then it calls `Close()` to close the file. After that, the output will depend on the return value of `Read()`.

```

C file-test4.c  file-test0.txt  Makefile
project5 > Problem > code > build.linux > file-test0.txt
4 Hello from file-test0
5 Hello from file-test0
6

```

```

zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/file-test4
Hello from file-test0
Hello from file-test0
close correctly
Process ../test2/file-test4 exited normally
^C
Cleaning up after signal 2
zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ █

```

The reason we can not use `printf()` is that we didn't and can't include the standard I/O library which is "" when the compilation is dependent on nachos instead of Unix. So the program has to call the I/O system call in nachos to achieve the I/O operations, because the program only includes the "syscall.h".

Q7.2

For prog3.c, this program calls `Exec()` for prog4.c. Then it prints the child process id. Finally `Exit()` to finish itself with value 0.

For prog3b.c, this program calls `Exec()` for prog4.c. Then it calls `Join()` to wait for the child process until the given child process is finish. If the `Join()` return 0, it prints the child process id. Finally `Exit()` to finish itself with value 0.

For prog4.c, this program calls `Write()` to print Hello from prog4 .Finally `Exit()` to finish itself with value 0.

The running result of prog3:

```

zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/prog3
Hello from prog3. Child process id: 1
Process ../test2/prog3 exited normally
Hello from prog4
Process ../test2/prog4 exited normally
█

```

In nachos, non-preemptive round-robin is the default scheduler. Thus the parent processor will run first and print Hello from prog3. Child process id: 1 then exit. The child processor runs until the parent processor finishes. Then the child process prog4 will print hello from prog4 and exits normally.

Prog3b test output:

```

zren08@lcs-vc-cis486-2:~/project5/Problem/code/build.linux$ ./nachos -x ../test2/prog3b
Hello from prog4
Process ../test2/prog4 exited normally
Hello from prog3b. Child process id: 1
Process ../test2/prog3b exited normally
^C
Cleaning up after signal 2
█

```

The output is shown in the screenshot below. prog3b.c calls `Exec()` to create a child process for prog4.c. Because the parent process calls `Join()`, the parent process which is prog3b.c has to wait for that prog4.c finished and get the exit status. Thus in the output, "Hello from prog4" is printed first. "Hello from prog3b" is printed until prog4 is finished which means that "Process

../test/prog4 exited normally". Then do the program3b's (parents) work. Hello from prog3b output the child processid, and exit normally.