Operating Systems
Assignment 1 – *Easy*

Total: 50 Marks

Instructions:

1. The assignment has to be done individually.
2. You can use Piazza for any queries related to the assignment and avoid asking queries on the last day.

The assignment has five parts.

1 Installing and Testing

- xv6 is available at the following link.
- **Install xv6 version 11**.
- The instructions for building and installing xv6 can be found here.
- The build instructions for Qemu can be found here.

2 Enhanced Shell for xv6: 10 Marks

In this part, you must protect the shell with a username-password-based xv6 login system. The user should enter the password, and the shell should only start after successful authentication. The username and password values should be defined as macros in the `Makefile` in the following format:

```
USERNAME=<username>
PASSWORD=<password>
```

There should be no spaces and special characters in the usernmae or password: only upper/lower case letters and numbers.

The user should get a maximum of three attempts, after which the login process should get disabled. The user should receive the command-line prompt "Enter Username" and then "Enter Password". This prompts need to be repeated if the entered information is incorrect (subject to having enough available attempts). The password should only be asked for if the correct username is given. An example follows.

```
$Enter Username:
$Enter Password:
 Login successful
$
```

3 Shell Command: `history`: 10 Marks

In this part, you must implement an additional command `history` that needs to display a list of all the processes that have been executed until now (sorted in ascending order of time). Each entry of the list must contain the following information: `pid, process name, total memory utilization`. Total memory utilization should include the memory in **bytes** allocated for text, bss, data, stack and heap segments for each process. For this, you must create another system call in xv6 called *sys_gethistory*, which will return the required data. The signature of the system call is as follows:

```
int sys_gethistory()
```

If the system call is successful, it should return the data in the aforementioned format; otherwise, it should return -1.
The syscall identifier (syscall.h) for this call should be 22.

```
$history
  3 ls 10
  4 echo 5
$
```

4 Shell command: `block` : 10 Marks

In this part, you must implement a command `block` that blocks a system call for any process that has been invoked by the current shell. The call is blocked until the `unblock` command is invoked. Hence, we need to define two new system calls: *sys_block* and *sys_unblock*. The signatures of these system calls are as follows:

```
int sys_block(int syscall_id)
int sys_unblock(int syscall_id)
```

- *syscall_id:* Identifier of the system call to be blocked/unblocked. Note that the list of supported syscalls in xv6 is given in the file syscall.h

In both cases, if the system call is successful, it should return 0; otherwise, it should return -1 (if it is blocked).
The syscall identifiers (syscall.h) for these calls should be 23 and 24, respectively.

Create two user-level commands *block* and *unblock*, both of which will take a single argument (syscall ID) and block/unblock the specified system call in the system (for all the processes spawned by the current shell), respectively.

Sample command for blocking syscall 7 (SYS_exec).

```
$block 7
```

The shell should print "syscall 7 is blocked" when a blocked syscall is invoked.

Critical System Calls: Ensure that the *fork* and *exit* system calls cannot be blocked at the kernel level.

5 Shell command : chmod : 10 Marks

In this part, you must implement the `chmod` command. More information about `chmod` can be found at this link.
For this, you must implement a system call *sys_chmod*, for which the signature is:

```
int sys_chmod(const char* file, int mode)
```

- *file:* The name of the file whose permissions are to be modified
- *mode:* 3-bit integer
  - *bit 0:* Allows the file to be read.
  - *bit 1:* Allows the file to be written.
  - *bit 2:* Allows execution.

If the system call is successful, it should return 0; otherwise, it should return -1.
The syscall identifier (in syscall.h) for this call should be 25.
The command should take two arguments (the filename and mode) and modify the permissions of the file as specified. If an operation is attempted that is not allowed, print the message `Operation <op> failed` where `op` can be read, write or execute.

6 Report: 10 Marks

Page limit: 10
The report should mention the implementation methodology for all the parts of the assignment. Adding small code snippets is alright. Alternatively, the pseudocode should also suffice.

- Any details that are relevant to the implementation.
- Submit a pdf file containing all the relevant details.
- Say what you have done that is extra.

7 Submission Instructions

- We will run MOSS on the submissions. Any cheating will result in a zero in the assignment, a penalty as per the course policy, and possibly much stricter penalties (including a fail grade).

- There will be NO demo for assignment 1 (easy). Your code will be evaluated using a check script (check.sh) on hidden test cases, and marks will be awarded based on that. You can find the test scripts here (will be released soon). The README file inside the test scripts tarball contains the instructions on how to run the test scripts.

How to submit:

1. Copy your report file to the xv6 root directory.
2. Then, in the root directory, run the following commands:

```
make clean
tar -czvf assignment1_easy_<entryNumber>.tar.gz *
```

   This will create a tarball in the same directory with the name, $assignment1\_easy\_\langle entryNumber\rangle.tar.gz$. Submit this tarball on Moodle. Entry number format: 2020CSZ2445. (*All English letters will be in capitals in the entry number.*)

3. Please note that if the report is missing in the root directory, no marks will be awarded.