COL331 Assignment 1
Shell Commands Implementation

Yash Bansal - 2022CS51133

Contents

1 Protected Shell for xv6

The following steps are used to implement username-password-based authentication in xv6.

1. Defining the USERNAME and PASSWORD macros in makefile. These macros are used to hardcode the username and password to the executable.
2. The starting file that initializes the first shell in xv6 is **init.c** , so we ask the user for a username and password just before the fork process to initialize the shell.
3. These are asked for a maximum of 3 times, and if we are out of valid attempts, the shell will not be created.
4. If the password is incorrect after entering the correct username, we ask for the username again.

2 Shell commands:- General steps

The following steps are used to implement any shell command in xv6.

1. **Defining the new system call identifier:-**

   In **syscall.h**, the identifier numbers for all the system calls are defined.

```
#define SYS_gethistory 22
#define SYS_block 23
#define SYS_unblock 24
#define SYS_chmod 25
```

2. **Declaring the new system call functions:-**

   In **syscall.c**, we declare the new system call functions and map them to the corresponding system call identifier.

```
extern int sys_gethistory(void);
extern int sys_block(void);
extern int sys_unblock(void);
extern int sys_chmod(void);
```

```
[SYS_gethistory] sys_gethistory,
[SYS_block] sys_block,
[SYS_unblock] sys_unblock,
[SYS_chmod] sys_chmod,
```

3. **Declaring the new user level function functions:-**

   In **user.h**, we declare the user level functions, which then finally map to the corresponding assembly level function in **usys.S**. This assembly code calls the corresponding system call using the syscall identifier.

```
int gethistory(void);
int block(int);
int unblock(int);
int chmod(const char* , int);
```

```
SYSCALL(gethistory)
SYSCALL(block)
SYSCALL(unblock)
SYSCALL(chmod)
```

4. **Making the required cases:-**

   In **sh.c**, the required user-level function is called depending upon the command written to the shell.

3 History

1. In **proc.h**, we define the structure **proc_history**, which stores the necessary details for each completed process. In addition, an array of these structs is initialized. This contains an attribute time counter, which is updated each time a new process is created.

```
struct proc_history
{
    int pid;
    char proc_name[100];
    int mem_util;
    int time_stamp;
};

#define max_procs 100
extern struct proc_history hist_array[max_procs];
```

2. Whenever a function is exited from the exec system call, we add it to this array. We added new attributes timestamp and is_failed to the proc struct to store the time counter when the process was created and whether the process has been exited successfully.

3. We add the struct to the list only when the process is exited successfully.

4. When history is called, we sort the array based on the timestamp, and then the required details are printed.

4 Block-Unblock

1. In structure **proc**, we add a new attribute **block_bit_vector**, which is essentially a bit vector (as an integer). The nth bit of this attribute stores whether the corresponding system call is blocked or not for the child processes.

2. When the block function is called, the corresponding bit is set of the block_bit_vector attribute. If unblock is called, the bit is unset.

3. In **syscall.c**, whenever a system call is called, we check for all the ancestors of the process whether that system call is blocked or not. If the call is blocked by any of the ancestor, we do not allow the process to do the system call.