Operating Systems – Assignment 1 Report

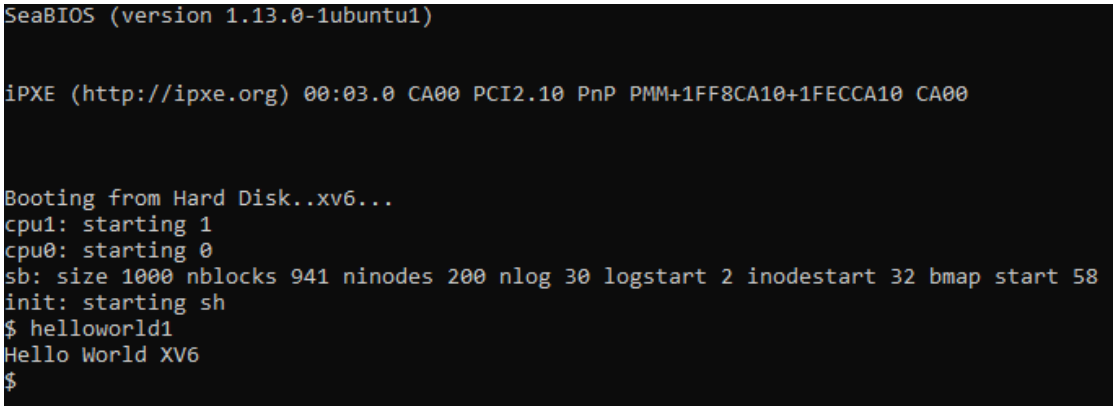
Introduction

On this assignment we extend the functionality of the XV6 operating system by:

- Making a "helloworld" user space program to practice making user space programs.
- Adding a PATH variable to the shell.
- Making a "tee" user space program to practice making user space programs with changeable file descriptors.
- Adding a "getpinfo" system call to the kernel.
- Changing the wait and exit system calls to include the common behavior of these system calls in most operating systems.

Helloworld User Space Program

Shell Output:



PATH Variable Implementation

Shell Output:

```
Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ set PATH /:/l1/:/l2/
$ mkdir l1
$ cd l1
$ ls
.              1 22 32
..             1 1 512
$ echo borak > borak.txt
$ ls
.              1 22 48
..             1 1 512
borak.txt      2 23 6
$ echo yuval | cat
yuval
```

Implementation Code:

```
//PATH implementation variables
char *PATH[10];
int num_of_paths;
```

```
87      //------------------------------------------------------------------------------------
88      //if execution in current directory failed, check if program exists in PATH directories
89      //------------------------------------------------------------------------------------
90      for(int i = 0 ; i < num_of_paths ; i++){
91        char* absolutePath = malloc((strlen(PATH[i]) + strlen(ecmd->argv[0]))*sizeof(char));
92        memset(absolutePath,0,(strlen(PATH[i]) + strlen(ecmd->argv[0]))*sizeof(char));
93        absolutePath = strcpy(absolutePath,PATH[i]);
94        strcpy(absolutePath+strlen(PATH[i]),ecmd->argv[0]);
95        exec(absolutePath, ecmd->argv);
96      }
97      //------------------------------------------------------------------------------------
```

```
187     //------------------------------------------------------------------------------------
188     //set PATH command
189     //------------------------------------------------------------------------------------
190     if(buf[0] == 's' && buf[1] == 'e' && buf[2] == 't' && buf[3] == ' ' &&
191       buf[4] == 'P' && buf[5] == 'A' && buf[6] == 'T' && buf[7] == 'H' && buf[8] == ' '){
192       set_PATH(buf);
193       continue;
194     }
195     //------------------------------------------------------------------------------------
```

```c
//------------------------------------------------------------------------
//given the input buffer, set the PATH global variable to store the given
//paths.
//------------------------------------------------------------------------

void set_PATH(char * buf){

        //initiate assisting variables
        int counter = 0;
        char* start_ptr = &buf[9];
        char* end_ptr = strchr(buf,':');

        //go through every path in the buffer
        while(start_ptr != NULL){
          *end_ptr = 0;

          //copy it into a cell in PATH
          PATH[counter] = (char*)malloc(strlen(start_ptr)*sizeof(char)+1);
          memset(PATH[counter],0,strlen(start_ptr)*sizeof(char)+1);
          PATH[counter] = strcpy(PATH[counter],start_ptr);
          start_ptr = end_ptr + 1;
          counter++;

          //if there are no more paths in the buffer
          if(*start_ptr!='/'){

            //update num_of_paths and finish
            PATH[counter-1][strlen(PATH[counter-1])-1] = 0;
            num_of_paths = counter;
            break;
          }
          end_ptr = strchr(start_ptr,':');

        }
}
```

Making A "tee" User Space Program

Shell Output:

```
Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ tee
Usage is: tee FILE1 or tee FILE1 FILE2
$ echo hi > 1.txt
$ cat 1.txt
hi
$ echo Borak > 2.txt
$ cat 2.txt
Borak
$ tee 1.txt
PLEASE TELL ME WHAT TO WRITE IN 1.txt AND IN STANDARD OUTPUT:
OS rocks!
SUCCESFULY READ FROM STANDARD INPUT
OS rocks!
SUCCESFULY WROTE TO STANDARD OUTPUT
SUCCESFULY WROTE TO FILE 1.txt
$ cat 1.txt
OS rocks!
$ tee 1.txt 2.txt
SUCCESFULY READ FROM 1.txt
SUCCESFULY WROTE TO 2.txt
$ cat 2.txt
OS rocks!
$ tee 3.txt
PLEASE TELL ME WHAT TO WRITE IN 3.txt AND IN STANDARD OUTPUT:
I'm tired
SUCCESFULY READ FROM STANDARD INPUT
I'm tired
SUCCESFULY WROTE TO STANDARD OUTPUT
SUCCESFULY WROTE TO FILE 3.txt
$ cat 3.txt
I'm tired
$
```

Program Code:

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

int main(int argc, char *argv[]){

    //initialization of buffer for input/output
    char buf[100];
    memset(buf,0,100);

    //if only 1 argument was given
    if(argc == 2){

        //open the file
        int fileoutput_fd = open(argv[1], O_WRONLY|O_CREATE);
        printf(1,"PLEASE TELL ME WHAT TO WRITE IN %s AND IN STANDARD OUTPUT:\n", argv[1]);

        //get input from Standard input(fd=0)
        if(read(0,buf,100) > 0){
            printf(1,"SUCCESFULY READ FROM STANDARD INPUT\n");
            buf[strlen(buf)] = 0;

            //write to standard output(fd=1)
            write(1,buf,strlen(buf));
            printf(1,"SUCCESFULY WROTE TO STANDARD OUTPUT\n");

            //write to file(fd = fileoutput_fd)
            write(fileoutput_fd,buf,strlen(buf));
            printf(1,"SUCCESFULY WROTE TO FILE %s\n",argv[1]);

            //close file and exit
            close(fileoutput_fd);
            exit(0);
        }

        //if read from stanard input failed print error message, close file and exit
        printf(2,"READING ERROR\n");
        close(fileoutput_fd);
        exit(0);

    }
```

```c
    //if 2 arguments were given
    else if(argc == 3){

        //open files for reading/writing
        int fileinput_fd = open(argv[1], O_RDONLY);
        int fileoutput_fd = open(argv[2], O_WRONLY|O_CREATE);

        //get input from file(fd = fileinput_fd)
        if(read(fileinput_fd,buf,100) > 0){
            printf(1,"SUCCESFULY READ FROM %s\n",argv[1]);
            buf[strlen(buf)] = 0;

            //write to other file(fd = fileoutput_fd)
            write(fileoutput_fd,buf,strlen(buf));
            printf(1,"SUCCESFULY WROTE TO %s\n",argv[2]);

            //close both files and exit
            close(fileoutput_fd);
            close(fileinput_fd);
            exit(0);
        }

        //if read from file 1 failed print error message, close files and exit
        printf(2,"READING ERROR\n");
        close(fileoutput_fd);
        close(fileinput_fd);
        exit(0);
    }

    //if not 1 or 2 arguments were given, print error message
    printf(2,"Usage is: tee FILE1 or tee FILE1 FILE2\n");
    exit(0);
}
```

Adding A "getpinfo" System Call

Shell Output:

```
Booting from Hard Disk...
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ getpinfo
Row Number: 0           Process ID: 1           Process Name:init
Row Number: 1           Process ID: 2           Process Name:sh
$ init
init: starting sh
$ getpinfo
Row Number: 0           Process ID: 1           Process Name:init
Row Number: 1           Process ID: 2           Process Name:sh
Row Number: 2           Process ID: 3           Process Name:init
Row Number: 3           Process ID: 4           Process Name:sh
$
```

System Call Code:

proc.c

```c
//---------------------------------------------------------------------------------------------------------
//getpinfo system call: print all currently used processes(not UNUSED)
//---------------------------------------------------------------------------------------------------------
void getpinfo(void){

    struct proc *p;
    int i = 0;

    //lock the ptable so it won't change while printing
    acquire(&ptable.lock);

    //for evety process which is not UNUSED, print its details
    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++)
        if(p->state != UNUSED)
            cprintf("Row Number: %d \t\tProcess ID: %d\t\tProcess Name:%s\n", i++, p->pid, p->name);

    //release the ptable
    release(&ptable.lock);

}
//---------------------------------------------------------------------------------------------------------
```

Changing The Wait And Exit System Calls

Shell Output:

```
Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ wait_test
Let's do a fork and see what parent recieves.
I'm a child! And I'm exiting with status code 24!
I'm a parent, and I got child status: 24
$
```

System Call Code - Exit:

```
//---------------------------------------------------
//change process' exit status field to hold the status
//---------------------------------------------------
curproc->exitStatus = status;
//---------------------------------------------------
```

System Call Code - Wait:

```
//-----------------------------------------------------------------------
//if status is not a null pointer, update its value with the returned status
//-----------------------------------------------------------------------
if(status!=0)
  *status = p->exitStatus;
//-----------------------------------------------------------------------
```

Test Code:

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

int main(int argc, char *argv[])
{
    printf(1, "Let's do a fork and see what parent recieves.\n");
    int pid = fork();
    if(pid == 0){
        printf(1, "I'm a child! And I'm exiting with status code 24!\n");
        exit(24);
    } else if(pid < 0){
        printf (1, "This fork has failed\n");
        exit(0);
    }
    int status;
    wait(&status);
    printf(1, "I'm a parent, and I got child status: %d\n", status);
    exit(0);
}
```