CS342300 2023 MP1 – System Call

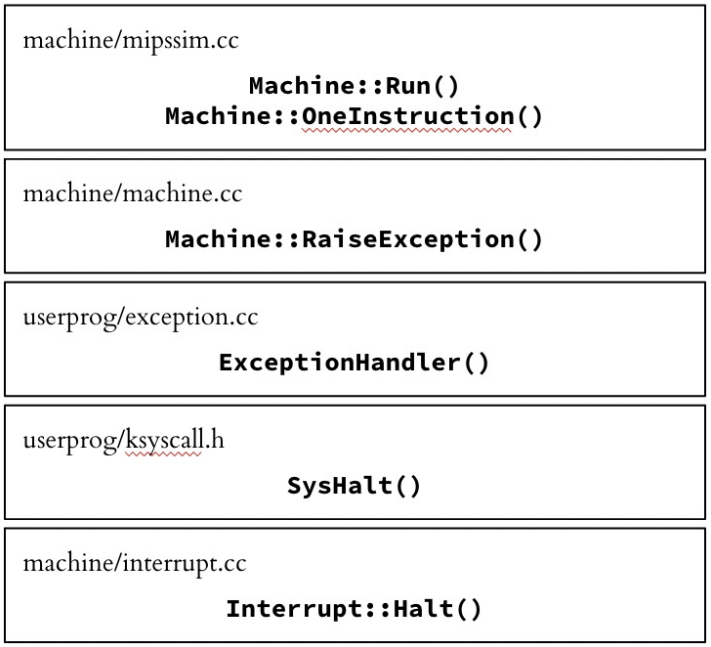
Deadline: 2023/03/13 23:59

★ Goal

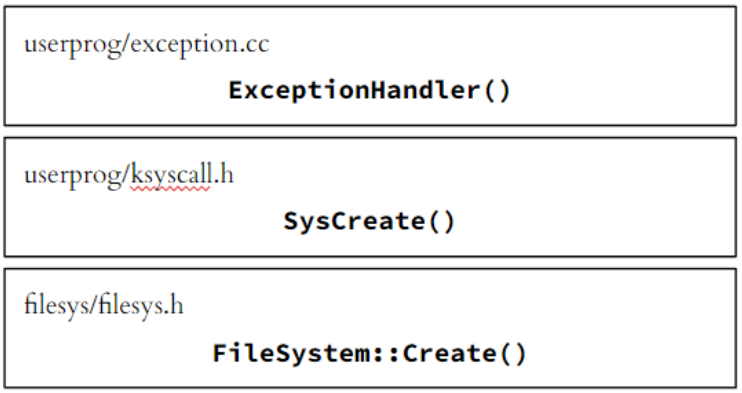
1. Understand how to work in a Linux environment.
2. Understand how system calls are implemented by OS.
3. Understand the difference between user mode and kernel mode.

★ Assignment

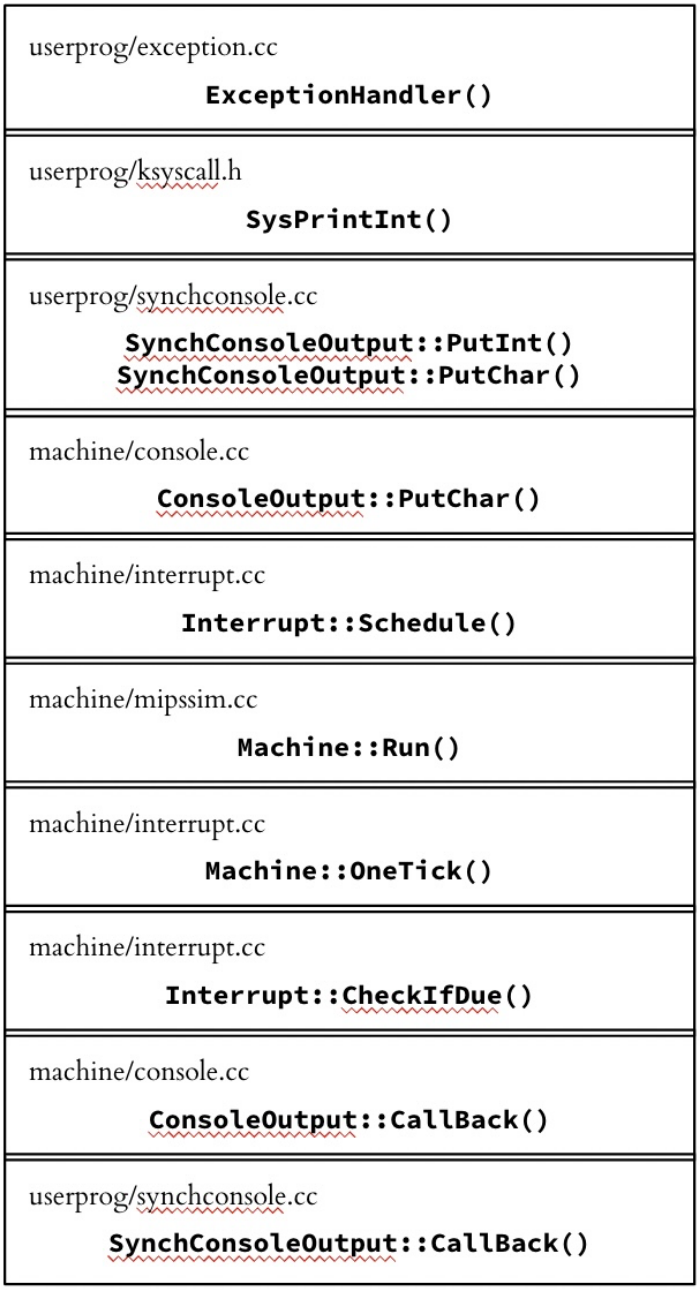
- Trace code
  - Trace the **SC_Halt** system call to understand the implementation of a system call. (Sample code: halt.c)



  - Trace the **SC_Create** system call to understand the basic operations and data structure in a file system. (Sample code: createFile.c)



- Trace the **SC_PrintInt** system call to understand how NachOS implements asynchronized I/O using CallBack functions and register schedule events. (Sample code: add.c)
  $ ../build.linux/nachos -d + -e add



- Requirements (Include the following answers in your writing report)
  - Explain the purposes and details of each function call listed in the code path above.
  - Explain how the arguments of system calls are passed from user program to kernel in each of the above use cases.
- Implement four I/O system calls in NachOS

2-1. Working item:

★ `OpenFileId Open(char *name);`

Open a file with the name, and return its corresponding OpenFileId.

Return -1 if fail to open the file.

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

Write “size” characters from the buffer into the file, and return the number of characters actually written to the file.

Return -1, if fail to write the file.

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

Read “size” characters from the file to the buffer, and return the number of characters actually read from the file.

Return -1, if fail to read the file.

★ `int Close(OpenFileId id);`

Close the file with id.

Return 1 if successfully close the file. Otherwise, return -1.

Need to delete the OpenFile after you close the file

(Can’t only set the table content to NULL)

2-2. Requirement:

- **Must maintain OpenFileTable and use the table entry number of OpenFileTable as the `OpenFileId`.**
- **Must handle invalid file open requests, including the non-existent file, exceeding opened file limit (at most 20 files), etc.**
- All valid file open requests must be accepted if the opened file limit (at most 20 files) is not reached.
- **Must handle invalid file read, write, close requests, including invalid id, etc.**
- DO NOT use any IO functions from standard libraries (e.g. printf(), cout, fopen(), fwrite(), write(), etc.).
- DO NOT change any code under “machine/” folder
- DO NOT modify the content of OpenFileTable outside “filesystem/” folder
- **DO NOT modify the declaration of OpenFileTable, including the size.**

2-3. Example testcases:

- First use the command “**../build.linux/nachos -e fileIO_test1**” to write a file.
- Then use the command “**../build.linux/nachos -e fileIO_test2**” to read the file

```
[test@lsalab test]$ ../build.linux/nachos -e fileIO_test2
fileIO_test2
Passed! ^_^
Machine halting!

This is halt
Ticks: total 777, idle 0, system 110, user 667
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0
```

2-4. Hint & Reminder:

- We use the stub file system for this homework, so DO NOT change or remove **the flag -DFILESYS_STUB in the Makefile under build.linux/**.
- There will be hidden test cases to test all the I/O system calls. Make sure your implementation meets all the requirements.
- There are some TODOs to help you finish the assignment.
- Files to be modified
  - test/start.S
  - userprog/syscall.h, exception.cc, ksyscall.h
  - filesys/filesys.h

- Report
  - Cover page, including studentID.
  - Explain how system calls work in NachOS as requested in the **Trace code** part.

- Explain your implementation as requested in the **Implementation** part.
- What difficulties did you encounter when implementing this assignment?
- Any feedback you would like to let us know.

★ Instructions

1. Set VPN
   - https://reurl.cc/NZpGOQ
2. Login server
   - 10.121.187.197
   - Username: os23sXX (e.g. os23s01)
   - Password: You are required to reset the password once you login
3. Install NachOS

```
$ cp -r /home/os2023/share/NachOS-4.0_MP1 ~/.
$ cd NachOS-4.0_MP1/code/build.linux
$ make clean
$ make
```

4. Compile / Rebuild NachOS

```
$ cd NachOS-4.0_MP1/code/build.linux
$ make clean
$ make
```

5. Test NachOS

```
$ cd NachOS-4.0_MP1/code/test
$ make clean
$ make halt
$ ../build.linux/nachos -e halt
```

★ Grading

1. Implementation correctness – 60%
   (a) Pass the public and hidden test cases.
   (b) You DO NOT need to upload NachOS code to eeclass, and just put your code to the folder named "NachOS-4.0_MP1" in your home directory.
   (c) Your working directory will be copied for validation after the deadline.
2. Report – 15%
   (a) Name the report "**MP1_report_[StudentID].pdf"**, and upload it to eeclass.
3. Demo – 25%
   (a) We will ask several questions about your codes.
   (b) Demo will take place on our server, so you are responsible to make sure your code works on our server.
4. TAs won't help you recover from file loss or misedited. Please make sure you backup your code. You can use git or simply make a copy in your local.
5. **Late submissions will not be accepted.** Refer to the course syllabus for detailed homework rules and policies.
6. **Plagiarism**
   (a) **NEVER SHOW YOUR CODE** to others.
   (b) If the codes are similar to other people (including your upperclassman) and you can't answer questions properly during the demo, you will be identified as plagiarism.

★ Appendix (Nachos Directory Structure)

- lib/
  - Utilities used by the rest of the Nachos code

- machine/
  - The machine simulation
  - All files here CANNOT be modified for any homework assignments
- threads/
  - Nachos is a multi-threaded program. Thread support is found here. This directory also contains the main() routine of the nachos program in main.cc.
- test/
  - User test programs to run on the simulated machine. This directory contains its own Makefile.
  - This is where you can write your own test programs
- userprog/
  - Nachos operating system code to support the creation of address spaces, loading of user (test) programs, and execution of test programs on the simulated machine.
  - You might need to modify the kernel code here
- network/
  - Nachos operating system support for networking. Several independent simulated Nachos machines can talk to each other through a simulated network.
  - We don't need to touch the code in this course.
- filesys/
  - Two different file system implementations are here. The **"real"** file system uses the simulated workstation's simulated disk to hold files. A **"stub"** file system translates Nachos file system calls into UNIX file system calls.