Academic Report Cover Page

CIS 657

OS Project 4

Ameya Kale
SU ID: 3049670941
Email Address: akkale@svr.edu

December 2, 2022

Learning Objectives

After completing this project assignment, students can

1. describe how system call exceptions are made and handled in Nachos,
2. describe how virtual to physical address translation is performed in the Nachos Paging mech- anism.
3. describe and implement the Write (· · ·) system call.
4. describe and use Machine::ReadMem(int addr, int size, int *value) to read from Nachos emulated RAM. This function call performs virtual to physical address translation as well.
5. implement Exit(·) system call.

Question 1

Ans:

Among the 11 steps in the slide, step 6 is equivalent to calling the void ExceptionHandler (ExceptionType which) in exception.cc file. In this step a trap to the kernel is made. From the comments in the exception.cc, we learned that this file is the entry point into the Nachos Kernel.

Question 2

Ans:

Among the 11 steps in the slide, steps 7, 8, 9, 10 and 11 are implemented in the exception.cc file. The above steps are made in the kernel space and the execption.cc file implements system calls in the kernel space.

Question 3

Ans:

The function that is used to access the parameter values in ExceptionHandler() is called readRegister(). The argument passed to this function is number of the register that we need to read the value from.

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

Question 4

Ans:

The file that defines OpenFileId and ConsoleOutput is syscall.h in the userprog directory.

Evidence of definition of OpenFileId and ConsoleOutput in syscall.h file:

```
 99   /* A unique identifier for an open Nachos file. */
100   typedef int OpenFileId;
101
102   /* when an address space starts up, it has two open files, representing
103    * keyboard input and display output (in UNIX terms, stdin and stdout).
104    * Read and Write can be used directly on these, without first opening
105    * the console device.
106    */
107
108   #define ConsoleInput    0
109   #define ConsoleOutput   1
```

Grep output:

OpenFileId

```
akkale@lcs-vc-cis486-2:~/Problem/code$ grep -r "OpenFileId" userprog
userprog/syscall.h:typedef int OpenFileId;
userprog/syscall.h:/* Open the Nachos file "name", and return an "OpenFileId" that can
userprog/syscall.h:OpenFileId Open(char *name);
userprog/syscall.h:int Write(char *buffer, int size, OpenFileId id);
userprog/syscall.h:int Read(char *buffer, int size, OpenFileId id);
userprog/syscall.h:int Seek(int position, OpenFileId id);
userprog/syscall.h:int Close(OpenFileId id);
```

ConsoleOutput:

```
akkale@lcs-vc-cis486-2:~/Problem/code$ grep -r "ConsoleOutput" userprog
userprog/synchconsole.h:class SynchConsoleOutput : public CallBackObj {
userprog/synchconsole.h:    SynchConsoleOutput(char *outputFile); // Initialize the console device
userprog/synchconsole.h:    ~SynchConsoleOutput();
userprog/synchconsole.h:    ConsoleOutput *consoleOutput;// the hardware display
userprog/syscall.h:#define ConsoleOutput       1
userprog/synchconsole.cc:// SynchConsoleOutput::SynchConsoleOutput
userprog/synchconsole.cc:SynchConsoleOutput::SynchConsoleOutput(char *outputFile)
userprog/synchconsole.cc:    consoleOutput = new ConsoleOutput(outputFile, this);
userprog/synchconsole.cc:// SynchConsoleOutput::~SynchConsoleOutput
userprog/synchconsole.cc:SynchConsoleOutput::~SynchConsoleOutput()
userprog/synchconsole.cc:// SynchConsoleOutput::PutChar
userprog/synchconsole.cc:SynchConsoleOutput::PutChar(char ch)
userprog/synchconsole.cc:// SynchConsoleOutput::CallBack
userprog/synchconsole.cc:SynchConsoleOutput::CallBack()
```

Question 5

Ans:

- The Write() system call has three parameters passed to it: buffer, size and the file id.

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

- The buffer is of type char, size is of type int and the file id is of typedef int.
- The size denotes total number of bytes in the file.
- The write() system call reads these size number of bytes from the file and stores them into a buffer.
- The call will return the number of bytes read.
- It will read whatever is available, if the file size is small.

Evidence of explanation of each parameter of Write() system call in syscall.h:

```
128  int Write(char *buffer, int size, OpenFileId id);
129
130  /* Read "size" bytes from the open file into "buffer".
131   * Return the number of bytes actually read -- if the open file isn't
132   * long enough, or if it is an I/O device, and there aren't enough
133   * characters to read, return whatever is available (for I/O devices,
134   * you should always wait until you can return at least one character).
135   */
```

Question 6

Ans:

Code Snippet:

```
82      case SC_Write:
83        printf("Write system call made by %s\n", kernel->currentThread->getName());
84        printf("In ExceptionHandler, Write() System Call is made. The first parameter is %d, the second
          parameter is %d, and the third parameter is %d \n", kernel->machine->ReadRegister(4),
          kernel->machine->ReadRegister(5), kernel->machine->ReadRegister(6));
85        printf("\n");
86        break;
```

Explanation:

- From the explanation in exception.cc, we learned that the parameters are stored in the registers r4, r5 and r6. Therefore, we pass 4, 5 and 6 respectively to the readRegister() function to read the respective values.
- We print the values in the format specified in the question using the printf function.
- While printing the values, we use %d format specifier to treat the values as integers.

Code Output:

```
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/prog1
The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Exit system call made by ../test1/prog1
```

Output Explanation:
- The first line prints the message “Write system call made by ../test1/prog1”.
- ../test1/prog1 is the name of the thread that made the system call.
- The next line is the message that we coded. The message follows the format specified in the question to print the parameter values.
- The first parameter value is 496, second parameter value is 18 and the third parameter value is 1.
- These two messages are printed 5 times as the Write() system call is called in a loop that runs for 5 times. This is coded in the prod1.cc file in test1 directory.
  - Evidence:

```
12    int i,j;
13    for (i = 0; i < 5; i++)
14    {
15      Write(str, 18, output);
16      for (j = 0; j < 10000; j++);
17    }
18
19    Exit(0);
```

Question 7

Ans:

Code Snippet:

```
82        case SC_Write:
83          printf("Write system call made by %s\n", kernel->currentThread->getName());
84          printf("In ExceptionHandler, Write() System Call is made. The first parameter is %d, the second
            parameter is %d, and the third parameter is %d \n", kernel->machine->ReadRegister(4),
            kernel->machine->ReadRegister(5), kernel->machine->ReadRegister(6));
85          printf("\n");
86          break;
```

Explanation:

- From the explanation in exception.cc, we learned that the parameters are stored in the registers r4, r5 and r6. Therefore, we pass 4, 5 and 6 respectively to the readRegister() function to read the respective values.
- We print the values in the format specified in the question using the printf function.
- While printing the values, we use %d format specifier to treat the values as integers.

Code Output:

```
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/prog1 -x ../test1/prog2
Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Exit system call made by ../test1/prog1
Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Exit system call made by ../test1/prog2
```

Output Explanation:

1) The value of the first parameters for prog1 and prog2 are the same, i.e. 496.
2) Yes, these values are the same.
3) They are the virtual address of str defined in each program.
4) The first value, i.e. 496 is the address of the str, second value, 18 is the size of the str and the third value, 1 is the file id.

Question 8

Ans:

- addr: addr is the virtual address from where the read operation will take place
- size: The size parameter denotes the number of bytes to be read from addr. The size parameter can take values 1, 2 or 4.
- value: After the read operation, the value parameter stores the data. It is where the write operation is performed.

Question 9

Ans:

- All instructions are 4 bytes long.
- Therefore, in order to point to the next instruction the PC will need to jump 4 bytes forward.
- Hence, we need the code to increment the program counter by 4.

Question 10:

Ans:

1) Code for Write() system call:

```
82          case SC_Write:
83          {
84            int str, size, fileId;
85            str = kernel->machine->ReadRegister(4);
86            size = kernel->machine->ReadRegister(5);
87            fileId = kernel->machine->ReadRegister(6);
88
89            char arr[size];
90
91            for(int i = 0; i < size; i++){
92              int buff;
93              kernel->machine->ReadMem(str + i, 1, &buff);
94              arr[i] = buff;
95            }
96
97            printf("The value of the buffer array is: ");
98            for(int i = 0; i < size; i++){
99              printf("%c", arr[i]);
100           }
101
102           printf("\n");
103
104           printf("Write system call made by %s\n", kernel->currentThread->getName());
105           printf("In ExceptionHandler, Write() System Call is made. The first parameter is %d, the second parameter is
              %d, and the third parameter is %d \n", str, size, fileId);
106           printf("\n");
107
108           kernel->machine->WriteRegister(2, size);
109           break;
110         }
```

Code Explanation:

- We store the three parameter values, address of str, size and the file id of the write system call() in variables in str, size and fileId respectively.
- Then, we will declare a character array arr with the same size specified by size variable.
- Then, we will iterate a loop for size number of times.
- In this loop, we will write the content of the file into a buffer variable buff.

- Then, we will store the content of that buffer variable into the ith location of the array arr.
- We will then print all the elements in the array.
- Finally the writeRegister() call will finish writing the contents to the registers when the write operation is finished.

<u>Output for ./nachos -x ../test1/prog1 -x ../test1/prog2</u>:

```
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/prog1 -x ../test1/prog2
The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog1

Write system call made by ../test1/prog1
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Exit system call made by ../test1/prog1
Process ../test1/prog1 exited normallyThe value of the buffer array is: Hello from prog2

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog2

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog2

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog2

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

The value of the buffer array is: Hello from prog2

Write system call made by ../test1/prog2
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 18, and the third parameter is 1

Exit system call made by ../test1/prog2
^CProcess ../test1/prog2 exited normally
Cleaning up after signal 2
```

<u>Output Explanation</u>:
- The first line is the buffer content value that read from the file prog1.
- Then, the write system call will be made by the file and then the three parameters will be printed similar to what we saw in the question 6 answer.

- The messages will be printed 5 times in a loop as defined in the prog1.c file.
- Similarly, the output for file prog2.c will be generated.

2) Code in myProg.c

code > test1 > C myProg.c > main()

```c
#include "syscall.h"

char data[2048];

int
main()
{
  OpenFileId output = ConsoleOutput;

  char* str = "My name is Ameya Kale and today's date is 12/02/2022\n";

  //int len = sizeof(str);
  //print("Length = %d", len);

  int i,j;
  for (i = 0; i < 10; i++)
  {
    Write(str, 56, output);
    for (j = 0; j < 10000; j++);
  }

  Exit(0);
}
```

Code Explanation:

- We can create the custom user program similar to prog1 and prog2 files.
- We will name this file as myProg.c
- We will modify the str value to display my name and the date.
- In the for loop we will loop for 10 times and call the write() system call.
- The size parameter will have the value 56, i.e. the total number of characters in str.

Output:

```
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/myProg
The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

The value of the buffer array is: My name is Ameya Kale and today's date is 12/02/2022

Write system call made by ../test1/myProg
In ExceptionHandler, Write() System Call is made. The first parameter is 496, the second parameter is 56, and the third parameter is 1

Exit system call made by ../test1/myProg
^CProcess ../test1/myProg exited normally
Cleaning up after signal 2
```

<u>Output Explanation</u>:

- The first line the buffer content value that read from the file myProg.c
- The line prints my name Ameya Kale and the date 12/02/2022.
- After that the write system call message and the parameter values are printed.
- This is looped 10 times as stated in the myProg.c file.

Question 11

Ans:

<u>Code for Exit() system call</u>:

```
115         case SC_Exit:
116         {
117           printf("Exit system call made by %s\n", kernel->currentThread->getName());
118
119           int getStatus;
120           getStatus = kernel->machine->ReadRegister(4);
121
122           if(getStatus == 0){
123             printf("Process %s exited normally", kernel->currentThread->getName());
124           }
125           else{
126             printf("Process %s exited abnormally", kernel->currentThread->getName());
127           }
128
129           kernel->currentThread->Finish();
130           break;
131         }
132         default:
133           cerr << "Unexpected system call " << type << "\n";
134           break;
135       }
136       break;
137     default:
138       cerr << "Unexpected user mode exception" << (int)which << "\n";
139       break;
140   }
```

Code Explanation:

- We will first store the value of register 4 in a variable getStatus.
- If the value of getStatus is 0, then we will print the message “Process x exited normally”.
- Else, we will print the message “Process x exited abnormally”.
- Here, x is the name of the process.

code > test1 > C exit-test0.c > main()

```
1  #include "syscall.h"
2
3  char data[2048];
4
5  int main()
6  {
7    Exit(0);
8  }
```

code > test1 > C exit-test1.c > main()

```
1  #include "syscall.h"
2
3  char data[2048];
4
5  int main()
6  {
7    Exit(1);
8  }
```

Code:

Code Explanation:

- We will create two new files exit-test0.c and exit-test1.c
- Inside the file exit-test0 in the main function, we will simply return 0 to the Exit system call.
- And in the file exit-test1 in the main function, we will simply return 1 to the Exit system call.

Output:

```
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/exit-test0
Exit system call made by ../test1/exit-test0
^CProcess ../test1/exit-test0 exited normally
Cleaning up after signal 2
akkale@lcs-vc-cis486-2:~/Problem/code/build.linux$ ./nachos -x ../test1/exit-test1
Exit system call made by ../test1/exit-test1
^CProcess ../test1/exit-test1 exited abnormally
Cleaning up after signal 2
```

Output Explanation:

- The exit-test0.c file passes value 0 to the Exit system call.
- If the register value in register 4, then the process will exit normally, else if the value is 1, it will exit abnormally.
- The exit-test1.c file passes value 1 to the Exit system call.

Conclusion:

The code was run successfully and all the questions were answered successfully.