Yousef Jarrar, Jose Perez

CSE 461

Lab 1

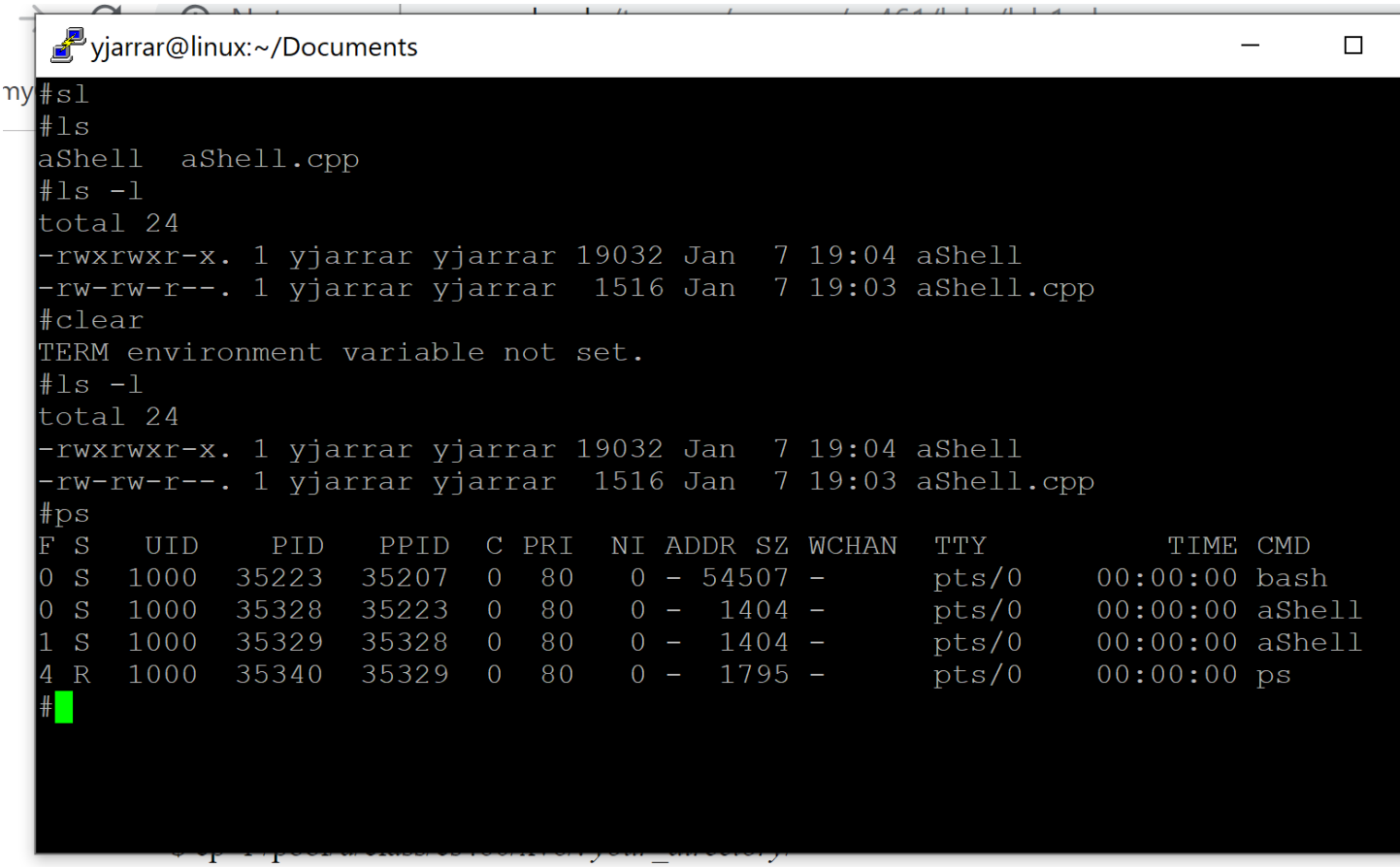
20 Total Points.

1. Writing a Simple Shell

```
/*Yousef Jarrar - Jose Perez
Dr. Tong Yu
Part 1 - Writing a Simple Shell
Lab 1
*/
#include < stdio.h >
#include < stdlib.h >
#include < sys / wait.h >
#include < unistd.h >
#include < string.h >
 void read_command(char cmd[], char * par[]) {
   char line[1024];
   int count = 0, i = 0, j = 0;
   char * array[100], * pch;

   for (;;) {
     int c = fgetc(stdin);
     line[count++] = (char) c;
     if (c == '\n') break;
   }

   if (count == 1) return;
   pch = strtok(line, " \n");
```

```
    //Parse the line into words
    while (pch != NULL) {
      array[i++] = strdup(pch);
      pch = strtok(NULL, " \n");
    }

    //first word is the command
    strcpy(cmd, array[0]);

    for (int j = 0; j < i; j++)
      par[j] = array[j];
    par[i] = NULL;
  }

void type_prompt() {
  static int first_time = 1;
  if (first_time) {
    const char * CLEAR_SCREEN_ANSI = " \e[1;1H\e[2J";
    write(STDOUT_FILENO, CLEAR_SCREEN_ANSI, 12);
    first_time = 0;
  }

  printf("#");
}

int main() {
  char cmd[100], command[100], * parameters[20];
  char * envp[] = {
```

```
    (char * )
    "PATH=/bin",
    0
  };
  while (1) {
    type_prompt();
    read_command(command, parameters);
    if (fork() != 0) {
      wait(NULL);
    } else {
      strcpy(cmd, "/bin/");
      strcat(cmd, command);
      execve(cmd, parameters, envp);
    }

    if (strcmp(command, "exit") == 0) {
      break;
    }
  }
  return 0;
}
```



3. Debugging

Part A:

[yjarrar@linux xv6-public]$ gdb

GNU gdb (GDB) Fedora 8.2-5.fc29

Copyright (C) 2018 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law.

Type "show copying" and "show warranty" for details.

This GDB was configured as "x86_64-redhat-linux-gnu".

Type "show configuration" for configuration details.

For bug reporting instructions, please see:

<http://www.gnu.org/software/gdb/bugs/>.

```
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
warning: File "/home/yjarrar/Documents/Lab 1/xv6-public/.gdbinit" auto-loading .
To enable execution of this file add
        add-auto-load-safe-path /home/yjarrar/Documents/Lab 1/xv6-public/.gdbint
line to your configuration file "/home/yjarrar/.gdbinit".
To completely disable this security protection add
        set auto-load safe-path /
line to your configuration file "/home/yjarrar/.gdbinit".
For more information about this security protection see the
--Type <RET> for more, q to quit, c to continue without paging--c
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
        info "(gdb)Auto-loading safe path"
(gdb) target remote:26000
Remote debugging using :26000
warning: Remote gdbserver does not support determining executable automatically.
RHEL <=6.8 and <=7.2 versions of gdbserver do not support such automatic executable
detection.
The following versions of gdbserver support it:
- Upstream version of gdbserver (unsupported) 7.10 or later
- Red Hat Developer Toolset (DTS) version of gdbserver from DTS 4.0 or later (only on
x86_64)
- RHEL-7.3 versions of gdbserver (on any architecture)
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x0000fff0 in ?? ()
(gdb) file kernel
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from kernel...done.
(gdb) break swtch
Breakpoint 1 at 0x8010479b: file swtch.S, line 11.
```

```
(gdb) continue
Continuing.

Thread 1 hit Breakpoint 1, swtch () at swtch.S:11
11        movl 4(%esp), %eax
(gdb) step
12        movl 8(%esp), %edx
(gdb) step
15        pushl %ebp
(gdb) step
swtch () at swtch.S:16
16        pushl %ebx
(gdb) step
swtch () at swtch.S:17
17        pushl %esi
(gdb) step
swtch () at swtch.S:18
18        pushl %edi
(gdb) steo
Undefined command: "steo".  Try "help".
(gdb) step
swtch () at swtch.S:21
21        movl %esp, (%eax)
(gdb) step
22        movl %edx, %esp
(gdb) step
swtch () at swtch.S:25
25        popl %edi
(gdb) step
swtch () at swtch.S:26
26        popl %esi
(gdb) step
swtch () at swtch.S:27
27        popl %ebx
```

```
(gdb) step
swtch () at swtch.S:28
28        popl %ebp
(gdb) step
swtch () at swtch.S:29
29        ret
(gdb) step
forkret () at proc.c:401
401       release(&ptable.lock);
(gdb) step
release (lk=0x80112d20 <ptable>) at spinlock.c:49
49        if(!holding(lk))
(gdb) step
holding (lock=0x80112d20 <ptable>) at spinlock.c:94
94        r = lock->locked && lock->cpu == mycpu();
(gdb) step
mycpu () at x86.h:99
99        return eflags;
(gdb) step
45        apicid = lapicid();
(gdb) step
lapicid () at lapic.c:103
103       if (!lapic)
(gdb) step
105       return lapic[ID] >> 24;
(gdb) step
mycpu () at proc.c:48
48        for (i = 0; i < ncpu; ++i) {
(gdb) step
49          if (cpus[i].apicid == apicid)
(gdb) step
50            return &cpus[i];
(gdb) step
popcli () at x86.h:99
```

```
99        return eflags;
(gdb) step
121       if(--mycpu()->ncli < 0)
(gdb) stpe
Undefined command: "stpe".  Try "help".
(gdb) step
mycpu () at x86.h:99
99        return eflags;
(gdb) step
45        apicid = lapicid();
(gdb) continue
Continuing.

Thread 1 hit Breakpoint 1, swtch () at swtch.S:11
11        movl 4(%esp), %eax
(gdb) clear
Deleted breakpoint 1
(gdb) break exec
Breakpoint 2 at 0x80100a80: file exec.c, line 20.
(gdb) continue
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 2, exec (path=0x1c "/init", argv=0x8dfffed0) at exec.c:20
20        struct proc *curproc = myproc();
(gdb) continue
Continuing.

Thread 2 hit Breakpoint 2, exec (path=0x846 "sh", argv=0x8dffeed0) at exec.c:20
20        struct proc *curproc = myproc();
(gdb) continue
Continuing.
[Switching to Thread 1]
```

Thread 1 hit Breakpoint 2, exec (path=0x18e0 "ls", argv=0x8dfbeed0) at exec.c:20

*20   struct proc *curproc = myproc();*

(gdb)

Explanation: We started off with running "xv6" in a terminal. We then opened a second session of the terminal command line and started a gbd connection on port 26000. We then loaded the Kernel File and inserted a Breakpoint (context switching) to start the debugging process. We were able to determine the different points of how the xv6 boots, and observed the different opcodes that were being used during the process. Once completed, we cleared the breakpoint and inserted another (exec system call). We noticed that at this point the kernel loaded it's first user-mode process "init". It then loads into an interactive shell program and stops as it is waiting for a command (we noticed this on the operating system side, opposite of gdb). Once it was supplied with a system call "ls -l" the operating system continued to work, until the next hang (user input).

Part B:

(gdb) disass

Dump of assembler code for function mycpu:

0x80103840 <+0>: push ebp

0x80103841 <+1>: mov ebp,esp

0x80103843 <+3>: push esi

0x80103844 <+4>: push ebx

0x80103845 <+5>: pushf

0x80103846 <+6>: pop eax

0x80103847 <+7>: test ah,0x2

0x8010384a <+10>: jne 0x801038a9 <mycpu+105>

0x8010384c <+12>: call 0x80102830 <lapicid>

=> 0x80103851 <+17>: mov esi,DWORD PTR ds:0x80112d00

0x80103857 <+23>: test esi,esi

0x80103859 <+25>: jle 0x8010389c <mycpu+92>

0x8010385b <+27>: movzx edx,BYTE PTR ds:0x80112780

0x80103862 <+34>: cmp eax,edx

0x80103864 <+36>: je 0x80103895 <mycpu+85>

0x80103866 <+38>: xor edx,edx

*0x80103868 <+40>: lea esi,[esi+eiz*1+0x0]*

0x8010386f <+47>: nop

0x80103870 <+48>: add edx,0x1

0x80103873 <+51>: cmp edx,esi

0x80103875 <+53>: je 0x8010389c <mycpu+92>

0x80103877 <+55>: imul ecx,edx,0xb0

--Type <RET> for more, q to quit, c to continue without paging--c

0x8010387d <+61>: movzx ebx,BYTE PTR [ecx-0x7feed880]

0x80103884 <+68>: cmp ebx,eax

0x80103886 <+70>: jne 0x80103870 <mycpu+48>

0x80103888 <+72>: lea eax,[ecx-0x7feed880]

0x8010388e <+78>: lea esp,[ebp-0x8]

0x80103891 <+81>: pop ebx

0x80103892 <+82>: pop esi

0x80103893 <+83>: pop ebp

0x80103894 <+84>: ret

0x80103895 <+85>: mov eax,0x80112780

0x8010389a <+90>: jmp 0x8010388e <mycpu+78>

0x8010389c <+92>: sub esp,0xc

0x8010389f <+95>: push 0x801074dc

0x801038a4 <+100>: call 0x80100390 <panic>

0x801038a9 <+105>: sub esp,0xc

0x801038ac <+108>: push 0x801075b8

0x801038b1 <+113>: call 0x80100390 <panic>

End of assembler dump.

Part C:

```
/*****************************************************************************************
 * cp.c
 * By: Yousef Jarrar and Jose Perez
 * CSE 461 Lab 1

*****************************************************************************************/
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

char buf[512];
int main(int argc, char *argv[]) {
  int fd0, fd1, fd2, n;
  if (argv <= 2) {
```

```
  printf(1, "Need 2 Arguements!\n");
  exit();
}

for (int i =2; i<=argc; i++){
  //opens README file
  if((fd0 = open(argv[1], O_RDONLY))<0) {
    printf(1, "cp: Cannot Open %s\n", argv[1]);
    exit();
  }

  //opens myFile1
  if((fd1 = open(argv[2], O_CREATE|O_RDWR))<0) {
    printf(1, "cp: Cannot Open %s\n", argv[2]);
    exit();
  }

  //opens the myFile2
  if((fd2 = open(argv[3], O_CREATE|O_RDWR))<0) {
    printf(1, "cp: Cannot Open %s\n", argv[3]);
    exit();
  }

  while((n = read ( fd0, buf, sizeof(buf)))>0){
    //writes onto the myFiles
    write (fd1, buf, n);
    write (fd2, buf, n);
  }
  //closes README File
  close(fd0);
```

```
  //closes myFile1 and myFile2
   close(fd1);
   close(fd2);
  }
  exit();
}
```

```
iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF913E0+1FED13E0 C980



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 sta8
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2327
cat            2 3 16164
echo           2 4 14980
forktest       2 5 9260
grep           2 6 18432
init           2 7 15600
kill           2 8 15016
ln             2 9 14900
ls             2 10 17504
mkdir          2 11 15132
rm             2 12 15108
sh             2 13 27668
stressfs       2 14 15952
usertests      2 15 65948
wc             2 16 16856
cp             2 17 15824
zombie         2 18 14716
console        3 19 0
myFile         2 20 2327
```

```
$ cp
Need 2 Arguements!
$ cp README myFile1 myFile2
$ ls
.               1 1 512
..              1 1 512
README          2 2 2327
cat             2 3 16164
echo            2 4 14980
forktest        2 5 9260
grep            2 6 18432
init            2 7 15600
kill            2 8 15016
ln              2 9 14900
ls              2 10 17504
mkdir           2 11 15132
rm              2 12 15108
sh              2 13 27668
stressfs        2 14 15952
usertests       2 15 65948
wc              2 16 16856
cp              2 17 15824
zombie          2 18 14716
console         3 19 0
myFile          2 20 2327
myFile1         2 21 2327
myFile2         2 22 2327
$ cat myFile2
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.

ACKNOWLEDGMENTS

xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
```

yjarrar@linux:~/Documents/Lab 1/xv6-public

```
$ $
$
$
$
$
$
$
$
$
$
$ cat myFile1
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.

ACKNOWLEDGMENTS

xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14,
2000)). See also https://pdos.csail.mit.edu/6.828/, which
provides pointers to on-line resources for v6.

xv6 borrows code from the following sources:
    JOS (asm.h, elf.h, mmu.h, bootasm.S, ide.c, console.c, and others)
    Plan 9 (entryother.S, mp.h, mp.c, lapic.c)
    FreeBSD (ioapic.c)
    NetBSD (console.c)

The following people have made contributions: Russ Cox (context switching,
locking), Cliff Frey (MP), Xiao Yu (MP), Nickolai Zeldovich, and Austin
Clements.

We are also grateful for the bug reports and patches contributed by Silas
Boyd-Wickizer, Anton Burtsev, Cody Cutler, Mike CAT, Tej Chajed, eyalz800,
Nelson Elhage, Saar Ettinger, Alice Ferrazzi, Nathaniel Filardo, Peter
Froehlich, Yakir Goaron,Shivam Handa, Bryan Henry, Jim Huang, Alexander
Kapshuk, Anders Kaseorg, kehao95, Wolfgang Keller, Eddie Kohler, Austin
Liew, Imbar Marinescu, Yandong Mao, Matan Shabtay, Hitoshi Mitake, Carmi
Merimovich, Mark Morrissey, mtasm, Joel Nider, Greg Price, Ayan Shafqat,
Eldar Sehayek, Yongming Shen, Cam Tenny, tyfkda, Rafael Ubal, Warren
Toomey, Stephen Tu, Pablo Ventura, Xi Wang, Keiichi Watanabe, Nicolas
Wolovick, wxdao, Grant Wu, Jindong Zhang, Icenowy Zheng, and Zou Chang Wei

The code in the files that constitute xv6 is
Copyright 2006-2018 Frans Kaashoek, Robert Morris, and Russ Cox.

ERROR REPORTS

Please send errors and suggestions to Frans Kaashoek and Robert Morris
(kaashoek,rtm@mit.edu). The main purpose of xv6 is as a teaching
operating system for MIT's 6.828, so we are more interested in
```