Nachos System Calls and Paging

1 An Important Reminder

Review the university's academic integrity policy (http://class.syr.edu/academic-integrity). Remember that violating academic integrity policy will significantly jeopardize your grade.

2 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 mechanism.
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.

3 Preliminary Requirement

Download "`Project4.tar.gz`" from Bb. Decompress it using the command, "`tar -xvf Project4.tar.gz`". A directory called "Problem" will be created. All nachos files/directories will be created under "Problem."

4 Understanding how system calls are made

Refer to Page 43 ("Lecture Slides: Chapter2-Processes-and-threads-p1"). Read the comments at the beginning of `exception.cc`, from line 1 to line 50.

Question 1 *(E: 2 points) Among the 11 steps in the slide, which step is equivalent to calling* `void ExceptionHandler(ExceptionType which)` *in* `exception.cc`*? Explain.*

Question 2 *(E: 3 points) Among the 11 steps in the slide, which steps are implemented in* `exception.cc`*? Explain.*

Question 3 *(E: 5 points) Read the documentation in* `exception.cc` *to answer this question. The system call code and the user program's parameter values making the call are passed to the kernel, specifically to* `void ExceptionHandler(ExceptionType which)`*. What is the name of the function you can use to access these parameter values in* `ExceptionHandler()` *?*

5 Accessing Program Address Space from Kernel

```
#include "syscall.h"
char data[2048];
int main()
{
  OpenFileId output = ConsoleOutput;
  char* str = "Hello from prog1\n";
  int i, j;
  for (i = 0; i < 5; i++) {
    Write(str, 18, output);
    for (j = 0; j < 10000; j++);
  }
Exit(0);
}
}
```

Question 4 *(E: 5 points) The above is* `write.c` *in* `test1` *directory. Read* `write.c` *carefully and fully understand each line of the code. Find the file that defines* `OpenFileId` *and* `ConsoleOutput` *in* `userprog` *directory and explain how* `OpenFileId` *and* `ConsoleOutput` *are defined. Show the evidence of your investigation by using* `grep` *and by submitting a screenshot of the relevant part of the code.*

Question 5 *(E: 5pts) Pay attention to the line,* `Write(str, 18, output)`*. As we mentioned in the previous projects, the current implementation of the* `Write()` *system call in* `exception.cc` *is not complete; i.e., it does not print out the* `str` *buffer's content passed from* `write.c`*. In the source file you identified in Question 4, find the explanations on each parameter of the* `Write(...)` *system call. And explain* `Write(str, 18, output)`*. Show the evidence of your investigation by submitting a screenshot of the relevant part of the code.*

Question 6 *(M: 10 points) Insert the source codes for accessing each parameter value of* `Write(str, 18, output)` *in the* `SC_Write` *part of* `ExceptionHandler(·)`*, compile Nachos, and print out the values of the parameters in integer types. The printed message should be "*`in ExceptionHandler, Write() System Call is made. The first parameter is` x`, the second parameter is` y`, and the third parameter is` z*," where x, y, and z are the values of the parameters in integers. Screenshot of the code snippet and output of the program are needed.*

Question 7 *(E: 5 points) The first parameter of* `Write(str, 18, output)` *is the address of the string buffer,* `str`*, defined in* `write.c`*. Run* `./nachos -x ../test1/prog1 -x ../test1/prog2` *command with the changes you made in Question 6. (1) What are the values of the first parameters printed out for* `prog1` *and* `prog2` *in* `ExceptionHandler(···)`*? (2) Are these values the same or different? (3) Are they virtual addresses or physical addresses of* `str` *defined in each program? (4) Explain about these values in your answer. Screenshot of the code snippet and output of the program are needed.*

Question 8 *(M: 10 points) Read* `bool Machine::ReadMem(int addr, int size, int *value)` *in* `machine/translate.cc` *carefully. This function can be called when you want to read from the RAM when you have a virtual address. The function will call* `Translate(addr, &physicalAddress, size, FALSE)` *function to translate a given virtual address to the corresponding physical address. Explain the purpose of each parameter in* `ReadMem(int addr, int size, int *value)`*.*

Question 9 *(E: 5 points) At the end of* `exception.cc`*, there is a line that increments the Program Counter. Why do we need this code? The increment is done by adding* 4*. Why* 4*?*

6 Implementing `Write()` and `Exit` System Calls

Hint: You can use `Machine::ReadRegister()` to read from registers, `Machine::WriteRegister()` to write to registers, and `Machine::ReadMem()` to read from a virtual memory address. Recall that `Machine::ReadMem()` performs virtual to physical translation using `Machine::Translate()`.

6.1 `Write()` System Call (M: 15 points)

`Write(···)` system call writes a string to a specified destination. Depending on the third parameter value, a `Write(···)` call either write the content of the buffer to the Console (i.e., to the screen) or to a file. In this project, we are not implementing the case for writing to a file; therefore it always writes to the screen (i.e., `ConsoleOutput`).

Implement the `Write()` system call by completing the `SC_Write` case in `void ExceptionHandler(·)` so that Nachos will be able to print out the content of the buffer passed by a user program. For example, `./nachos -x ../test1/prog1` will printout the buffer content `"Hello from prog1"`. Note

that you cannot hard code the message. (Attach a screenshot of the `SC_Write` implementation). Whatever buffer content is passed by a user program through a `Write()` call, your Nachos implementation must be able to print the buffer content. Refer to the comments in `exception.cc` and the code investigation part for hints on how to obtain the parameter values. Read `syscall.h` carefully for the definition of system calls and required return values for each system call, including for `Write()`.

6.2 `void Exit(int status)` System Call (E: 10 points)

For this project, `void Exit(int status)` can be implemented with a simple call to `Thread::Finish()`. If `status` is `0`, print out "`Process` x `exited normally`"; if `status` is `1`, print out "`Process` x `exited abnormally`," where x is the program name. Note that you cannot hard code the message. (Attach a screenshot of the `SC_Exit` implementation).

7 Tests and Output

Provide screenshots of the test cases you run and explain the results of each test case.

Question 10 *(M: 15 points)*

1. *Test your program with "*`./nachos -x ../test1/prog1 -x ../test1/prog2`*".*
2. *Write a user program that calls* `Write(···)` *system call and print out your name and the date for 10 times. The date can be hardcoded as a part of the string. Test your program with this new user program.*

Provide screenshots of the test cases you run and explain the results of each test case. Provide screenshots of your test program source code

Question 11 *(M: 10 points) Design two user programs that can test your implementation of* `Exit(int status)` *system call, one for status 0 and the other for status 1. Name them* `exit-test0.c` *and* `exit-test1.c`*, respectively. Provide screenshots of the test cases you run and explain the results of each test case. Provide screenshots of your test program source code*

8 What to Submit

You must submit two files to Bb: (1) The PDF report and (2) The source code zip file.

1. **PDF report:** You need to submit a detailed project report describing what you have done to implement the requirements and your outputs. Answer all questions. Please also include the important code snippets (screenshots) followed by an explanation. These code snippets

should include the modified part of the Nachos source code, whatever code you have written, including test programs. Simply attaching code without any description will not receive credits. Include screenshots whenever necessary, for example, to show your program outputs and explain the outputs. In summary, you need to make your best effort to convince the grader that your submission satisfies all requirements for this project, without the TAs look through or running your program.

2. **Source code zip file:** After "`make clean`" in `build.linux`, please compress your whole `nachos` folder, name the compressed file LastName,FirstName.zip (NOTE: .zip ONLY!)