PA-6

Name: Raghunandan Gajanan Bhat

SUNetID: rgbhat@syr.edu

Task-1: System calls for File I/O

✪ **[R-1]:**

Create system call: `int Create(char *name)`

```
148
149
150         case SC_Create:
151           //printf("Create() system call is called \n");
152         {
153                 int start = (int)kernel->machine->ReadRegister(4);
154                 char filename[60];
155                 get_parameter(filename, start);
156                 int create_status = open(filename, O_CREAT|O_RDWR|O_TRUNC, S_IRWXU);
157                 if(create_status == -1){
158                         kernel->machine->WriteRegister(2, -1);
159                 }else{
160                         kernel->machine->WriteRegister(2, 1);
161                 }
162         }
163           break;
164
165
```

Open system call: `OpenFileId Open(char *name)`

```
165
166
167          case SC_Open:
168            //printf("Open() system call is called\n");
169          {
170                  int start = (int)kernel->machine->ReadRegister(4);
171                  char filename[60];
172                  get_parameter(filename, start);
173                  int fd = open(filename, O_RDWR);
174                  if(fd == -1){
175                          kernel->machine->WriteRegister(2, -1);
176                  }else{
177                          kernel->machine->WriteRegister(2, (int)fd);
178                  }
179          }
180            break;
181
```

Write system call: `int Write(char *buffer, int size, OpenFileId id)`

```
132
133
134         case SC_Write:
135           //printf("Write system call made by %s\n", kernel->currentThread->getName());
136         {
137                 int buf_addrs = (int)kernel->machine->ReadRegister(4);
138                 int size = (int)kernel->machine->ReadRegister(5);
139                 int dest_addrs = (int)kernel->machine->ReadRegister(6);
140                 //cout<<"Write address: "<<buf_addrs<<endl;
141
142                 int byte_count = SysWrite(buf_addrs, size, dest_addrs);
143                 kernel->machine->WriteRegister(2, byte_count);
144         }
145
146           break;
147
```

`SysWrite()` **function**

```
52
53 //----------------------------------------------------------------------
54 // Write System call
55 // Read 'size' amount of bytes from 'buf_addrs' and
56 //write it to 'dest_addrs' and return the number of
57 //bytes actually written to 'dest_buf'
58
59 int SysWrite(int buf_addrs, int size, int dest_addr){
60         int i = -1;
61         char* content = (char*)malloc((size+2) * sizeof(char));
62
63         int len = get_parameter(content, buf_addrs);
64         if(dest_addr == ConsoleOutput){
65                 if(len < 0){
66                         return -1;
67                 }else{
68                         cout << content;
69                         return len;
70                 }
71         }else{
72                 int byte_count = write(dest_addr, content, size);
73                 if(byte_count < 0){
74                         return -1;
75                 }else{
76                         return byte_count;
77                 }
78         }
79         free(content);
80         return i;
81 }
82
```

Read system call: `int Read(char *buffer, int size, OpenFileId id)`

```
109           */
110         case SC_Read:
111           //printf("Read system call made by %s\n", kernel->currentThread->getName());
112         {
113                 int buffer = (int)kernel->machine->ReadRegister(4);
114                 int size = (int)kernel->machine->ReadRegister(5);
115                 int file_id = (int)kernel->machine->ReadRegister(6);
116
117                 char* unix_buf = (char*)malloc((size+2) * sizeof(char));
118                 int bytes_read = read(file_id, (char *)unix_buf, size);
119
120                 if(bytes_read < 0){
121                         kernel->machine->WriteRegister(2, -1);
122                 }else{
123                         for(int i = 0; i< bytes_read; i++){
124                                 kernel->machine->WriteMem(buffer+i, 1, unix_buf[i]);
125                         }
126                         kernel->machine->WriteRegister(2, (int)bytes_read);
127                 }
128
129                 free(unix_buf);
130         }
131           break;
132
```

Close system call: `int Close(OpenFileId id)`

```
184
185         case SC_Close:
186           //printf("Close() system call is called\n");
187         {
188                 int fd = (int)kernel->machine->ReadRegister(4);
189                 if(close(fd) == 0){
190                         kernel->machine->WriteRegister(2, 0);
191                 }else{
192                         kernel->machine->WriteRegister(2, -1);
193                 }
194         }
195           break;
196
```

Helper function to extract filename: `void get_parameter(char *filename, int start_addr)`

```
16
17 //-------------------------------------------------------------------------
18 //get_parameter
19 //given the start address of the parameter, extract the name of the
20 //parameter/filename
21
22 int get_parameter(char *filename, int start_addr){
23         int ch, i = 0;
24         if(kernel->machine->ReadMem(start_addr, 1, &ch)){
25                 while((char(ch) != '\0')){
26                         filename[i] = (char)ch;
27                         i++;
28                         start_addr++;
29                         kernel->machine->ReadMem(start_addr, 1, &ch);
30                 }
31                 filename[i] = '\0';
32                 return i;
33         }else{
34                 return -1;
35         }
36         return -1;
37 }
38
```

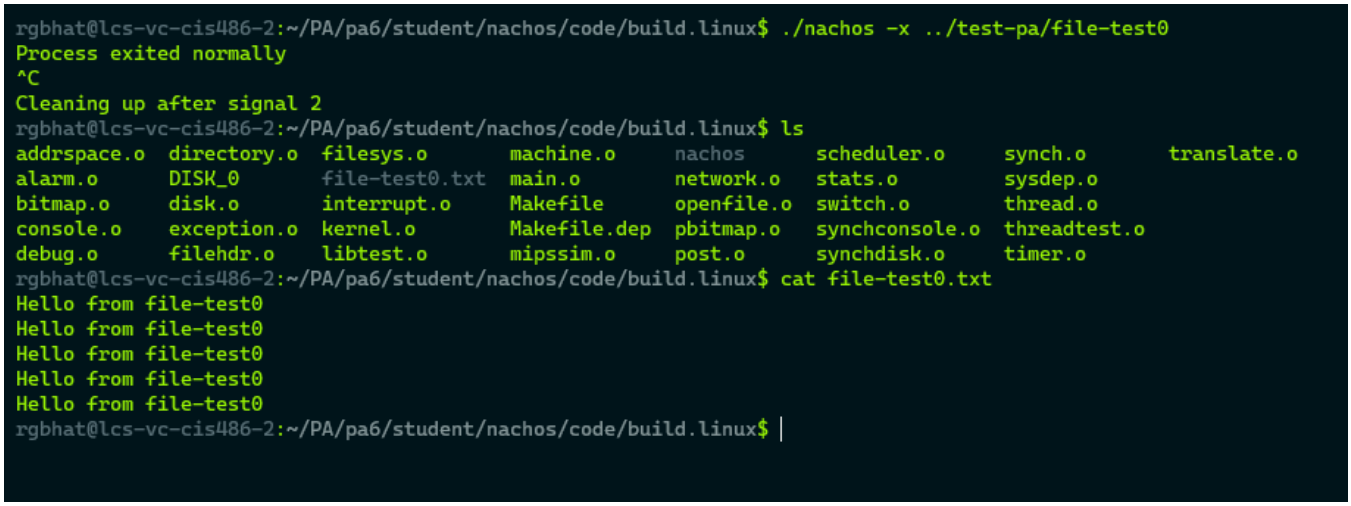
✪ [R-2]: Testing

1. Test files:
    - `file-test0.c` : `file-test0.c` will create a new file '`file-test0.txt`' using `Create()` system call and opens the file using `Open()` system call if the file is successfully created. Then using the file id of the file, the message-'`Hello from file-test0`' is written to the file mentioned by the file descriptor. Once finished, the file is closed using the `Close()` system call.
    - `file-test1.c`: `file-test1.c` creates a `file-test1.txt` file and opens it using `Open()`. Then '`Opened file-test1.txt`' is written to `ConsoleOutput`. Then program tries to open `nofile.txt`. Since the file does not exist, '`File doesn't exist`' message is printed to `ConsoleOutput`.
    - `file-test2.c: file-test2.c` first creates `file-test2.txt` and it is opened. Then the message – '`Hello from file-test2`' is written to the file 5 times and closed. Then again it is opened and a new file `test-file2a.txt` is created and opened. Then contents of the `file-test2.txt` is read using `Read()` system call and data is written to `file-test2a.txt` using `Write()` system call with help of a buffer. Then both the files are closed.
    - `file-test3.c: file-test3.c` first creates file-test3.txt and opens it. Then 'Hello from file-test3' message is written to the file. Then number of bytes written to the file is written to ConsoleOutput. Then file is closed.
2. Features tested in each test program
    - `file-test0.c : Create(), Open(), Write(), Close()`
    - `file-test1.c : Create(), Open(), Write(), Close()`
    - `file-test2.c : Create(), Open(), Read(), Write(), Close()`

- `file-test3.c : Create(), Open(), Write(), Close()`

3. Run

`file-test0.c`



`file-test0.c` will create a new file '`file-test0.txt`' using `Create()` system call and opens the file using `Open()` system call if the file is successfully created. Then using the file id of the file, the message- '`Hello from file-test0`' is written to the file mentioned by the file descriptor. Once finished, the file is closed using the `Close()` system call.

`file-test1.c`

`file-test1.c` creates a `file-test1.txt` file and opens it using `Open()`. Then '`Opened file-test1.txt'` is written to `ConsoleOutput`. Then program tries to open `nofile.txt`. Since the file does not exist, '`File doesn't exist`' message is printed to `ConsoleOutput`.

`file-test2.c`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ./nachos -x ../test-pa/file-test2
Process exited normally
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ls
addrspace.o  directory.o  filesys.o       libtest.o     mipssim.o   post.o          synchdisk.o   timer.o
alarm.o      DISK_0       file-test2a.txt  machine.o     nachos      scheduler.o     synch.o       translate.o
bitmap.o     disk.o       file-test2.txt   main.o        network.o   stats.o         sysdep.o
console.o    exception.o  interrupt.o      Makefile      openfile.o  switch.o        thread.o
debug.o      filehdr.o    kernel.o         Makefile.dep  pbitmap.o   synchconsole.o  threadtest.o
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ cat file-test2.txt
Hello from file-test2
Hello from file-test2
Hello from file-test2
Hello from file-test2
Hello from file-test2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ cat file-test2a.txt
Hello from file-test2
Hello from file-test2
Hello from file-test2
Hello from file-test2
Hello from file-test2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$
```

`file-test2.c` first creates `file-test2.txt` and it is opened. Then the message – '`Hello from file-test2`' is written to the file 5 times and closed. Then again it is opened and a new file `test-file2a.txt` is created and opened. Then contents of the `file-test2.txt` is read using `Read()` system call and data is written to `file-test2a.txt` using `Write()` system call with help of a buffer. Then both the files are closed.

`file-test3.c`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ./nachos -x ../test-pa/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 exited normally
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ls
addrspace.o  directory.o  filesys.o       machine.o     nachos      scheduler.o     synch.o       translate.o
alarm.o      DISK_0       file-test3.txt  main.o        network.o   stats.o         sysdep.o
bitmap.o     disk.o       interrupt.o     Makefile      openfile.o  switch.o        thread.o
console.o    exception.o  kernel.o        Makefile.dep  pbitmap.o   synchconsole.o  threadtest.o
debug.o      filehdr.o    libtest.o       mipssim.o     post.o      synchdisk.o     timer.o
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ cat file-test3.txt
Hello from file-test3
Hello from file-test3
Hello from file-test3
Hello from file-test3
Hello from file-test3
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$
```

`file-test3.c` first creates `file-test3.txt` and opens it. Then '`Hello from file-test3`' message is written to the file. Then number of bytes written to the file is converted to string and it is written to ConsoleOutput along with the message '`Number of Characters Written is` '. Then file is closed.

✪ **[R-3]:** `write-test.c`

```c
#include "syscall.h"

int main()
{

  int i, j;
  OpenFileId output = ConsoleOutput;
  char* name = "Raghunandan Bhat\n";
  char* date = "04/29/22\n";

  for (i = 0; i < 10; i++)
  {
    Write(name, 20, output);
    Write(date, 10, output);
    for (j = 0; j < 10000; j++);
  }
  Exit(0);
}
```

Changes in `Makefile`

```
ifeq ($(hosttype),unknown)
PROGRAMS = unknownhost
else
# change this if you create a new test program!
PROGRAMS = add halt shell matmult sort segments read write read-write prog1 prog2 exit-test0 exit-test1 file-test0 file-test1 file-test2 file-test3 prog3 prog3b prog4 prog5 write-test
endif
```

```
## write-test
write-test.o: write-test.c
	$(CC) $(CFLAGS) -c write-test.c
write-test: write-test.o start.o
	$(LD) $(LDFLAGS) start.o write-test.o -o write-test.coff
	$(COFF2NOFF) write-test.coff write-test
```

Output: `./nachos -x ../test-pa/write-test`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ./nachos -x ../test-pa/write-test
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Raghunandan Bhat
04/29/22
Process exited normally
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$
```

✪ **[R-4]:**

1. file-test4.c

```c
#include "syscall.h"
#include "stdbool.h"

int tostring(char str[], int num)
{
    int i, rem, len = 0, n;
    bool isNegative = false;

    if(num < 0){
        isNegative = true;
        len++;
        num = (-1) * num;
    }
    if(num == 0){
        str[0] = 0 + '0';
        len++;
    }

    n = num;
    while (n != 0)
    {
        len++;
        n /= 10;
    }

    if(isNegative){
        for (i = 0; i < len-1; i++)
        {
                rem = num % 10;
                num = num / 10;
                str[len - (i + 1)] = rem + '0';
        }
        str[0] = '-';
    }else{
        for (i = 0; i < len; i++)
        {
                rem = num % 10;
                num = num / 10;
                str[len - (i + 1)] = rem + '0';
        }
    }

    str[len] = '\0';
  return len+1;
}
```

```
48 int main(){
49
50         int read_bytes = 0, length = 0;
51         char* str = "Hello from test-file4\n";
52         OpenFileId id, close_id, open_id;
53         char buffer[100];
54         char nofbytes[100];
55
56         if(Create("file-test4.txt") < 0){
57                 Write("Error creating file\n", 21, ConsoleOutput);
58                 Exit(1);
59         }
60
61         id = Open("file-test4.txt");
62         if(id < 0){
63                 Write("Error opening file\n", 20, ConsoleOutput);
64                 Exit(1);
65         }else{
66                 Write(str, 22, id);
67         }
68         Close(id);
69
70         open_id = Open("file-test4.txt");
71         if(open_id > 0){
72                 read_bytes = Read(&buffer, 22, open_id);
73                 if(read_bytes > 0){
74                         Write("Read(): Successful: read ", 25, ConsoleOutput);
75                         length = tostring(&nofbytes, read_bytes);
76                         Write(&nofbytes, length, ConsoleOutput);
77                         Write(" bytes\n", 7, ConsoleOutput);
78                 }else{
79                         Write("Read(): Failed: returned ", 25, ConsoleOutput);
80                         length = tostring(&nofbytes, read_bytes);
81                         Write(&nofbytes, length, ConsoleOutput);
82                         Write("\n", 1, ConsoleOutput);
83                 }
84         }
85
86         close_id = Close(open_id);
```

```
 86         close_id = Close(open_id);
 87
 88         if(close_id < 0){
 89                 Write("Close(): Failed: returned ", 26, ConsoleOutput);
 90                 length = tostring(&nofbytes, close_id);
 91                 Write(&nofbytes, length, ConsoleOutput);
 92                 Write("\n", 1, ConsoleOutput);
 93         }else{
 94                 Write("Close(): Successful: returned ", 30, ConsoleOutput);
 95                 length = tostring(&nofbytes, close_id);
 96                 Write(&nofbytes, length, ConsoleOutput);
 97                 Write("\n", 1, ConsoleOutput);
 98         }
 99
100         read_bytes = Read(&buffer, 22, open_id);
101         if(read_bytes > 0){
102                 Write("Read(): Successful: read ", 25, ConsoleOutput);
103                 length = tostring(&nofbytes, read_bytes);
104                 Write(&nofbytes, length, ConsoleOutput);
105                 Write(" bytes\n", 7, ConsoleOutput);
106         }else{
107                 Write("Read(): Failed: returned ", 25, ConsoleOutput);
108                 length = tostring(&nofbytes, read_bytes);
109                 Write(&nofbytes, length, ConsoleOutput);
110                 Write("\n", 1, ConsoleOutput);
111         }
112         close_id = Close(open_id);
113
114         if(close_id < 0){
115                 Write("Close(): Failed: returned ", 26, ConsoleOutput);
116                 length = tostring(&nofbytes, close_id);
117                 Write(&nofbytes, length, ConsoleOutput);
118                 Write("\n", 1, ConsoleOutput);
119         }else{
120                 Write("Close(): Successful: returned ", 30, ConsoleOutput);
121                 length = tostring(&nofbytes, close_id);
122                 Write(&nofbytes, length, ConsoleOutput);
123                 Write("\n", 1, ConsoleOutput);
124         }
```

```
125
126 Exit(0);
127 }
```

Output: `./nachos -x ../test-pa/file-test4`

```
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$ ./nachos -x ../test-pa/file-test4
Read(): Successful: read 22 bytes
Close(): Successful: returned 0
Read(): Failed: returned -1
Close(): Failed: returned -1
Process exited normally
^C
Cleaning up after signal 2
rgbhat@lcs-vc-cis486-2:~/PA/pa6/student/nachos/code/build.linux$
```

2.

a) `int printf(const char *format,…);` printf converts, formats, and prints its arguments on the standard output. It returns the number of characters printed. The format string can have either a normal string which is printed to standrad output stream. It can also have a format specifier-which starts with '%', converts the arguments to printf and prints it to standrad output stream

b) Write() function can not print integer variables since it only supports writing from character buffer. If we pass a string literal which contains an integer(char* str = "123\n"), Write() will be able to print it, but not integers. The Write() should be changed to accept, both character and integers as a parameter and the implementation of Write() should be able to differentiate between different types of paramters.