CS333 – INTRODUCTION TO OPERATING SYSTEM

PROJECT 01
System calls & File - Network Operations

1. General rule:

The project is done in groups: each group has a maximum of **3** students, a minimum of **2** students

- The same exercises will all be scored 0 for the entire practice (even though there are scores for other exercises and practice projects).

- Environment: Linux virtual machine (recommend Ubuntu 18.04), Nachos 4.0

2. Submission Requirements:

Submit assignments directly on the course website, not accepting submissions via email or other forms.

Filename: **StudentID1_StudentID2_StudentID3.zip** (with StudentID1 < StudentID2 < StudentID3)

Ex: Your group has 3 students: 2012001, 2012002 and 2012003, the filename is: **2012001_2012002_2012003.zip**

Include:

1. **StudentID1_StudentID2_StudentID3_Report.pdf:** Writeups should be short and sweet. Do not spend too much effort or include your source code on your writeups. The purpose of the report is to give you an opportunity to clarify your solution, any problems with your work, and to add information that may be useful in grading. If you had specific problems or issues, approaches you tried that didn't work, or concepts that were not fully implemented, then an explanation in your report may help us to assign partial credit
2. **StudentID1_StudentID2_StudentID3_Source**: your team's source code (NachOS-4.0/code/)

Submit the program: when you're done, delete the object files (.o) and executables you created

Note: It is necessary to strictly follow the above requirements, otherwise the work will not be graded.

3. Demo Interviews

Your implementation is graded on completeness, correctness, programming style, thoroughness of testing, your solution, and code understanding.

When administering this course, we do our best to give a fair assessment to each individual based on each person's contribution to the project

4. Grading:

Part		**Describle**	**Point**
1	1	syscall Create	**0,5**
	2	syscall OpenFileID, Close	**0,75**
	3	syscall Read,Write	**0,75**
	4	syscall Seek	**0,5**
	5	syscall Remove	**0,5**
2	1	syscall sockectTCP	**0,5**
	2	syscall Connect	**0,5**
	3	syscall Send	**0,5**
		syscall Receive	**0,5**
3		Advanced	**1**
4		Test program	
	1,2,3,4,5	create, copy, cat, delete, concatenate (0,25 for each of parts)	**1,5**
	6	echo	**1,5**
	7	file transfer	**1**
		Report (*your project will not be graded without the report*)	

Content

Part1. Implement system call for File operations

1. Implement the **int Create(char *name)** system call. The createfile system call will use the Nachos Filesystem Object Instance to create a zero-length file. Remember, the filename exists in user space. This means the buffer that the user space pointer points to must be translated from user memory space to system memory space. The createfile system call returns 0 for successful completion, -1 for an error.

2. Implement the **OpenFileID Open(char *name, int type)** and int **Close(OpenFileID id)** system calls. The user program can open two types of "files", files that can be read only and files that can be read and write. Each process will allocate a fixed size file descriptor table. For now, *set this size to be **20** file descriptors*. The first two file descriptors, 0 and 1, will be reserved for console input and console output respectively. The open file system call will be responsible for translating the user space buffers when necessary and allocating the appropriate kernel constructs. You will use the filesystem objects provided to you in the filesystem directory. (NOTE: We are using the FILESYSTEM_STUB code) The calls will use the Nachos Filesystem Object Instance to open and close files.

   The Open system call returns the file descriptor id (OpenFileID == an integer number), or –1 if the call fails. Open can fail for everal reasons, such as trying to open a file or mailbox that does not exist or if there is not enough room in the file descriptor table. The type parameter will be set to 0 for a standard file and 1 for a read only file. If the type parameter is set to any other value, the system call should fail. The close system call will take a file descriptor as the parameter. The system call will return –1 on failure and 0 on success

3. Implement the int **Read(char *buffer, int charcount, OpenFileID id)** and int **Write(char *buffer, int charcount, OpenFileID id)** system calls. These system calls respectively read and write to a file descriptor ID. Remember, you must translate the character buffers appropriately and you must differentiate between console IO (OpenFileID 0 and 1) and File (any other valid OpenFileID).

   In case the Console is a read and write console, using the *SynchConsoleInput* class and the *SynchConsoleOutput* class for reading and writing, you must ensure that the correct data is returned to the user. Reading and writing to the console will return the correct number of characters read or written, not the charcount. In case of read or write failure, return -1

4. Implement the int **Seek(int pos, OpenFileID id)** system call. Seek will move the file cursor to a specified location. The parameter pos will be the absolute character position within a file. If pos is a –1, the position will be moved to the end of file. The

system call will return the actual file position upon success, -1 if the call fails. Seeks on console I/O will fail

5. Install system call int Remove(char *name). The Remove system call will use the Nachos FileSystem Object to delete the file. Note: need to check if the file is open or not before deleting.

Part2. Implement system call for Network operations

1. Implement system call int **SocketTCP()**. You build an array of file descriptor tables with the size of 20 file descriptors.

   The system function call "**SocketTCP**" will return the file descriptor id (int is an integer), or -1 if there is an error.

2. Implement system call int **Connect(int socketid, char *ip, int port)**. Connect to server according to IP and port information. Returns 0 if the connection is successful, and -1 if it fails

3. Implement system call **int Send(int socketid, char *buffer, int len), int Receive(int socketid, char *buffer, int len)**. Send and receive data from socket.

   - If successful, returns the number of bytes sent or received

   - If connection is closed, return 0

   - If it fails, return -1

4. Implement system call **int Close(int socketid)**. Close socket with 0 for success, -1 for error

Part3. Advanced

As part 1 and part 2, there are 2 file descriptor table arrays. In this requirement,

- You are only allowed to use **a single array** for file and network management
- Only use Read/Write system call for file system and network (it means that you don't use Send/Receive system call at all).

Part4. Test program

1. Write a program createfile that generates a file to test the **Create** system call. You can use a fixed filename or let the user input from the console from **ReadString**

   Vd: Tạo file hello.txt

   ```
   ./nachos -x ../test/createfile hello.txt
   ```

2. Write a program **cat** , ask for filename, and then display the contents of that file

   Vd: Print the content of file hello.txt

   ```
   ./nachos -x ../test/cat hello.txt
   ```

3. Write a program **copy**, ask for the name of the source file and the destination file, and make a copy

   Vd: Copy file a.txt to file b.txt

   ```
   ./nachos -x ../test/copy a.txt b.txt
   ```

4. Write a **delete** program to check the system call Remove

   Vd: xóa file hello.c

   ```
   ./nachos -x ../test/delete hello.c
   ```

5. Write a **concatenate** program to concatenate the contents of two files, requiring the input of source file name 1 and source file 2

   ```
   ./nachos -x ../test/concatenate a.txt b.txt
   ```

6. Write a client program and a server program to test network sockets. Echo program:

   a. Client sends a message

   b. The server receives the message and converts it to uppercase before sending it back to the client

   c. The client receives the message string (capitalized), then outputs it to the screen

   Your server program can refer to:
   https://mohsensy.github.io/programming/2019/09/25/echo-server-and-client-using-sockets-in-c.html

   You will write your own client program, create 4 sockets and connect to the server. Use these 4 sockets to send message to server and close socket

   ```
   ./nachos -x ../test/echoclient
   ```

7. Write a client program and a server program to test network sockets. File transfer program:

   a. Client will read and send file content (text file) to Server

   b. The server receives the message and turns it into uppercase, then sends it back to the client

   c. The client receives the message string (capitalized), then writes it to the file

   Just create a socket to test this part

   ```
   ./nachos -x ../test/fileclient a.txt b.txt
   ```

file a.txt is the source file (which is read and sent), file b.txt is the destination file (which is written)