In His Name

Operating System Lab
First Project

Dorin Mosalman Yazdi
Marzieh Bagheri Nia
Kiavash Jamshidi

Fall 99

LAB-1A

First question

xv6 is a modern implementation of Sixth Edition Unix in ANSI C for multiprocessor x86 and RISC-V systems. It is used for pedagogical purposes in MIT's Operating Systems Engineering course as well as Georgia Tech's (CS 3210) Design of Operating Systems Course. Final architecture of the program is determined by the characteristics of a file named gdbinit.tmpl. If the file is 16 bits then i8086 is the architecture and 32 bits case results in i386 architecture.

Second question

An xv6 process consists of user-space memory (instructions, data, and stack) and per-process state private to the kernel. Xv6 can time-share processes; it transparently switches the available CPUs among the set of processes waiting to execute. When a process is not executing, xv6 saves its CPU registers, restoring them when it next runs the process. Each process can be uniquely identified by a positive integer called its process identifier, or pid.

Third question

File descriptor is a small integer representing a kernel-managed object that a process may read from or write to. it is a number that is uniquely assigned to a file that is open in the OS and by which each file is distinguished from each other. A pipe is a small kernel buffer exposed to processes as a pair of file descriptors, one for reading and one for writing. Writing data to one end of the pipe makes that data available for reading from the other end of the pipe. Pipes provide a way for processes to communicate.

Fourth question

FSSIZE is size of file system in blocks (1000). It's been defined in ¨param.h¨ and it's been used in ¨ide.c¨ and ¨mkfs.c¨.

NPROC is maximum number of processes (64). It's been defined again in ¨param.h¨ and it's been used in ¨proc.c¨.

In my opinion, NPROC is more efficient than FSSIZE because the pace of OS depends on it.

Fifth question

Xv6 is naive about disk failures: if a disk operation fails, xv6 panics. Whether this is reasonable depends on the hardware: if an operating system sits atop special hardware that uses redundancy to mask disk failures, perhaps the operating system sees failures so infrequently that panicking is okay. On the other hand, operating systems using plain disks should expect failures and handle them more gracefully, so that the loss of a block in one file doesn't affect the use of the rest of the files system.

If all the buffers are busy, something has gone wrong: get panics. A more graceful response might be to sleep until a buffer became free, though there would then be a possibility of deadlock.

LAB-1B

First question

First duty: Manages the resources.
Second duty: Create user-interface for users use.
Third one: Manage users to use resources properly.

Second question

Basic headers

Lays out a lot of constants and conventions for XV6. These constants include definitions, implementation of some structures and functions and some other stuff in order to use C language in xv6. These constants will be used frequently in other parts.

Entering XV6

Including some machine level code being run after boot lock. Xv6 boot loader executes kernel by entry after reading it from disk. Entry is defined in ¨elf.h¨ header. A page table is created by entry in order to run kernel which maps physical address to virtual. Page table is defined in the Main.c file.

Processes

Monitoring the handling of a process by CPU is done by these files. Each process has two main parts: user stack and kernel (4KB page table being managed by virtual address). Depending on the process and switches these two stacks can contain different data. At the time of interrupt or system call mode is changed to kernel and back to user when execution is finished.

Locks

Lock is itself a synchronous mechanism that limits threads in accessing resources. xv6 must introduce a coordination mechanism to keep them from interfering with each otherXv6 uses the same low-level concept for both: locks. Locks provide mutual exclusion, ensuring that only one CPU at a time can hold a lock. If xv6 only accesses a data structure while holding a particular lock, then xv6 can be sure that only one CPU at a time is accessing the data structure. In this case, we say that the lock protects the data structure.

Low-level Hardware

Initializing application, sending messages, managing IO's, accessing hardware devices, etc. These tasks are carried out by kernel interface which has the series of low level functions.

User-level

Shell commands that allow users to intervene with operations on file systems. These commands include close, fork, kill and etc.

String operations

Functions to enable working with strings and collaborating with memory by strings.

Pipes

Pipe is a connection tunnel between file descriptors. You may see it as a buffer inside kernel containing two layers, one for reading (0) and other for writing (1). Writing on one side of pipe allows reading from the other side.

File System
File System handles two data types: One is array of bytes and Second is a tree of directories. File System allows making directories, linking them and determining how to interact with a directory (r, w, rw, create).

System Calls
This part is the manager of system calls. A process will change mode to kernel by system call. Inspections are made after the system call type is determined using a vector table. System call replaces process memory with a memory image but keeps the last memory.

Boot loader
Boot loader is loaded from the first sector of disk onto memory as the Device is turned on. Its main task is to put xv6 Kernel on memory.

Link
Link is a system call that appends a new name to file. A file can have multiple names being known as link.

Fourth question
UPROGS
For adding a user program to xv6, we use this phrase. For example, for adding a program named cat.c, we add the following code in makefile:

```
¨ UPROGS=\
  _cat\¨
```

ULIB
ULib is a developer library that provides some features as universal physics, user access lists, and much more! In makefile, for adding and using objectfiles, we use this phrase and it combines all the objectfiles we choose together and holds them in a library named ULIB that can be pointes in other commands in makefile!

Eighth question
Objcopy command copies the contents of an object file to another. It reads the object files and can write it in another format. In this case, it generates a way binary file which will produce a memory dump on the contents. As mentioned before, compiled files are stored as binary files. So Objcopy is used for make.

Ninth question
Booting is the first stage to run an operating system so not all the extensions of the C language are loaded. This means that we will have some limitations if we only use C. That's why it is essential to use assembly code for booting. However, after booting and when all the extensions of C are loaded it is more convenient to only use C.

Tenth question

- **General Purpose Registers:** Registers used to perform most of the instruction (ex. EAX, EBX, …).
- **Segment Registers:** Hold the segment address of various items. They can be set by a general register or special instructions (ex. CS, DS, ...).
- **Status Registers:** Contains the current state of the processor (EFLAGS, RFLAGS, ...).
- **Control Registers:** Used in the protected mode programming (ex. CR0 to CR4).

Fourteenth question

The equivalent code to ¨entry.s¨ is ¨head_64.s¨ that can be found at:
¨https://github.com/torvalds/linux/blob/master/arch/x86/kernel/head_64.S¨

Fifteenth question

Each virtual address must have a physical location to store its table, so instead of having a virtual table plus a physical reference, we store it as a physical address to use less space and make it easier to use.

Sixteenth question

The equivalent function in Linux operating system is ¨start_kernel¨.
The functions which they are called in main, are in the following order:

- **kinit1(end, P2V(4*1024*1024)):** This function used to free up the physical pages of memory.
- **kvmalloc():** This function used to create a new page and returns its URL.
- **mpinit()**: This function detects other processors.
- **lapicinit():** This function controls interrupts.
- **seginit():** This function organizes the segmentation table.
- **picinit():** This function disables programmable interrupt table.
- **ioapicinit():** This function controls I/O interrupts.
- **consoleinit():** This function manages the console in terms of hardware.
- **uartinit():** This function manages Intel hardware serial port.
- **pinit():** This function manages process.
- **tvinit():** This function makes a vector of traps.
- **binit():**This function frees up memory physical pages.
- **fileinit():** This function creates the file management table.
- **ideinit():** This function is used to quantify and identify disks.
- **startothers():** This function starts the rest of the processors task.
- **kinit2(P2V(4*1024*1024), P2V(PHYSTOP)):** This function is used to prepare pages for other processors.
- **userinit():** this function creates the first process.
- **mpmain():** This function goes to the end of the first process.

Nineteenth question

The equivalent function in Linux operating system is ¨struct task_struct¨.
Its various aspects are:

- **uint sz:** It expresses the required memory for processes in bytes unit.
- **pde_t* pgdir:** It stores the table address of the running user level application.
- **char *kstack:** It is a pointer to the bottom of the kernel stack for this process.
- **enum procstate state:** It expresses the status of the process at any moment.
- **int pid:** It holds this number as an ID for each process.
- **struct proc *parent:** It is a pointer to the process that is the father of this process.
- **struct trapframe *tf:** It holds the frame for system access.
- **struct context *context:** This structure holds the context when we need to do context switching.
- **void *chan:** If this flag is one (active), it indicates that the process is sleep.
- **int killed:** If this flag is one (active), it indicates that the process is dead.
- **struct file *ofile[NOFILE]:** This structure holds files that have been opened for the process.
- **struct inode *cwd:** It Maintains the current directory.
- **char name[16]:** It Retains the process name and is also used in debugging cases.

LAB-1C

First question

¨Bt¨ command prints the list of all the functions called. This list is a stack starting with the last function till the first function called. We can also print only a specific number of functions by giving this command a number as an input argument.

```
(gdb) bt
#0  sys_read () at sysfile.c:71
#1  0x80104a69 in syscall () at syscall.c:139
#2  0x80105a25 in trap (tf=0x8dffefb4) at trap.c:43
#3  0x8010583f in alltraps () at trapasm.S:20
#4  0x8dffefb4 in ?? ()
```

Second question

An infinite loop can be implemented with a while statement in the sys_read() function. This loop creates an interrupt in this function which can be seen in both user and kernel mode.

```
(gdb) c
Continuing.
[Switching to Thread 2]

Thread 2 hit Breakpoint 1, sys_read () at sysfile.c:71
71      {
(gdb)
```

As seen in the figure above, kernel mode exactly returns the function and the line which interrupt has happened. While in the user mode (figure below) it is unknown which function was being run before receiving the interrupt.

```
(gdb) c
Continuing.
^C
Thread 1 received signal SIGINT, Interrupt.
0x80103951 in ?? ()
(gdb)
```

Third question

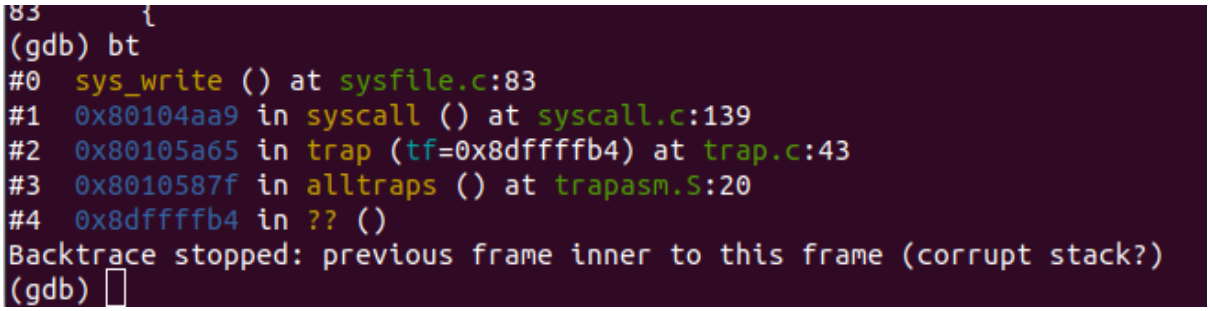
While Print shows the value stored in a named variable, x command shows the contents of a memory address. ¨info registers **regname**¨ can be used to print the value of the specific register **regname**.

Fourth question

The equivalent command for **n** is **ni (nexti)** and the equivalent command for **s** is **si (stepi)**.

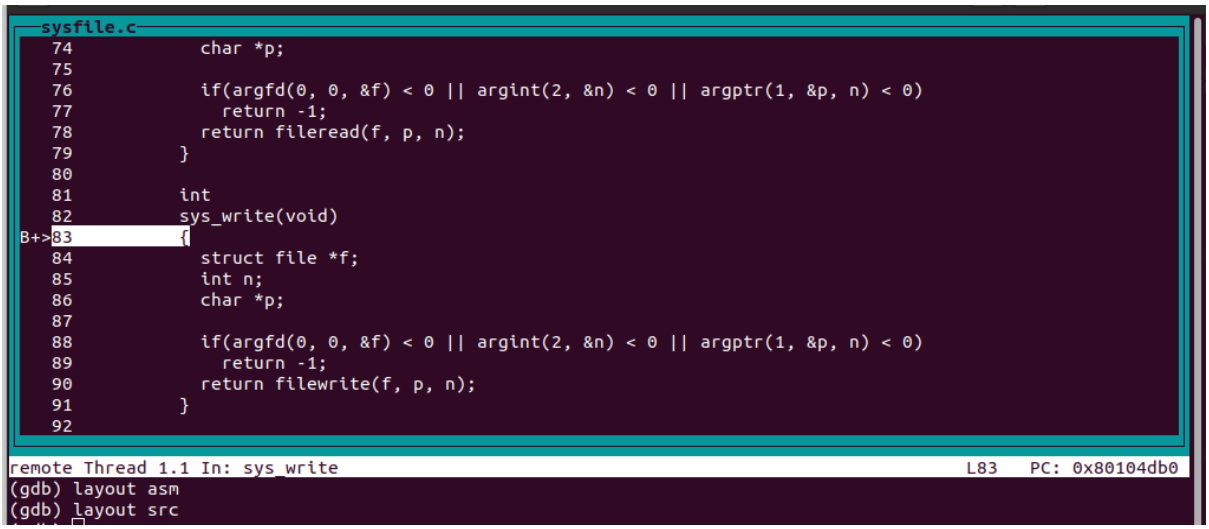
Fifth question

The ¨Backtrace (bt)¨ command is a summary of how the program got to this point. Each line of this command displays the frame number, PC value, source file name, line number in the main code, and function input arguments. An example image of this command is below:

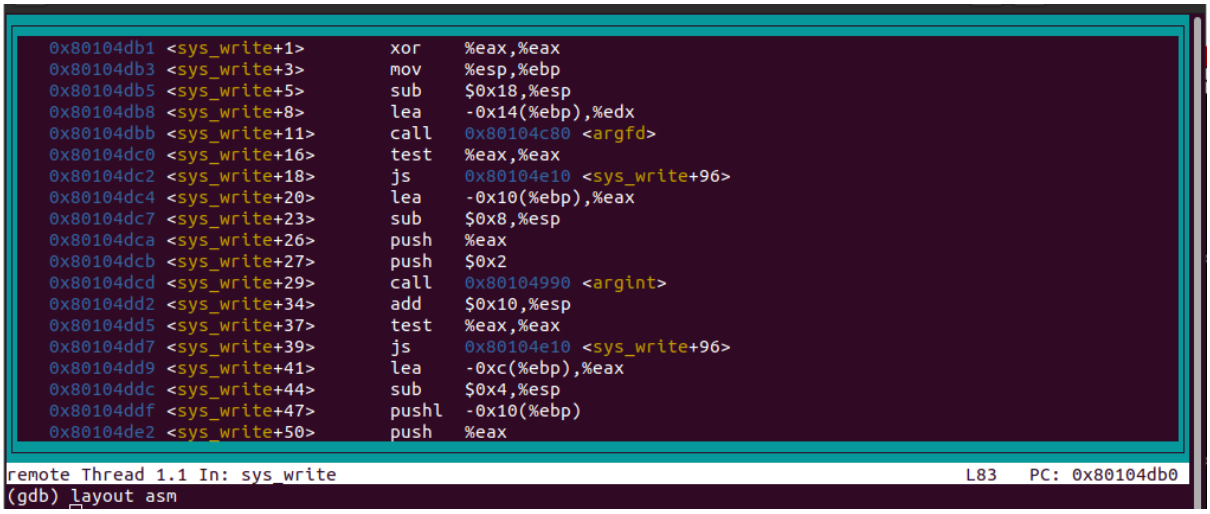


Sixth question

The ¨layout src¨ command shows every lines of the program that is running. An example image of this command is below:



The ¨layout asm¨ command does the same works but in assembly language. An example image of this command is below:



For example, **int** means some interrupt is happening and **ret** means return!