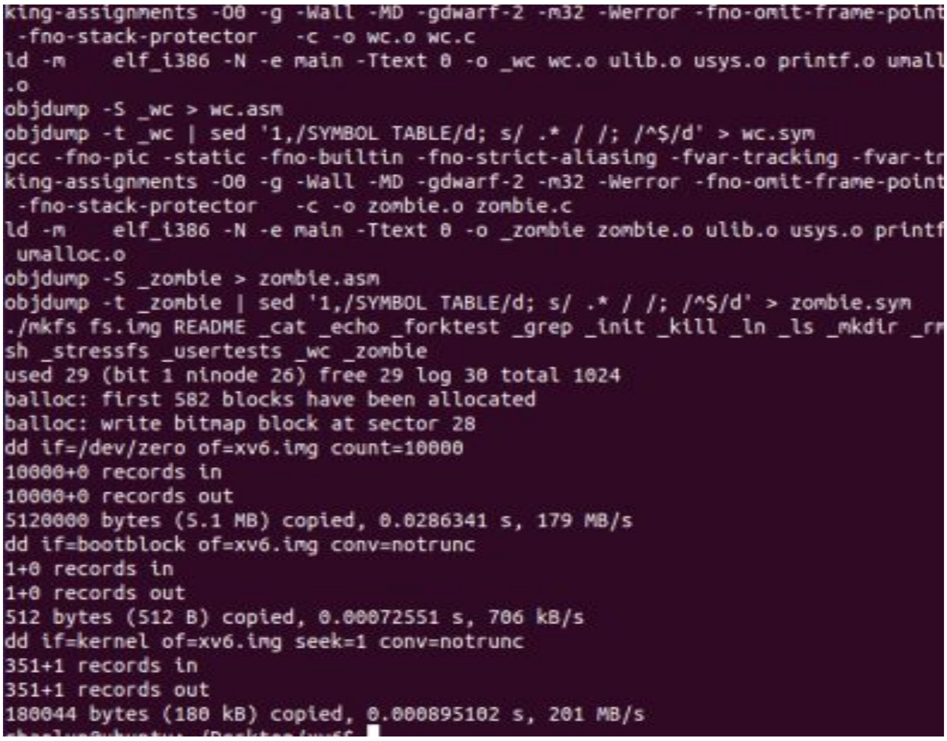
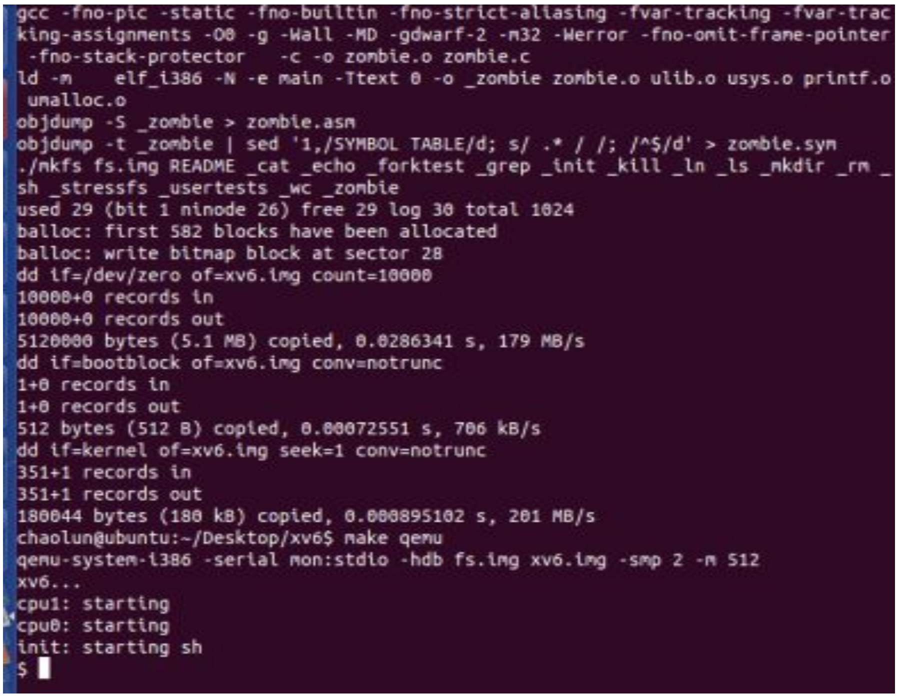
Project 1 document

Part 1:



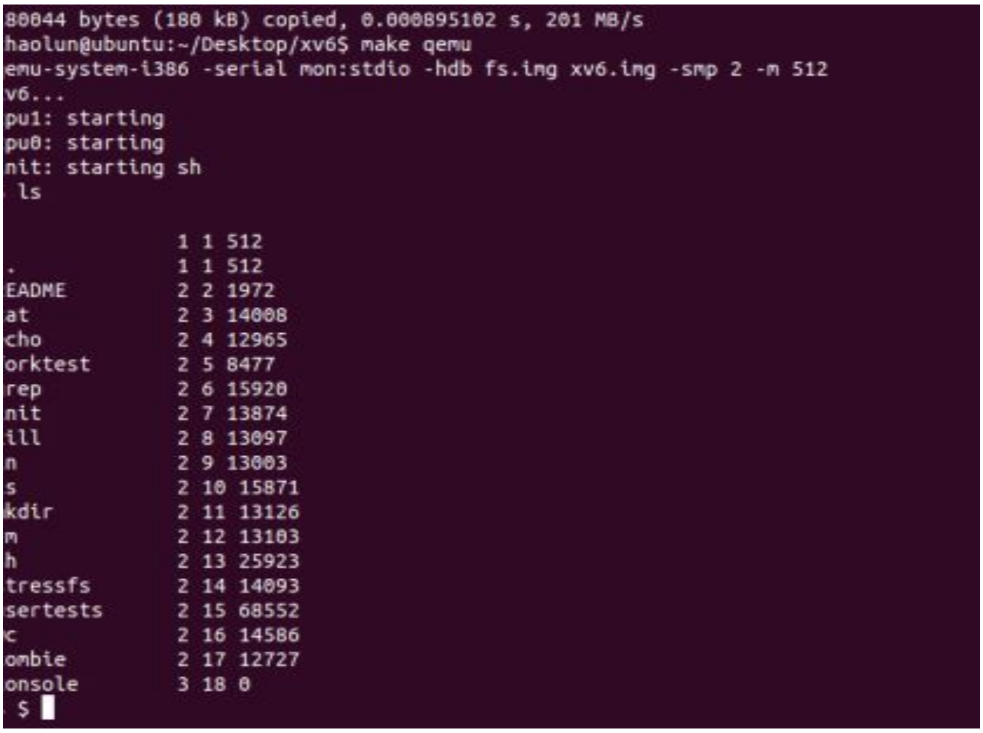
```
king-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-point
 -fno-stack-protector   -c -o wc.o wc.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _wc wc.o ulib.o usys.o printf.o umall
.o
objdump -S _wc > wc.asm
objdump -t _wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > wc.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -fvar-tracking -fvar-tr
king-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-point
 -fno-stack-protector   -c -o zombie.o zombie.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf
 umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm
sh _stressfs _usertests _wc _zombie
used 29 (bit 1 ninode 26) free 29 log 30 total 1024
balloc: first 582 blocks have been allocated
balloc: write bitmap block at sector 28
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB) copied, 0.0286341 s, 179 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.00072551 s, 706 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
351+1 records in
351+1 records out
180044 bytes (180 kB) copied, 0.000895102 s, 201 MB/s
```

Successfully compiled the xv6 by typing make in linux terminal



```
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -fvar-tracking -fvar-trac
king-assignments -O0 -g -Wall -MD -gdwarf-2 -m32 -Werror -fno-omit-frame-pointer
 -fno-stack-protector   -c -o zombie.o zombie.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf.o
 umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _
sh _stressfs _usertests _wc _zombie
used 29 (bit 1 ninode 26) free 29 log 30 total 1024
balloc: first 582 blocks have been allocated
balloc: write bitmap block at sector 28
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB) copied, 0.0286341 s, 179 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.00072551 s, 706 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
351+1 records in
351+1 records out
180044 bytes (180 kB) copied, 0.000895102 s, 201 MB/s
chaolun@ubuntu:~/Desktop/xv6$ make qemu
qemu-system-i386 -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$
```

Successfully run the xv6 on QEMU



```
80044 bytes (180 kB) copied, 0.000895102 s, 201 MB/s
haolun@ubuntu:~/Desktop/xv6$ make qemu
emu-system-i386 -serial mon:stdio -hdb fs.img xv6.img -smp 2 -m 512
v6...
pu1: starting
pu0: starting
nit: starting sh
 ls

              1 1 512
.             1 1 512
EADME         2 2 1972
at            2 3 14008
cho           2 4 12965
orktest       2 5 8477
rep           2 6 15920
nit           2 7 13874
ill           2 8 13097
n             2 9 13003
s             2 10 15871
kdir          2 11 13126
m             2 12 13103
h             2 13 25923
tressfs       2 14 14093
sertests      2 15 68552
c             2 16 14586
ombie         2 17 12727
onsole        3 18 0
 $ █
```

Using ls command to see if the xv6 is working well on QEMU

Part 2

Analyzing read(10, buf, n)

In user space:

1) read function is declared in user.h, line 10 :

```
int read(int, void*, int);
```

2) read function is defined in usys.S ,line 4 - line 9 :

```
SYSCALL(read), which was defined as:
.globl read;                    //declares read as a global symbol
read:                           //which is the entry point of read
movl $SYS_read, %eax;           //store system call number in eax register
int $T SYSCALL;                 //trigger a software interrupt and enter the kernel
ret                             //return result to the caller of read
```

3) $SYS_read was defined in syscall.h, line 6

```
#define SYS_read    5         //system call number of read is set as 5
```

4) $T_SYSCALL was defined in traps.h, line 27

```
#define $T_SYSCALL   64   // 64 is also the No. for the interrupt handler vector
```

Enter the Kernal:

int $T_SYSCALL   triggers a software interrupt:

1) CPU saves the current state, and calls the interrupt handler vector64.

2) Vector64 is in vectors.S, line317-321:

```
  .globl vector64
  vector64:
      pushl $0
      pushl $64
      jmp alltraps                    //jump to alltraps function
```

3) vector64 jump to alltraps function in vectors.S, line 321.

4) alltraps creates the trapframe and calls trap(struct trapframe *tf) function, which is in file trap.c line 37-47

5) struct trapframe (tf) saves the user-space registers, tf->eax contains the system call number which is SYS_read.

```
         proc->tf = tf;          //line42 in file trap.c, load the value from registers
```

Syscall dispatch:

1)trap (tf), which is in trap.c, calls syscall (void) : //happens in line43 in file trap.c

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL)          //will return true in this case
  {
    if(proc->killed)                   //check if the process have been killed.
      exit();                          //if true exit
    proc->tf = tf;                   //load value form tf passing into trap function
    syscall();                         //call syscall function
    if(proc->killed)                   //check if the process have been killed.
      exit();                          //if true exit
    return;
  }
}
```

2)syscall, which is in syscall.c line126-139, reads the syscall number in eax, and calls sys_read:

```
void
syscall(void)
{
  int num;
  num = proc->tf->eax;                                   //read syscall number in eax
  if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
    proc->tf->eax = syscalls[num]();                 //return is saved in tf->eax

  } else {
    cprintf("%d %s: unknown sys call %d\n",
            proc->pid, proc->name, num);
    proc->tf->eax = -1;                              //unknown sys_call number return -1
  }
}
```

3)sys_read is defined to be syscalls[SYS_read], in file syscall.c line 107:

```
static int (*syscalls[])(void) = {
      .....
   [SYS_read]     sys_read,          // //syscalls[read] is defined to be sys_read
     ......
}
```

4)sys_read, which is in sysfile.c line66-76, reads the parameter from user stack with argfd, argint, and argptr. Since fd=10 is invalid, argfd will return -1, so the return value of sys_read is -1.

```
int
sys_read(void)
{
  struct file *f;
  int n;
  char *p;

  if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
    return -1;                         //In this case,argfd(0, 0, &f) < 0 ==True, return -1
  return fileread(f, p, n);          //if sys_read() is valid, read form file
}
```

5)return value is saved in tf->eax(in syscall.c line132), control was then turned back to trap:

```
   proc->tf->eax = syscalls[num]();    //syscalls[read]==-1, saved to       proc->tf->eax
```

6)trap returns to alltraps, restores user registers and returns to user space with iret.