PA-4

Name: Raghunandan Gajanan Bhat

SUNetID: rgbhat@syr.edu

Task-1: Revisiting system calls in Nachos

✪ **[R-1]:** `Read()` system call takes 3 parameters and returns an integer.

- `char *buffer` – is the placeholder for data read from the open file
- `int size` – size of bytes to be read from the open file
- `OpenFileID id` – is the open file from which we have to read

`Read()` system call returns the number of bytes it actually read from the open file.

```
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  */
136 int Read(char *buffer, int size, OpenFileId id);
137
```

✪ **[R-2]:** Register `r2`($2) is used to hold the system call number. arg1 will be in register `r4`, arg2 will be in register `r5` and arg3 will be in register `r6`. The return value is stored back in register `r2`.

✪ **[R-3]:** `ReadRegister()` is the function used to access the values stored in registers.

Task-2: Accessing Address space from Nachos Kernel

✪ **[R-1]:** `OpenFileId` and `ConsoleOutput` are defined in `userprog/syscall.h` file.

`grep`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/userprog$ grep 'OpenFileId' ./*
./syscall.h:typedef int OpenFileId;
./syscall.h:/* Open the Nachos file "name", and return an "OpenFileId" that can
./syscall.h:OpenFileId Open(char *name);
./syscall.h:int Write(char *buffer, int size, OpenFileId id);
./syscall.h:int Read(char *buffer, int size, OpenFileId id);
./syscall.h:int Seek(int position, OpenFileId id);
./syscall.h:int Close(OpenFileId id);
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/userprog$
```

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/userprog$ grep 'ConsoleOutput' ./*
./synchconsole.cc:// SynchConsoleOutput::SynchConsoleOutput
./synchconsole.cc:SynchConsoleOutput::SynchConsoleOutput(char *outputFile)
./synchconsole.cc:    consoleOutput = new ConsoleOutput(outputFile, this);
./synchconsole.cc:// SynchConsoleOutput::~SynchConsoleOutput
./synchconsole.cc:SynchConsoleOutput::~SynchConsoleOutput()
./synchconsole.cc:// SynchConsoleOutput::PutChar
./synchconsole.cc:SynchConsoleOutput::PutChar(char ch)
./synchconsole.cc:// SynchConsoleOutput::CallBack
./synchconsole.cc:SynchConsoleOutput::CallBack()
./synchconsole.h:class SynchConsoleOutput : public CallBackObj {
./synchconsole.h:    SynchConsoleOutput(char *outputFile); // Initialize the console device
./synchconsole.h:    ~SynchConsoleOutput();
./synchconsole.h:    ConsoleOutput *consoleOutput;// the hardware display
./syscall.h:#define ConsoleOutput       1
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/userprog$
```

`OpenFileId`

```
 89
 90 /* File system operations: Create, Remove, Open, Read, Write, Close
 91  * These functions are patterned after UNIX -- files represent
 92  * both files *and* hardware I/O devices.
 93  *
 94  * Note that the Nachos file system has a stub implementation, which
 95  * can be used to support these system calls if the regular Nachos
 96  * file system has not been implemented.
 97  */
 98
 99 /* A unique identifier for an open Nachos file. */
100 typedef int OpenFileId;
101
```

`ConsoleOutput`

```
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
110
```

✪ **[R-2]:** When `Write(char *buffer, int size, OpenFileId id)` is executed, the `Write()` system call writes 'size' bytes from output buffer to the open file. In `Wirte(str, 17, output)`, 17 bytes from `'str'` are written into open file `'output'`.

```
123
124 /* Write "size" bytes from "buffer" to the open file.
125  * Return the number of bytes actually written, if successful.
126  * On failure, a negative error code is returned.
127  */
128 int Write(char *buffer, int size, OpenFileId id);
129
```

✪ **[R-3]:** Edits to the `userprog/execption.cc`

```
103
104         case SC_Write:
105           //printf("Write system call made by %s\n", kernel->currentThread->getName());
106           printf("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));
107
108
109           break;
110
```

✪ **[R-4]:** `./nachos -x ../test-pa/write`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$ ./nachos -x ../test-pa/write
The first parameter is 496, the second parameter is 17, the third parameter is 1
The first parameter is 496, the second parameter is 17, the third parameter is 1
The first parameter is 496, the second parameter is 17, the third parameter is 1
The first parameter is 496, the second parameter is 17, the third parameter is 1
The first parameter is 496, the second parameter is 17, the third parameter is 1
Exit system call made by main
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$
```

✪ **[R-5]:** `bool ReadMem(int addr, int size, int *value)`

- addr – the virtual address to read from
- size – the number of bytes to read, it can either be 1, 2 or 4 bytes
- value – placeholder to write the result, i.e the value read from virtual address

✪ **[R-6]:** The increment of program counter is needed to point to the next instruction to be executed. Since each instruction is 4 bytes wide, we will increment the PC by 4 to point to the next instruction.

Task-3 : `Write()` system call implementation

✪ **[R-1]:** Implementation of `Write()`

```
111
112         case SC_Write:
113           {
114                 //printf("Write system call made by %s\n", kernel->currentThread->getName());
115                 //printf("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));
116                 //read parameters to the write system call
117                 int source = (int)kernel->machine->ReadRegister(4);
118                 int size = (int)kernel->machine->ReadRegister(5);
119                 int dest = (int)kernel->machine->ReadRegister(6);
120                 int byte_count = 0;
121                 //read each memeory location and write the contents of the memory location
122                 for(int i = 0; i < size; i++){
123                         if(kernel->machine->ReadMem(source, 1, &dest)){
124                                 printf("%c", dest);
125                                 source = source + 1;
126                                 byte_count++;
127                         }
128                 }
129                 //write the number of bytes written to console output to register 2
130                 kernel->machine->WriteRegister(2, (int)byte_count);
131           }
132           break;
133
```

✪ **[R-2]:** `./nachos -x ../test-pa/write`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$ ./nachos -x ../test-pa/write
Hello from prog1
Hello from prog1
Hello from prog1
Hello from prog1
Hello from prog1
Exit system call made by main
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$
```

✪ **[R-3]:** `./nachos -x ../test-pa/prog2`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$ ./nachos -x ../test-pa/prog2
Hello from prog2
Hello from prog2
Hello from prog2
Hello from prog2
Hello from prog2
Exit system call made by main
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$
```

Task 4 – Implementation of Exit() system call

✪ **[R-1]:** Implementation of `Exit()`

```
83
84          case SC_Exit:
85          {
86            printf("Exit system call made by %s\n", kernel->currentThread->getName());
87            //arg1 stores exit status value and arg1 is in register r4
88            int exit_status = kernel->machine->ReadRegister(4);
89            if(exit_status == 0){
90                  printf("Process exited normally\n");
91            }else{
92                  printf("Process exited abnormally with status %d\n", exit_status);
93            }
94            // Let's finish this thread
95            kernel->currentThread->Finish();
96          }
97            break;
98
```

✪ **[R-2]:** `./nachos -x ../test-pa/exit-test0`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$ ./nachos -x ../test-pa/exit-test0
Hello from exit0
Exit system call made by main
Process exited normally
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$
```

✪ **[R-3]:** `./nachos -x ../test-pa/exit-test1`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$ ./nachos -x ../test-pa/exit-test1
Hello from exit1
Exit system call made by main
Process exited abnormally with status 1
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa4/student/nachos/code/build.linux$
```