**OS- Assignment-1**

**Group Members:**

Vani Pailla : U95251377

Asha Tummuru : U38611026

**Environment:**

The following procedures were employed in the development and testing of the code: The code was developed and tested on Ubuntu 22.04, a Linux-based OS, utilizing GCC (version 11.3.0) for compilation. The build process was managed using a Makefile, and QEMU was utilized for virtualization when testing the xv6 operating system. While the majority of the development work was conducted using Visual Studio Code with C extensions, Vim was occasionally used. GDB served as the primary debugging tool, with bochs utilized in select cases to simulate the operating system.

**Part-1: Adding Hello World Program**

After creating a file named hello.c, I added code to print "hello xv6" from the user end. Subsequently, I included the essential dependencies in the makefile to enable the conversion of the object file into an executable file.

Change in Makefile

- The file name will be included in the UPROGS section.

**UPROGS=\**

**\_cat\**

**\_echo\**

**\_forktest\**

**\_grep\**

**\_init\**

**\_kill\**

**\_ln\**

**\_ls\**

**\_mkdir\**

**\_rm\**

**\_sh\**

**\_stressfs\**

**\_usertests\**

**\_wc\**

**\_zombie\**

**\_hello\**

- The file name has been included in the EXTRA section as well.

EXTRA=\

mkfs.c ulib.c user.h cat.c echo.c forktest.c grep.c kill.c\

ln.c ls.c mkdir.c rm.c stressfs.c usertests.c wc.c zombie.c\

printf.c umalloc.c hello.c \

README dot-bochsrc \*.pl toc.\* runoff runoff1 runoff.list\

.gdbinit.tmpl gdbutil\

Input: here I have passing the user as command line argument to differentiate the user hello and kernel hello , in part -3 I shown the kernel hello.

**hello user**

output:

A black background with white text

Description automatically generated

Part-2: Modifying Current ls System Call

In the ls.c file, it's necessary to make a modification. When the ls command is used with the -a option, it should display the hidden files. If only the ls command is used, it should display the files in the current directory.

I've integrated logic into the command line arguments to accommodate the -a option. Additionally, I've initialized the hidden variable with a default value of “N”.

**for (i = 1; i < argc; i++)**

**{**

**hidden = (strcmp(argv[i], "-a") == 0) ? 'Y' : hidden;**

**ls((strcmp(argv[i], "-a") == 0) ? "." : argv[i], hidden);**

**}**

I've come across some empty spaces in the file names. To resolve this, I've been iterating through each character in the file name and replacing the first space with a backslash.

**if (st.type == T\_DIR && hidden == 'Y'){**

**char \*space\_pos = strchr(formatted\_name, ' ');**

**if (space\_pos != 0 {**

**\*space\_pos = '/';**

**}}**

**printf(1, "%s %d %d %d\n", formatted\_name, st.type, st.ino, st.size);**

Input**: ls**  
Output:

A screen shot of a computer

Description automatically generated

Input : ls -a

Output:

A screen shot of a computer

Description automatically generated

**Part-3 Writing Hello World System Call**

When incorporating the hello system call in xv6, it is crucial to update multiple files to include the required dependencies. These files comprise syscall.h, syscall.c, proc.c, uSYS.s, and user.h. This guarantees the seamless integration of the system call, ensuring that it operates as intended within the xv6 operating system.

Change in the files:  
syscall.h:

…  
#define SYS\_mknod 17

#define SYS\_unlink 18

#define SYS\_link 19

#define SYS\_mkdir 20

#define SYS\_close 21

#define SYS\_hello 22

The highlighted part I have added in the syscall.h

Syscall.c:

…  
extern int sys\_unlink(void);

extern int sys\_wait(void);

extern int sys\_write(void);

extern int sys\_uptime(void);

extern int sys\_hello(void);

static int (\*syscalls[])(void) = {

[SYS\_fork] sys\_fork,

[SYS\_exit] sys\_exit,

[SYS\_wait] sys\_wait,

[SYS\_pipe] sys\_pipe,

…

[SYS\_link] sys\_link,

[SYS\_mkdir] sys\_mkdir,

[SYS\_close] sys\_close,

[SYS\_hello] sys\_hello,

};

The highlighted part I have added in the syscall.c file

Proc.c:  
  
here I have implemented the hello system call.

***int sys\_hello(void){***

***cprintf("Hello from the Kernel!");***

***return 0;***

***}***

**Usys.s:**

…

SYSCALL(chdir)

SYSCALL(dup)

SYSCALL(getpid)

SYSCALL(sbrk)

SYSCALL(sleep)

SYSCALL(uptime)

SYSCALL(hello)

The highlighted part of the code I have added in the existing part of code.

User.h:

…  
int getpid(void);

char\* sbrk(int);

int sleep(int);

int uptime(void);

int hello(void);

The highlighted part of the code I have added in the existing part of code.

Input: hello kernel

Output:



**Part-4: Implementing the sleep command**

I have written a file named sleep.c, which includes a function utilizing the system call sleep(). This system call enables the program to halt for a specified number of clock ticks, as determined by the user.

In the Makefile, it's crucial to set the file name at two specific locations: under Uprogs and Extras. These configurations are essential and cannot be overlooked.

Input: sleep 50

Output: it will sleep for 50 ticks.



Make :A computer screen with text

Description automatically generated

Make clean:

A computer screen with white text

Description automatically generated

Make qemu:

A screenshot of a computer screen

Description automatically generated

A screen shot of a computer screen

Description automatically generated

A computer screen shot of a computer code

Description automatically generated