<u>Assignment - 2</u>

Introduction to Operating Systems
(UE15CS302)

By Prof. Nitin V Pujari

Submitted By:
Ajay Police Patil
(01FB15ECS021)

Tvarita Jain
(01FB15ECS328)

Department Of Computer Science and Engineering
PES University
Bangalore
Nov 2017

PID_IOS_0817_2

Write a background process to capture shell commands entered on xv6 shell.

INTRODUCTION

xv6, being a UNIX like operating system uses system calls to interface between the user program and the system. The shell is an ordinary program that reads commands from the user and executes them, and is the primary user interface to traditional Unix-like systems. The fact that the shell is a user program, not part of the kernel, illustrates the power of the system call interface: there is nothing special about the shell. It also means that the shell is easy to replace; as a result, modern Unix systems have a variety of shells to choose from, each with its own user interface and scripting features. The xv6 shell is a simple implementation of the essence of the Unix Bourne shell. Interacts with the kernel through system calls.

Whenever we type a command in xv6 shell , the main loop reads the input on the command line using getcmd function in sh.c. Then it calls fork, which creates a copy of the shell process. The parent shell calls wait, while the child process runs the command.The *getcmd* functon just takes input whatever you type in the xv6 shell.

A file descriptor is a small integer representing a kernel-managed object that a process may read from or write to. A process may obtain a file descriptor by opening a file, directory, or device, or by creating a pipe, or by duplicating an existing descriptor. By convention, a process reads from file descriptor 0 (standard input), writes output to file descriptor 1 (standard output), and writes error messages to file descriptor 2 (standard error).

The exit system call causes the calling process to stop executing and to release resources such as memory and open files.

The call write(fd, buf, n) writes n bytes from buf to the file descriptor fd and returns the number of bytes written. Fewer than n bytes are written only when an error occurs. Write writes data at the current file offset and then advances that offset by the number of bytes written: each write picks up where the previous one left off.

IMPLEMENTATION

It can be implemented by editing user mode process *sh.c* file . Creating a file and opening it using *open* system call with parameters filename as *history* and setting flags *O_CREATE* and *O_APPEND.* The *O_CREATE* flag creates the file and *O_APPEND* flag writes and appends command to the end of the file . Then we check that the *fd* is greater than zero , so that indicates that file has been created and we are ready to write in it. We write into the file using *write()* system call .
We compare with size of the string and size of the buffer , if fewer than n bytes are written , it will fail to write in the file *history* . It will exit from the loop using *exit()* system call by stoping execution . To write in a file clearly , so that no overlapping of leeter occurs , after

every command *“\n”* is concatenated . Then atlast we close the file using *close()* system call .

CODE

We can do this by editing user mode process *sh.c* file .
In *getcmd* function ,

```
int fd;
fd = open("history_dump", O_CREATE | O_APPEND);
if(fd >= 0)
{
   printf(1, "File Created\n");
}
else
{
   printf(1, "Error : Failed to Create\n");
   exit();
}

char *str = strcat(buf,"\n");
int size = strlen(str);

if(write(fd, str, size) != size)
{
   printf(1, "Error : Failed to write on File\n");
   exit();
}
printf(1, "Written\n");
close(fd);
```

Points

- *fd = open("history", O_CREATE | O_APPEND);*
  opens or creates and opens the file of name “*history*”
- flags used : *O_CREATE* and *O_APPEND*
- *O_CREATE :* creates the file
- *O_APPEND : b*efore each write, the file offset is positioned at the end of the file
- *str* – concatenates the commands that entered in shell(stored in buf) with “*\n*”
- *size* – size of the str.
- *write(fd, str, size)* – writes the n bytes (depends on size) from str to the file descriptor fd and returns bnumber of bytes written
- *close(fd)* - closes file descriptor

And we can read the contents from the file *“history_dump”* by adding system call *history.*

CODE

```
#include "types.h"
#include "user.h"
#include "stat.h"
#include "fcntl.h"

int main(void)
{
        int fd;
        //int ctr=1;
        static char buf[100];
        fd=open("history_dump",O_RDWR);
        read(fd,buf,500);


        printf(1,buf);

        exit();
}
```

Add strcat function ,

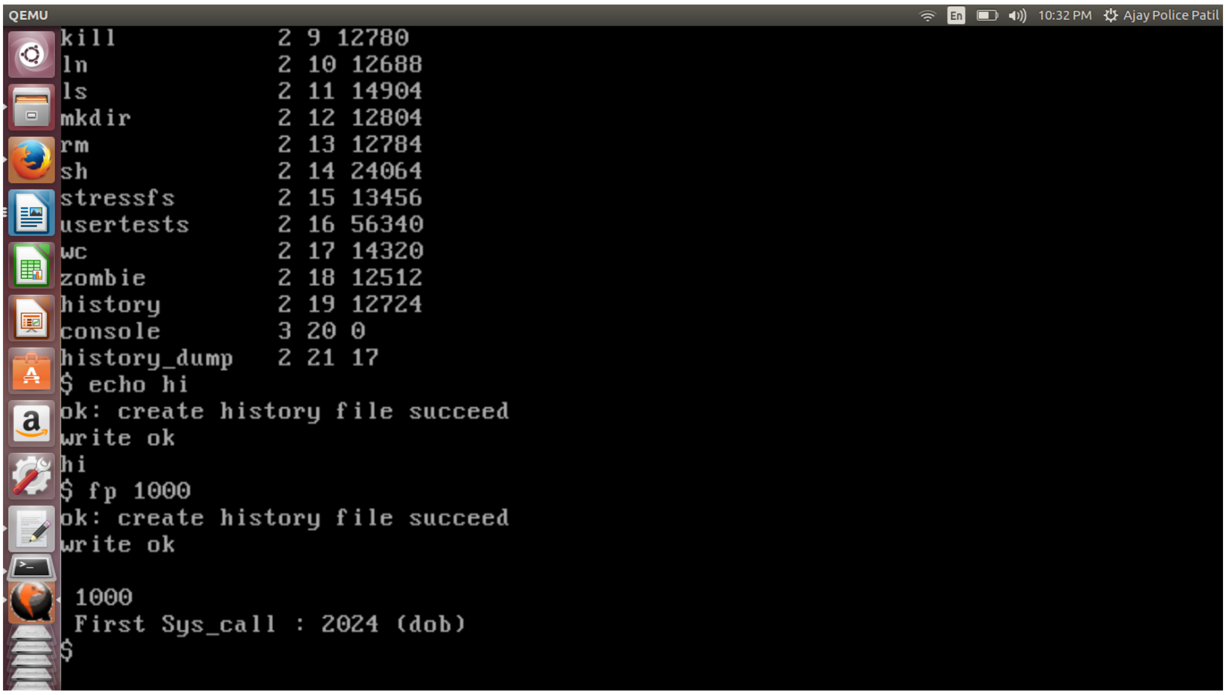
- Open *ulib.c* and *string.c* . Add the code
- Add definition of strcat in *user.*

CODE

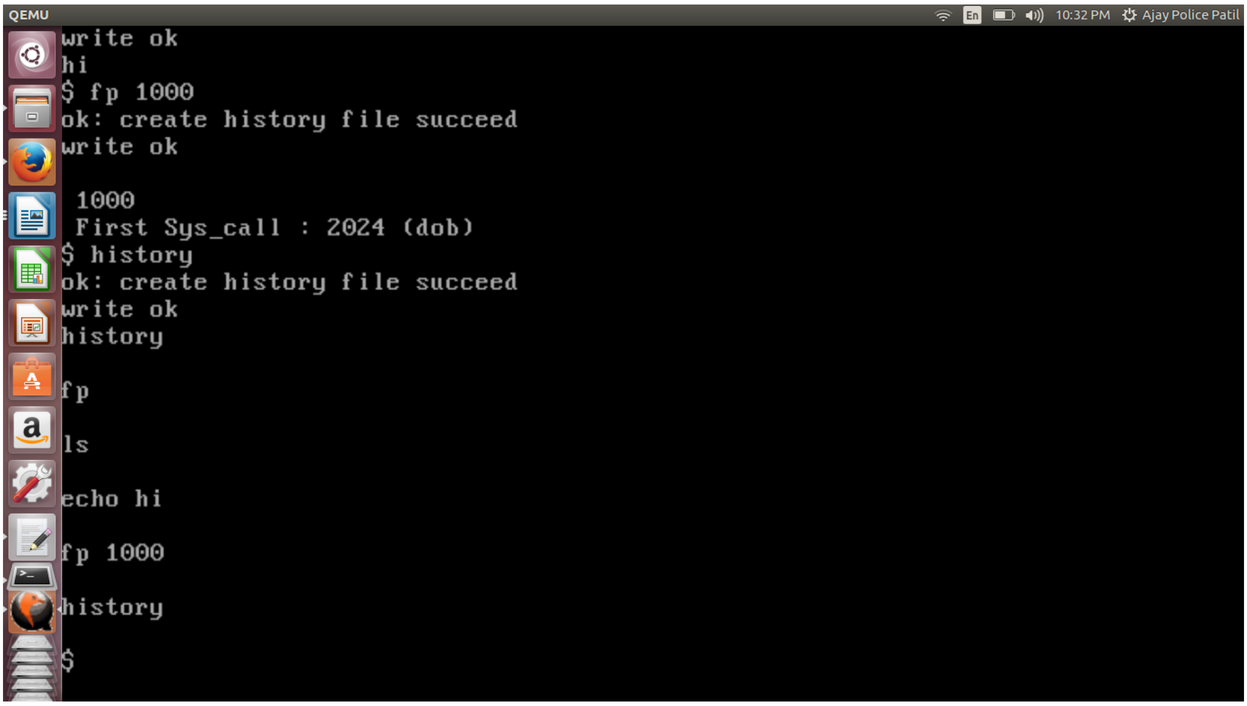
```
char* strcat(char* s1, const char* s2)
{
  char* b = s1;

  while (*s1) ++s1;
  while (*s2) *s1++ = *s2++;
  *s1 = 0;

  return b;
}
```

OUTPUT

$ echo hi
$ fp 1000



$ history



Further Improvement :

• We can add a system call to capture shell commands

• We can implement with arrow keys (&) to capture recently used command.

Reference Links:

[1] O_APPEND flag [xv6 flags]
https://github.com/guilleiguaran/xv6/pull/1/commits/ccdfc8f496ede9f16fce280c0042b3b526c99610

[2] xv6 Documentation [MIT's OS]
https://pdos.csail.mit.edu/6.828/2012/xv6/book-rev7.pdf