COL331: OPERATING SYSTEMS
ASSIGNMENT 1

AVNI AGGARWAL
2022CS11605

Enhanced Shell for xv6

Implementation details:

Made changes in init.c:

- Defined max_attempts as 3
- In main function, compare the username entered with the actual username and if it matches, then proceed to ask password, if it doesn't match, then ask username once again and increase the no of attempts.
- If username matches, then again ask for password and compare string with actual password, and if it matches then print login successful and proceed.
- If password doesn't match, then increase the no of attempts and again give prompt for entering username.

```
// Authentication loop
while (attempts < MAX_ATTEMPTS) {
    printf(1, "Enter username: ");
    gets(username, sizeof(username));
    username[strlen(username) - 1] = '\0';

    if (strcmp(username, USERNAME) != 0) {
        //printf(1, "User not found. Try again.\n");
        attempts++;
        continue;
    }

    printf(1, "Enter password: ");
    gets(password, sizeof(password));
    password[strlen(password) - 1] = '\0';

    // Check credentials
    if (strcmp(password, PASSWORD) == 0) {
        printf(1, "Login successful\n");
        break;
    } else {
        //printf(1, "Login failed. Try again.\n");
        attempts++;
    }
}
```

In the makefile, define macros and cflags.

```
USERNAME=cs5200444
PASSWORD=srijan
CFLAGS += -DUSERNAME=\"$(USERNAME)\" -DPASSWORD=\"$(PASSWORD)\"
```

Shell command : history

Implementation details:

- Define a syscall (22) in syscall.h, syscall.c and usys.S
- Define struct to store history entry and then make an array of such structs to store the history details

```
#define NPROC 64

struct history_entry {
  int pid;
  char name[16];   // Process name
  uint mem_usage;  // Total memory usage
};

extern struct history_entry proc_history[NPROC];  // Declare history array
extern int history_count;
```

Fig : proc.h

- Maintained a flag is_executed that tracks whether a process is executed completely or not (since we have to report history when exec is successful for a process). This flag is updated in exec function of exec.c
- In proc.c, in exit function, i am updating the proc_history array and history_count by writing pid, name , mem usage (curproc->sz) .

```
if (curproc->is_executed && history_count < NPROC) {
  struct history_entry *entry = &proc_history[history_count];  // Get history entry slot

  entry->pid = curproc->pid;
  safestrcpy(entry->name, curproc->name, sizeof(entry->name));

  // Store memory usage
  entry->mem_usage = curproc->sz; // Total memory size

  history_count++;

  //cprintf("Process added to history: PID %d, Name: %s, Memory: %d bytes\n", entry->pid, entry->name, entry->mem_usage);
}
```

Fig: proc.c

- Written a function sys_gethistory() in sysproc.c which sorts the proc_history array in starttime (pid) and then prints each history entry

```
int sys_gethistory(void) {
  if (history_count == 0) {
    return -1;  // No history available
  }

  for (int i = 0; i < history_count - 1; i++) {
    for (int j = 0; j < history_count - i - 1; j++) {
      if (proc_history[j].pid > proc_history[j + 1].pid) {
        struct history_entry temp = proc_history[j];
        proc_history[j] = proc_history[j + 1];
        proc_history[j + 1] = temp;
      }
    }
  }
  for (int i = 0; i < history_count; i++) {
    struct history_entry *entry = &proc_history[i];  // Get history entry
    cprintf("%d %s %d\n", entry->pid, entry->name, entry->mem_usage);
  }

  return 0;
}
```

Shell command: chmod

Implementation technique:

- Define a syscall (25) in syscall.h, syscall.c and usys.S
- Created an uint to store the permission bit in inode and dinode (in file.h and fs.h respectively) (in case of dinode, had to do padding). Initialised this permission bit as 7(111) in mkfs.c, sysfile.c, and fs.c .
- Then in file.c, where the fileread and filewrite functions are defined, made a check if the permission bit for read or write resp. Is 1 or not, if it is zero then return -1 and operation is failed.

```
if (!(f->ip->permission & 0b001)){
  cprintf("Operation read failed\n");
  return -1;
}
```

Fig: file.c (read permission bits check)

- For exec, the sys_exec function is defined in sys_file.c, there a same check like read and write is made and if bit is 0, then exec is failed.
- Made a function chmod in fs.c with parameters filename and mode, that sets the permission to be equal to mode and updates the inode pointer.

```
int chmod(const char *path, int mode){

  struct inode *ip;
  // cprintf("Changing permissions of %s to %d\n", path, mode);

  begin_op();
  if ((ip = namei((char*)path)) == 0) {
    end_op();
    return -1;
  }

  ilock(ip);
  ip->permission = mode; // Change permission
  iupdate(ip);      // Write back to disk
  iunlockput(ip);
  end_op();

  return 0;
}
```

- In sys_file.c, made a function sys_chmod(void) that calls this chmod function.
- In sh.c, if we find the chmod command, then first find filename and mode by string operations and call the chmod function.

```
int
sys_chmod(void)
{
  char *path;
  int mode;
  // cprintf("sys_chmod called\n");
  if(argstr(0, &path) < 0 || argint(1, &mode) < 0)
    return -1;
  if (chmod(path, mode)<0){
    return -1;
  }
  return 0; // Success
}
```