Started on	Saturday, 20 February 2021, 2:48:51 PM
State	Finished
Completed on	Saturday, 20 February 2021, 3:53:52 PM
Time taken	1 hour 5 mins
Grade	**14.64** out of 20.00 (**73**%)

Question **1**

Partially correct

Mark 0.57 out of 1.00

Select all the correct statements about code of bootmain() in xv6

```
void
bootmain(void)
{
  struct elfhdr *elf;
  struct proghdr *ph, *eph;
  void (*entry)(void);
  uchar* pa;

  elf = (struct elfhdr*)0x10000;  // scratch space

  // Read 1st page off disk
  readseg((uchar*)elf, 4096, 0);

  // Is this an ELF executable?
  if(elf->magic != ELF_MAGIC)
    return;  // let bootasm.S handle error

  // Load each program segment (ignores ph flags).
  ph = (struct proghdr*)((uchar*)elf + elf->phoff);
  eph = ph + elf->phnum;
  for(; ph < eph; ph++){
    pa = (uchar*)ph->paddr;
    readseg(pa, ph->filesz, ph->off);
    if(ph->memsz > ph->filesz)
      stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
  }

  // Call the entry point from the ELF header.
  // Does not return!
  entry = (void(*)(void))(elf->entry);
  entry();
}
```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

- [ ] a. The kernel file has only two program headers
- [ ] b. The condition if(ph->memsz > ph->filesz) is never true.
- [ ] c. The kernel file gets loaded at the Physical address 0x10000 +0x80000000 in memory.
- [x] d. The elf->entry is set by the linker in the kernel file and it's 8010000c ✔
- [ ] e. The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded
- [ ] f. The elf->entry is set by the linker in the kernel file and it's 0x80000000
- [ ] g. The readseg finally invokes the disk I/O code using assembly instructions
- [x] h. The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it. ✔
- [x] i. The kernel file gets loaded at the Physical address 0x10000 in memory. ✔
- [ ] j. The elf->entry is set by the linker in the kernel file and it's 0x80000000
- [x] k. The stosb() is used here, to fill in some space in memory with zeroes ✔

Your answer is partially correct.

You have correctly selected 4.
The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question **2**

Partially correct

Mark 0.25 out of 0.50

xv6.img: bootblock kernel

```
dd if=/dev/zero of=xv6.img count=10000
dd if=bootblock of=xv6.img conv=notrunc
dd if=kernel of=xv6.img seek=1 conv=notrunc
```

Consider above lines from the Makefile. Which of the following is incorrect?

- [ ] a. The bootblock may be 512 bytes or less (looking at the Makefile instruction)
- [x] b. The size of the kernel file is nearly 5 MB ✔
- [ ] c. The xv6.img is the virtual disk that is created by combining the bootblock and the kernel file.
- [ ] d. The size of xv6.img is exactly = (size of bootblock) + (size of kernel)
- [ ] e. The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies 10,000 blocks on the disk.
- [ ] f. The bootblock is located on block-0 of the xv6.img
- [ ] g. The kernel is located at block-1 of the xv6.img
- [ ] h. The size of the xv6.img is nearly 5 MB
- [ ] i. Blocks in xv6.img after kernel may be all zeroes.
- [x] j. The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk. ✔
- [ ] k. xv6.img is the virtual processor used by the qemu emulator

Your answer is partially correct.

You have correctly selected 2.
The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

Question **3**

Correct

Mark 0.50 out of 0.50

Order the events that occur on a timer interrupt:

Select another process for execution	5	✔
Jump to a code pointed by IDT	2	✔
Set the context of the new process	6	✔
Jump to scheduler code	3	✖
Execute the code of the new process	7	✔
Change to kernel stack	1	✔
Save the context of the currently running process	2	✖

Your answer is correct.

The correct answer is: Select another process for execution → 5, Jump to a code pointed by IDT → 2, Set the context of the new process → 6, Jump to scheduler code → 4, Execute the code of the new process → 7, Change to kernel stack → 1, Save the context of the currently running process → 3

Comment:

Question 4

Incorrect

Mark 0.00 out of 1.00

Consider the two programs given below to implement the command (ignore the fact that error checks are not done on return values of functions)

$ ls . /tmp/asdfksdf >/tmp/ddd 2>&1

Program 1

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(1);
    dup(fd);
    close(2);
    dup(fd);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Program 2

```
int main(int argc, char *argv[]) {
    int fd, n, i;
    char buf[128];

    close(1);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    close(2);
    fd = open("/tmp/ddd", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    execl("/bin/ls", "/bin/ls", ".", "/tmp/asldjfaldfs", NULL);
}
```

Select all the correct statements about the programs

Select one or more:

- [ ] a. Program 1 is correct for > /tmp/ddd but not for 2>&1
- [x] b. Both programs are correct ✖
- [ ] c. Program 1 does 1>&2
- [ ] d. Program 2 is correct for > /tmp/ddd but not for 2>&1
- [ ] e. Program 2 makes sure that there is one file offset used for '2' and '1'
- [ ] f. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd
- [ ] g. Program 2 does 1>&2
- [ ] h. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd
- [ ] i. Only Program 1 is correct
- [ ] j. Program 1 makes sure that there is one file offset used for '2' and '1'
- [ ] k. Both program 1 and 2 are incorrect
- [ ] l. Only Program 2 is correct

Your answer is incorrect.

The correct answers are: Only Program 1 is correct, Program 1 makes sure that there is one file offset used for '2' and '1'

Question **5**

Correct

Mark 0.50 out of 0.50

Select Yes/True if the mentioned element must be a part of PCB

Select No/False otherwise.

Yes	No		
◉ ✔	○ ✖	Pointer to the parent process	✔
◉ ✔	○ ✖	Process state	✔
◉ ✔	○ ✖	Process context	✔
○ ✖	◉ ✔	Pointer to IDT	✔
◉ ✔	○ ✖	List of opened files	✔
◉ ✔	○ ✖	PID	✔
○ ✖	◉ ✔	Function pointers to all system calls	✔
◉ ✔	○ ✖	Memory management information about that process	✔
◉ ✔	○ ✖	EIP at the time of context switch	✔
○ ✖	◉ ✔	Parent's PID	✔

Pointer to the parent process: Yes
Process state: Yes
Process context: Yes
Pointer to IDT: No
List of opened files: Yes
PID: Yes
Function pointers to all system calls: No
Memory management information about that process: Yes
EIP at the time of context switch: Yes
Parent's PID: No

Question **6**

Partially correct

Mark 0.29 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
int main(int argc, char *argv[]) {
   int fd1, fd2 = 1, fd3 = 1, fd4 = 1;

   fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
   fd2 = open("/tmp/2", O_RDDONLY);
   fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
   close(0);
   close(1);
   dup(fd2);
   dup(fd3);
   close(fd3);
   dup2(fd2, fd4);
   printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);
   return 0;
}
```

fd4	stdout	✖
0	stdin	✖
1	stdout	✖
fd3	stderr	✖
fd2	stdout	✖
2	stderr	✔
fd1	/tmp/1	✔

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: fd4 → /tmp/2, 0 → /tmp/2, 1 → /tmp/3, fd3 → closed, fd2 → /tmp/2, 2 → stderr, fd1 → /tmp/1

Question **7**

Correct

Mark 0.25 out of 0.25

Order the following events in boot process (from 1 onwards)

Boot loader	2 ✔
BIOS	1 ✔
Login interface	5 ✔
Shell	6 ✔
OS	3 ✔
Init	4 ✔

Your answer is correct.

The correct answer is: Boot loader → 2, BIOS → 1, Login interface → 5, Shell → 6, OS → 3, Init → 4

Question **8**

Correct

Mark 0.25 out of 0.25

Select the option which best describes what the CPU does during it's powered ON lifetime

- a. Ask the OS what is to be done, and execute that task
- b. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask the User or the OS what is to be done next, repeat
- c. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask OS what is to be done next, repeat
- d. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, repeat ✔
- e. Ask the user what is to be done, and execute that task
- f. Fetch instruction specified by OS, Decode and execute it, repeat

The correct answer is: Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, repeat

Question **9**

Correct

Mark 0.25 out of 0.25

Match a system call with it's description

pipe	create an unnamed FIFO storage with 2 ends - one for reading and another for writing ✔
dup	create a copy of the specified file descriptor into smallest available file descriptor ✔
dup2	create a copy of the specified file descriptor into another specified file descriptor ✔
exec	execute a binary file overlaying the image of current process ✔
fork	create an identical child process ✔

Your answer is correct.

The correct answer is: pipe → create an unnamed FIFO storage with 2 ends - one for reading and another for writing, dup → create a copy of the specified file descriptor into smallest available file descriptor, dup2 → create a copy of the specified file descriptor into another specified file descriptor, exec → execute a binary file overlaying the image of current process, fork → create an identical child process

Question **10**

Incorrect

Mark 0.00 out of 0.25

In bootasm.S, on the line

```
ljmp    $(SEG_KCODE<<3), $start32
```

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

- a. The code segment is 16 bit and only upper 13 bits are used for segment number
- b. While indexing the GDT using CS, the value in CS is always divided by 8
- c. The ljmp instruction does a divide by 8 on the first argument
- d. The code segment is 16 bit and only lower 13 bits are used for segment number
- e. The value 8 is stored in code segment ✖

Your answer is incorrect.

The correct answer is: The code segment is 16 bit and only upper 13 bits are used for segment number

Question **11**

Correct

Mark 1.00 out of 1.00

For each line of code mentioned on the left side, select the location of sp/esp that is in use

jmp *%eax in entry.S	The 4KB area in kernel image, loaded in memory, named as 'stack'	✔
readseg((uchar*)elf, 4096, 0); in bootmain.c	0x7c00 to 0	✔
ljmp $(SEG_KCODE<<3), $start32 in bootasm.S	Immaterial as the stack is not used here	✔
call bootmain in bootasm.S	0x7c00 to 0	✔
cli in bootasm.S	Immaterial as the stack is not used here	✔

Your answer is correct.

The correct answer is: jmp *%eax
in entry.S → The 4KB area in kernel image, loaded in memory, named as 'stack', readseg((uchar*)elf, 4096, 0);
in bootmain.c → 0x7c00 to 0, ljmp $(SEG_KCODE<<3), $start32
in bootasm.S → Immaterial as the stack is not used here, call bootmain
in bootasm.S → 0x7c00 to 0, cli
in bootasm.S → Immaterial as the stack is not used here

Question **12**

Correct

Mark 0.25 out of 0.25

Match the following parts of a C program to the layout of the process in memory

Instructions	Text section	✔
Dynamically allocated memory	Heap Section	✔
Global and static data	Data section	✔
Local Variables	Stack Section	✔

Your answer is correct.

The correct answer is:
Instructions → Text section,
Dynamically allocated memory → Heap Section,
Global and static data → Data section, Local Variables → Stack Section

Question **13**

Partially correct

Mark 0.10 out of 0.25

Select the order in which the various stages of a compiler execute.

Intermediate code generation	does not exist	✖
Syntatical Analysis	2	✔
Linking	3	✖
Pre-processing	1	✔
Loading	4	✖

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Intermediate code generation → 3, Syntatical Analysis → 2, Linking → 4, Pre-processing → 1, Loading → does not exist

Question **14**

Correct

Mark 0.25 out of 0.25

Which of the following are the files related to bootloader in xv6?

- a. bootasm.S, bootmain.c and bootblock.c
- b. bootasm.S and bootmain.c ✔
- c. bootmain.c and bootblock.S
- d. bootasm.s and entry.S

Your answer is correct.

The correct answer is: bootasm.S and bootmain.c

Question **15**

Incorrect

Mark 0.00 out of 1.00

Select the sequence of events that are NOT possible, assuming a non-interruptible kernel code

Select one or more:

- [ ] a. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return

- [ ] b.
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

- [ ] c. P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

- [ ] d. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running

- [ ] e. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes sytem call
system call returns
P1 running
timer interrupt
scheduler
P2 running

- [x] f. P1 running ✖
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns

P3 running
Timer interrupt
Scheduler
P1 running

Your answer is incorrect.

The correct answers are: P1 running
P1 makes sytem call and blocks
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again, P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return,
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

Question **16**

Partially correct

Mark 0.89 out of 1.00

Select all the correct statements about the state of a process.

- [ ] a. A waiting process starts running after the wait is over
- [x] b. A running process may terminate, or go to wait or become ready again ✔
- [x] c. Changing from running state to waiting state results in "giving up the CPU" ✔
- [x] d. Typically, it's represented as a number in the PCB ✔
- [x] e. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✔
- [x] f. Processes in the ready queue are in the ready state ✔
- [ ] g. A process in ready state is ready to receive interrupts
- [ ] h. A process waiting for any condition is woken up by another process only
- [x] i. A process changes from running to ready state on a timer interrupt ✔
- [ ] j. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers
- [x] k. A process can self-terminate only when it's running ✔
- [ ] l. A process that is running is not on the ready queue
- [x] m. A process in ready state is ready to be scheduled ✔
- [ ] n. A process changes from running to ready state on a timer interrupt or any I/O wait

Your answer is partially correct.

You have correctly selected 8.
The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question **17**

Partially correct

Mark 0.17 out of 0.50

Which of the following is/are not saved during context switch?

- [ ] a. Cache
- [ ] b. General Purpose Registers
- [ ] c. Stack Pointer
- [ ] d. Program Counter
- [x] e. Bus ✔
- [ ] f. MMU related registers/information
- [ ] g. TLB

Your answer is partially correct.

You have correctly selected 1.
The correct answers are: TLB, Cache, Bus

Question **18**

Partially correct

Mark 0.40 out of 0.50

Select all the correct statements about linking and loading.

Select one or more:

- [ ] a. Dynamic linking is possible with continous memory management, but variable sized partitions only.
- [x] b. Dynamic linking essentially results in relocatable code. ✔
- [ ] c. Dynamic linking and loading is not possible without demand paging or demand segmentation.
- [x] d. Loader is part of the operating system ✔
- [x] e. Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently) ✔
- [ ] f. Static linking leads to non-relocatable code
- [ ] g. Continuous memory management schemes can support dynamic linking and dynamic loading.
- [ ] h. Loader is last stage of the linker program
- [x] i. Continuous memory management schemes can support static linking and static loading. (may be inefficiently) ✔

Your answer is partially correct.

You have correctly selected 4.
The correct answers are: Continuous memory management schemes can support static linking and static loading. (may be inefficiently), Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently), Dynamic linking essentially results in relocatable code., Loader is part of the operating system, Dynamic linking and loading is not possible without demand paging or demand segmentation.

Question **19**

Correct

Mark 0.50 out of 0.50

Consider the following command and it's output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
  readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

```
xv6.img: bootblock kernel
    dd if=/dev/zero of=xv6.img count=10000
    dd if=bootblock of=xv6.img conv=notrunc
    dd if=kernel of=xv6.img seek=1 conv=notrunc

kernel: $(OBJS) entry.o entryother initcode kernel.ld
    $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
    $(OBJDUMP) -S kernel > kernel.asm
    $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- [x] a. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✔
- [ ] b. The kernel.asm file is the final kernel file
- [x] c. Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes. ✔
- [ ] d. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.
- [ ] e. Althought the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- [x] f. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✔
- [ ] g. The bootmain() code does not read the kernel completely in memory
- [x] h. readseg() reads first 4k bytes of kernel in memory ✔
- [x] i. The kernel.ld file contains instructions to the linker to link the kernel properly ✔

Your answer is correct.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain()., readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question **20**

Correct

Mark 1.00 out of 1.00

```
int f() {
    int count;
    for (count = 0; count< 4; count ++) {
        if (fork() ==0)
            printf("Operating-System\n");
    }
    printf("TYCOMP\n");
}
```

The number of times "Operating-System" is printed, is:

Answer: 15 ✔

The correct answer is: 15.00

Question **21**

Partially correct

Mark 0.37 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- [x] a. local variable declaration ✖
- [ ] b. expressions
- [ ] c. function calls
- [x] d. #directives ✔
- [x] e. typedefs ✔
- [x] f. global variables ✔
- [ ] g. pointer dereference

Your answer is partially correct.

You have selected too many options.

The correct answers are: #directives, typedefs, global variables

Question **22**

Correct

Mark 0.25 out of 0.25

Which of the following are NOT a part of job of a typical compiler?

- [x] a. Suggest alternative pieces of code that can be written ✔
- [ ] b. Convert high level langauge code to machine code
- [x] c. Check the program for logical errors ✔
- [ ] d. Invoke the linker to link the function calls with their code, extern globals with their declaration
- [ ] e. Process the # directives in a C program
- [ ] f. Check the program for syntactical errors

Your answer is correct.

The correct answers are: Check the program for logical errors, Suggest alternative pieces of code that can be written

Question **23**

Partially correct

Mark 0.86 out of 1.00

Select all the correct statements about calling convention on x86 32-bit.

- [x] a. Parameters may be passed in registers or on stack ✔
- [x] b. Compiler may allocate more memory on stack than needed ✔
- [ ] c. The return value is either stored on the stack or returned in the eax register
- [x] d. Return address is one location above the ebp ✔
- [ ] e. during execution of a function, ebp is pointing to the old ebp
- [x] f. The ebp pointers saved on the stack constitute a chain of activation records ✔
- [ ] g. Paramters are pushed on the stack in left-right order
- [ ] h. The two lines in the beginning of each function, "push %ebp; mov %esp, %ebp", create space for local variables
- [ ] i. Space for local variables is allocated by substracting the stack pointer inside the code of the caller function
- [x] j. Space for local variables is allocated by substracting the stack pointer inside the code of the called function ✔
- [x] k. Parameters may be passed in registers or on stack ✔

Your answer is partially correct.

You have correctly selected 6.
The correct answers are: Compiler may allocate more memory on stack than needed, Parameters may be passed in registers or on stack, Parameters may be passed in registers or on stack, Return address is one location above the ebp, during execution of a function, ebp is pointing to the old ebp, Space for local variables is allocated by substracting the stack pointer inside the code of the called function, The ebp pointers saved on the stack constitute a chain of activation records

Question **24**
Correct
Mark 1.00 out of 1.00

Match the program with it's output (ignore newlines in the output. Just focus on the count of the number of 'hi')

main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }	hi hi ✔
main() { int i = fork(); if(i == 0) execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }	hi ✔
main() { int i = NULL; fork(); printf("hi\n"); }	hi hi ✔
main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }	hi ✔

Your answer is correct.

The correct answer is: main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); } → hi hi, main() { int i = fork(); if(i == 0) execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); } → hi, main() { int i = NULL; fork(); printf("hi\n"); } → hi hi, main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); } → hi

Question **25**
Correct
Mark 0.25 out of 0.25

Rank the following storage systems from slowest (first) to fastest(last)

RAM	5 ✔
Cache	6 ✔
Registers	7 ✔
Hard Disk	3 ✔
Magnetic Tapes	1 ✔
Optical Disks	2 ✔
Non volatile memory	4 ✔

Your answer is correct.

The correct answer is: RAM → 5, Cache → 6, Registers → 7, Hard Disk → 3, Magnetic Tapes → 1, Optical Disks → 2, Non volatile memory → 4

Question **26**

Correct

Mark 0.50 out of 0.50

What will this program do?

```
int main() {
fork();
execl("/bin/ls", "/bin/ls", NULL);
printf("hello");
}
```

- a. run ls once
- b. one process will run ls, another will print hello
- c. run ls twice and print hello twice
- d. run ls twice and print hello twice, but output will appear in some random order
- e. run ls twice ✔

Your answer is correct.

The correct answer is: run ls twice

Question **27**

Partially correct

Mark 0.20 out of 0.25

Match the register with the segment used with it.

eip	cs	✔
ebp	ss	✔
esi	ds	✔
esp	ss	✔
edi	ds	✖

Your answer is partially correct.

You have correctly selected 4.
The correct answer is: eip → cs, ebp → ss, esi → ds, esp → ss, edi → es

Question **28**

Correct

Mark 1.00 out of 1.00

Select the correct statements about interrupt handling in xv6 code

- [x] a. All the 256 entries in the IDT are filled ✔
- [x] b. The function trap() is the called irrespective of hardware interrupt/system-call/exception ✔
- [x] c. The CS and EIP are changed only after pushing user code's SS,ESP on stack ✔
- [ ] d. On any interrupt/syscall/exception the control first jumps in trapasm.S
- [x] e. The trapframe pointer in struct proc, points to a location on kernel stack ✔
- [ ] f. The CS and EIP are changed only immediately on a hardware interrupt
- [x] g. Before going to alltraps, the kernel stack contains upto 5 entries. ✔
- [ ] h. The trapframe pointer in struct proc, points to a location on user stack
- [ ] i. xv6 uses the 0x64th entry in IDT for system calls
- [x] j. xv6 uses the 64th entry in IDT for system calls ✔
- [ ] k. The function trap() is the called only in case of hardware interrupt
- [x] l. Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt ✔
- [x] m. On any interrupt/syscall/exception the control first jumps in vectors.S ✔

Your answer is correct.

The correct answers are: All the 256 entries in the IDT are filled, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on kernel stack, The function trap() is the called irrespective of hardware interrupt/system-call/exception, The CS and EIP are changed only after pushing user code's SS,ESP on stack

Question **29**

Correct

Mark 0.50 out of 0.50

Some part of the bootloader of xv6 is written in assembly while some part is written in C. Why is that so?
Select all the appropriate choices

- [x] a. The setting up of the most essential memory management infrastructure needs assembly code ✔
- [ ] b. The code in assembly is required for transition to protected mode, from real mode; but calling convention was applicable all the time
- [x] c. The code in assembly is required for transition to protected mode, from real mode; after that calling convention applies, hence code can be written in C ✔
- [ ] d. The code for reading ELF file can not be written in assembly

Your answer is correct.

The correct answers are: The code in assembly is required for transition to protected mode, from real mode; after that calling convention applies, hence code can be written in C, The setting up of the most essential memory management infrastructure needs assembly code

Question **30**

Correct

Mark 0.50 out of 0.50

Order the sequence of events, in scheduling process P1 after process P0

Control is passed to P1	5	✔
Process P0 is running	1	✔
context of P1 is loaded from P1's PCB	4	✔
timer interrupt occurs	2	✔
context of P0 is saved in P0's PCB	3	✔
Process P1 is running	6	✔

Your answer is correct.

The correct answer is: Control is passed to P1 → 5, Process P0 is running → 1, context of P1 is loaded from P1's PCB → 4, timer interrupt occurs → 2, context of P0 is saved in P0's PCB → 3, Process P1 is running → 6

Question **31**

Correct

Mark 0.25 out of 0.25

What's the trapframe in xv6?

- a. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware only
- b. A frame of memory that contains all the trap handler code
- c. The IDT table
- d. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware + code in trapasm.S ✔
- e. A frame of memory that contains all the trap handler code's function pointers
- f. A frame of memory that contains all the trap handler's addresses
- g. The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by code in trapasm.S only

Your answer is correct.

The correct answer is: The sequence of values, including saved registers, constructed on the stack when an interrupt occurs, built by hardware + code in trapasm.S

Question **32**
Partially correct
Mark 1.30 out of 2.00

Following code claims to implement the command

/bin/ls -l | /usr/bin/head -3 | /usr/bin/tail -1

Fill in the blanks to make the code work.

Note: Do not include space in writing any option. x[1][2] should be written without any space, and so is the case with [1] or [2]. Pay attention to exact syntax and do not write any extra character like ';' or = etc.

int main(int argc, char *argv[]) {
int pid1, pid2;
int pfd[

2

✔][2];

pipe(

pfd[0]

✔);
pid1 =

fork()

✔ ;
if(pid1 != 0) {
close(pfd[0]

[0]

✔);
close(

0

✖);
dup(

pfd[0][1]

✔);
execl("/bin/ls", "/bin/ls", "

-l

✔ ", NULL);
}
pipe(

pfd[1]

✔);

pid2

✔ = fork();
if(pid2 == 0) {
close(

pfd[0][1]

✖ ;
close(0);
dup(

pfd[1][1]

✖);
close(pfd[1]

[0]

✔);
close(

1

✔);
dup(

pfd[1][0]

✖);
execl("/usr/bin/head", "/usr/bin/head", "

-3

✔ ", NULL);
} else {
close(pfd

[1][1]

✔);
close(

✖);
dup(

✖);
close(pfd

✖);
execl("/usr/bin/tail", "/usr/bin/tail", "

-1

✔ ", NULL);
}
}

Question **33**

Correct

Mark 0.25 out of 0.25

What is the OS Kernel?

- a. The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run ✔ correct
- b. Everything that I see on my screen
- c. Only the system programs like compiler, linker, loader, etc.
- d. The set of tools like compiler, linker, loader, terminal, shell, etc.

The correct answer is: The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run

◄ classroom-questions-15feb21

Jump to...

Mid Term feedback (Div1) ►