Project 1

1 The explicit system environment used to develop/test code

sw_vers

```
ProductName:     macOS
ProductVersion:  15.6.1
BuildVersion:    24G90
```

xcode –version

```
xcode-select version 2410.
```

clang –version

```
Apple clang version 16.0.0 (clang-1600.0.26.6)
Target: x86_64-apple-darwin24.6.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

make –version

```
GNU Make 3.81
Copyright (C) 2006  Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE.

This program built for i386-apple-darwin11.3.0
```

Shell

```
/bin/zsh
```

2 Additional Resources

I used the following resources to complete this project:

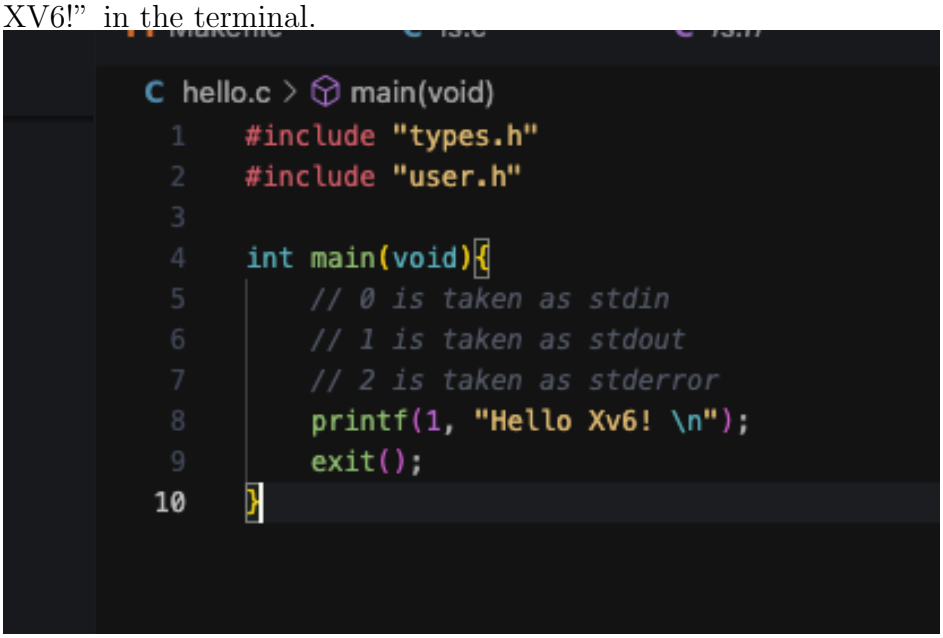
- Add a User Program in XV6
- Getting the process name and printing it out in XV6/C
- How to make a file appear to ls in xv6 qemu
- Google code generation: I copied the bubble sort implementation for sorting files based on size from google-generated code.

3 Additional Information

The screenshots for each task are added to the Screenshots folder.

Hello

- **hello-code.png**: This is the code implementation of hello.c file to display "Hello XV6!" in the terminal.



```
hello.c > main(void)
#include "types.h"
#include "user.h"

int main(void){
    // 0 is taken as stdin
    // 1 is taken as stdout
    // 2 is taken as stderror
    printf(1, "Hello Xv6! \n");
    exit();
}
```

- **hello-output.png**: Screenshot of the output in the xv6 terminal when `hello` call is made.

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

reeshavacharya@MacBookPro xv6-public % make qemu
i386-elf-gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie   -c -o ls.o ls.c
i386-elf-ld -m    elf_i386 -N -e main -Ttext 0 -o _ls ls.o ulib.o usys.o printf.o umalloc.o
i386-elf-ld: warning: _ls has a LOAD segment with RWX permissions
i386-elf-objdump -S _ls > ls.asm
i386-elf-objdump -t _ls | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > ls.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _hello _clear
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 682 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp 2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ hello
Hello Xv6!
$ 
```

ls

- **Hide files and directories**: I checked the first character of the file/folder name. If it was not equal to '.', print details as the original implementation. To test:

```
echo "this file is hidden" > .hidden_file
ls
```

The hidden file did not show up.

```
switch(st.type){
case T_FILE:
  printf(1, "%s %d %d %d\n", fmtname(path), st.type, st.ino, st.size);
  break;

case T_DIR:
  if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
    printf(1, "ls: path too long\n");
    break;
  }
  strcpy(buf, path);
  p = buf+strlen(buf);
  *p++ = '/';
  while(read(fd, &de, sizeof(de)) == sizeof(de)){
    if(de.inum == 0)
      continue;
    memmove(p, de.name, DIRSIZ);
    p[DIRSIZ] = 0;
    if(stat(buf, &st) < 0){
      printf(1, "ls: cannot stat %s\n", buf);
      continue;
    }
    if(fmtname(buf)[0]!='.'){
      if(st.type==1){
        printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
      }else {
        printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
      }
    }
  }
  break;
}
```

```
$ echo "this file is hidden" > .hidden_file
$ ls
README         2 2 2286
cat            2 3 14944
echo           2 4 13904
forktest       2 5 8440
grep           2 6 17644
init           2 7 14508
kill           2 8 13940
ln             2 9 13868
ls             2 10 16504
mkdir          2 11 13992
rm             2 12 13972
sh             2 13 27404
stressfs       2 14 14748
usertests      2 15 61104
wc             2 16 15296
zombie         2 17 13548
hello          2 18 13512
clear          2 19 13856
console        3 20 0
$
```

- **Add '/' to folder names for easy identification**: Implemented `format_directory_name(char *ftname)` to handle padded names and add '/'. To test:

```
mkdir new_folder
ls
```

Output: `new_folder/`

```c

char *format_directory_name(char *fmtname){
  for(int i = DIRSIZ-1; i>=0; i--){
    if(fmtname[i]!=' '){
      fmtname[i+1]='/';
      break;
    }
  }
  return fmtname;
}

```

```c
case T_DIR:
  if(strlen(path) + 1 + DIRSIZ + 1 > sizeof buf){
    printf(1, "ls: path too long\n");
    break;
  }
  strcpy(buf, path);
  p = buf+strlen(buf);
  *p++ = '/';
  while(read(fd, &de, sizeof(de)) == sizeof(de)){
    if(de.inum == 0)
      continue;
    memmove(p, de.name, DIRSIZ);
    p[DIRSIZ] = 0;
    if(stat(buf, &st) < 0){          (char [20])"ls: cannot stat %s\n"
      printf(1, "ls: cannot stat %s\n", buf);
      continue;
    }
    if(fmtname(buf)[0]!='.'){
      if(st.type==1){
        printf(1, "%s %d %d %d\n", format_directory_name(fmtname(buf)), st.type, st.ino, st.size);
      }else {
        printf(1, "%s %d %d %d\n", fmtname(buf), st.type, st.ino, st.size);
      }
    }
  }
  break;
```

```
QEMU
init: starting sh
$ mkdir new_folder
$ ls
README          2 2 2286
cat             2 3 14944
echo            2 4 13904
forktest        2 5 8440
grep            2 6 17644
init            2 7 14508
kill            2 8 13940
ln              2 9 13868
ls              2 10 17260
mkdir           2 11 13992
rm              2 12 13972
sh              2 13 27404
stressfs        2 14 14748
usertests       2 15 61104
wc              2 16 15296
zombie          2 17 13548
hello           2 18 13512
clear           2 19 13856
console         3 20 0
new_folder/     1 21 32
$
```

- **Sort files based on size**: Added `-s` flag parsing in `ls.c`. Used a `file_entry` struct and bubble sort.

```c
int main(int argc, char *argv[])
{
  int i;
  bool sort_size = false;

  for(int i = 1; i<argc; i++){
    if(argv[i][0] == '-' && argv[i][1] == 's' && argv[i][2] == '\0'){
      sort_size=true;
      break;
    }
  }

  if(argc < 2 || (argc ==2 && sort_size)){
    ls(".", sort_size);
    exit();
  }

  for(i=1; i<argc; i++){
    if(argv[i][0] == '-' && argv[i][1] == 's' && argv[i][2] == '\0'){
      continue;
    }
    ls(argv[i], sort_size);
  }

  exit();
}
```

```c
struct file_entry{
  char name[DIRSIZ+1];
  struct stat st;
};
```

```c
void swap_file_entry(struct file_entry *a, struct file_entry *b){
  struct file_entry temp = *a;
  *a = *b;
  *b = temp;
}

void bubbleSortFiles(struct file_entry files[], int n) {
  for(int i = 0; i < n - 1; i++){
      int swapped = 0;
      for(int j = 0; j < n - i - 1; j++){
        if(files[j].st.size < files[j + 1].st.size){
            swap_file_entry(&files[j], &files[j + 1]);
            swapped = 1;
        }
      }
      if(swapped == 0){
        break;
      }
  }
}
```

```c
void print_files_sorted(){
  for (int i = 0; i< 100; i++){
    if(files[i].st.size!=0){
      printf(1, "%s %d %d %d\n", files[i].name, files[i].st.type, files[i].st.ino, files[i].st.size);
    }
  }
}
```

- **Show file extension**:

```
echo > new_program.c
ls -s
```

It correctly displayed `new_program.c`.

```
init: starting sh
$ echo > new_program.c
$ ls -s
usertests      2 15 61232
sh             2 13 27536
ls             2 10 20672
grep           2 6 17772
wc             2 16 15424
cat            2 3 15072
stressfs       2 14 14876
init           2 7 14640
mkdir          2 11 14116
rm             2 12 14096
kill           2 8 14068
echo           2 4 14032
sleep          2 19 14020
ln             2 9 13996
clear          2 20 13980
zombie         2 17 13676
hello          2 18 13540
forktest       2 5 8568
README         2 2 2286
console        3 21 0
new_program.c  2 22 0
```

Hello SysCall

Implemented new syscall `sys_hello` (system call #22). Added in `user.h`, modules, and tested.

```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ hello
Hello from Kernel Mode!
$
```

```c
int
sys_hello(void){
  cprintf("Hello from Kernel Mode!\n");
  return 0;
}

int
sys_getpid(void)
{
  return myproc()->pid;
}
```

```c
static int (*syscalls[])(void) = {
[SYS_fork]    sys_fork,
[SYS_exit]    sys_exit,
[SYS_wait]    sys_wait,
[SYS_pipe]    sys_pipe,
[SYS_read]    sys_read,
[SYS_kill]    sys_kill,
[SYS_exec]    sys_exec,
[SYS_fstat]   sys_fstat,
[SYS_chdir]   sys_chdir,
[SYS_dup]     sys_dup,
[SYS_hello]   sys_hello,
[SYS_getpid]  sys_getpid,
[SYS_sbrk]    sys_sbrk,
[SYS_sleep]   sys_sleep,
[SYS_uptime]  sys_uptime,
[SYS_open]    sys_open,
[SYS_write]   sys_write,
[SYS_mknod]   sys_mknod,
[SYS_unlink]  sys_unlink,
[SYS_link]    sys_link,
[SYS_mkdir]   sys_mkdir,
[SYS_close]   sys_close,
};
```

```
extern int sys_chdir(void);
extern int sys_close(void);
extern int sys_dup(void);
extern int sys_exec(void);
extern int sys_exit(void);
extern int sys_fork(void);
extern int sys_fstat(void);
extern int sys_hello(void);
extern int sys_getpid(void);
extern int sys_kill(void);
extern int sys_link(void);
extern int sys_mkdir(void);
extern int sys_mknod(void);
extern int sys_open(void);
extern int sys_pipe(void);
extern int sys_read(void);
extern int sys_sbrk(void);
extern int sys_sleep(void);
extern int sys_unlink(void);
extern int sys_wait(void);
extern int sys_write(void);
extern int sys_uptime(void);
```

```c
// System call numbers
#define SYS_fork    1
#define SYS_exit    2
#define SYS_wait    3
#define SYS_pipe    4
#define SYS_read    5
#define SYS_kill    6
#define SYS_exec    7
#define SYS_fstat   8
#define SYS_chdir   9
#define SYS_dup    10
#define SYS_hello  22
#define SYS_getpid 11
#define SYS_sbrk   12
#define SYS_sleep  13
#define SYS_uptime 14
#define SYS_open   15
#define SYS_write  16
#define SYS_mknod  17
#define SYS_unlink 18
#define SYS_link   19
#define SYS_mkdir  20
#define SYS_close  21
```

```c
struct stat;
struct rtcdate;

// system calls
int fork(void);
int exit(void) __attribute__((noreturn));
int wait(void);
int pipe(int*);
int write(int, const void*, int);
int read(int, void*, int);
int close(int);
int kill(int);
int exec(char*, char**);
int open(const char*, int);
int mknod(const char*, short, short);
int unlink(const char*);
int fstat(int fd, struct stat*);
int link(const char*, const char*);
int mkdir(const char*);
int chdir(const char*);
int dup(int);
int getpid(void);
int hello(void);
char* sbrk(int);
int sleep(int);
int uptime(void);
```

```
#include "syscall.h"
#include "traps.h"

#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret

SYSCALL(fork)
SYSCALL(exit)
SYSCALL(wait)
SYSCALL(pipe)
SYSCALL(read)
SYSCALL(write)
SYSCALL(close)
SYSCALL(kill)
SYSCALL(exec)
SYSCALL(open)
SYSCALL(mknod)
SYSCALL(unlink)
SYSCALL(fstat)
SYSCALL(link)
SYSCALL(mkdir)
SYSCALL(chdir)
SYSCALL(dup)
SYSCALL(getpid)
SYSCALL(hello)
SYSCALL(sbrk)
SYSCALL(sleep)
SYSCALL(uptime)
```

Sleep

Created `sleep.c` which checks for an argument and calls existing `sleep` from `user.h`. Tested for 200 ticks successfully.

```
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 sleep.c clear.c\
    README dot-bochsrc *.pl toc.* runoff runoff1 runoff.list\
    .gdbinit.tmpl gdbutil\
```

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _hello\
    _sleep\
    _clear\
```

```
$ sleep
Usage: sleep </ticks>
$ sleep 200
```

```c
#include "user.h"

int main(int argc, char *argv[]){
    if(argc <2){
        printf(1, "Usage: sleep </ticks>\n");
    }
    int ticks = atoi(argv[1]);
    sleep(ticks);
    exit();
}
```