Dashboard / My courses / Computer Engineering & IT / CEIT-Even-sem-21-22 / OS-even-sem-21-22 / 24 January - 30 January / Topic-wise Quiz: 1 (system calls, x86, calling convention)

Started on	Monday, 24 January 2022, 7:29:59 PM
State	Finished
Completed on	Monday, 24 January 2022, 9:09:45 PM
Time taken	1 hour 39 mins
Grade	**12.11** out of 20.00 (**61**%)

Question **1**

Complete

Mark 0.50 out of 0.50

Compare multiprogramming with multitasking

○ a. A multitasking system is not necessarily multiprogramming

● b. A multiprogramming system is not necessarily multitasking

The correct answer is: A multiprogramming system is not necessarily multitasking

Question **2**

Complete

Mark 0.00 out of 2.00

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

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**

Complete

Mark 1.00 out of 1.00

```
int value = 5;
int main()
{
    pid_t pid;
    pid = fork();
    if (pid == 0) { /* child process */
        value += 15;
        return 0;
    }
    else if (pid > 0) { /* parent process */
        wait(NULL);
        printf("%d", value); /* LINE A */
    }
    return 0;
}
```

What's the value printed here at LINE A?

Answer: 5

The correct answer is: 5

Question **4**

Complete

Mark 2.00 out of 2.00

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

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

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

Question **5**

Complete

Mark 1.67 out of 2.00

Which of the following instructions should be privileged?

Select one or more:

- ☐ a. Set value of a memory location
- ☑ b. Modify entries in device-status table
- ☑ c. Access memory management unit of the processor
- ☑ d. Set value of timer.
- ☑ e. Turn off interrupts.
- ☐ f. Read the clock.
- ☑ g. Switch from user to kernel mode.
- ☐ h. Access I/O device.
- ☐ i. Access a general purpose register

The correct answers are: Set value of timer., Access memory management unit of the processor, Turn off interrupts., Modify entries in device-status table, Access I/O device., Switch from user to kernel mode.

Question **6**

Complete

Mark 1.00 out of 1.00

How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) ?

Select one:

- ○ a. It disallows hardware interrupts when a process is running
- ○ b. It prohibits one process from accessing other process's memory
- ● c. It prohibits a user mode process from running privileged instructions
- ○ d. It prohibits invocation of kernel code completely, if a user program is running

The correct answer is: It prohibits a user mode process from running privileged instructions

Question **7**

Complete

Mark 0.80 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

- ☐ a. Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode
- ☑ b. There is an instruction like 'iret' to return from kernel mode to user mode
- ☑ c. The two modes are essential for a multitasking system
- ☑ d. The two modes are essential for a multiprogramming system
- ☑ e. The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously

The correct answers are: The two modes are essential for a multiprogramming system, The two modes are essential for a multitasking system, There is an instruction like 'iret' to return from kernel mode to user mode, The software interrupt instructions change the mode from user mode to kernel mode and jumps to predefined location simultaneously, Some instructions are allowed to run only in user mode, while all instructions can run in kernel mode

Question **8**

Complete

Mark 1.00 out of 1.00

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

You can drag and drop the items below/above each other.

- Magnetic tapes
- Optical disk
- Hard-disk drives
- Nonvolatile memory
- Main memory
- Cache
- Registers

Question **9**

Complete

Mark 0.50 out of 0.50

Is the terminal a part of the kernel on GNU/Linux systems?

- ○ a. yes
- [✓ b. no]

The correct answer is: no

Question **10**

Complete

Mark 0.17 out of 2.00

Select all statements that correctly explain the use/purpose of system calls.

Select one or more:

- [x] a. Provide services for accessing files
- [x] b. Handle exceptions like division by zero
- [ ] c. Run each instruction of an application program
- [x] d. Switch from user mode to kernel mode
- [x] e. Handle ALL types of interrupts
- [ ] f. Provide an environment for process creation
- [x] g. Allow I/O device access to user processes

The correct answers are: Switch from user mode to kernel mode, Provide services for accessing files, Allow I/O device access to user processes, Provide an environment for process creation

Question **11**

Complete

Mark 0.33 out of 0.50

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

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

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

Question **12**

Complete

Mark 1.00 out of 1.00

Match the register with the segment used with it.

ebp	ss
edi	es
esi	ds
eip	cs
esp	ss

The correct answer is: ebp → ss, edi → es, esi → ds, eip → cs, esp → ss

Question **13**

Complete

Mark 1.00 out of 1.00

Why should a program exist in memory before it starts executing ?

- a. Because the variables of the program are stored in memory
- b. Because the hard disk is a slow medium
- c. Because the memory is volatile
- [✓ d. Becase the processor can run instructions and access data only from memory]

The correct answer is: Becase the processor can run instructions and access data only from memory

Question **14**

Complete

Mark 0.00 out of 0.50

Is the command "cat README > done &" possible on xv6? (Note the & in the end)

- [✓ a. no]
- b. yes

The correct answer is: yes

Question **15**

Complete

Mark 1.00 out of 2.00

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

a. Process the # directives in a C program

b. Check the program for logical errors

[✓ c. Suggest alternative pieces of code that can be written]

d. Check the program for syntactical errors

e. Invoke the linker to link the function calls with their code, extern globals with their declaration

f. Convert high level langauge code to machine code

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

Question **16**

Complete

Mark 0.14 out of 2.00

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

[✓ a. The two lines in the beginning of each function, "push %ebp; mov %esp, %ebp", create space for local variables]

b. Paramters are pushed on the stack in left-right order

c. during execution of a function, ebp is pointing to the old ebp

[✓ d. Compiler may allocate more memory on stack than needed]

[✓ e. The return value is either stored on the stack or returned in the eax register]

f. Space for local variables is allocated by substracting the stack pointer inside the code of the caller function

[✓ g. Parameters may be passed in registers or on stack]

h. Space for local variables is allocated by substracting the stack pointer inside the code of the called function

i. The ebp pointers saved on the stack constitute a chain of activation records

[✓ j. Return address is one location above the ebp]

[✓ k. Parameters may be passed in registers or on stack]

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

◄ (Task) Compulsory xv6 task

Jump to...

(Optional Assignment) Shell Programming(Conformance tests) ►

Started on	Wednesday, 9 February 2022, 7:01:07 PM
State	Finished
Completed on	Wednesday, 9 February 2022, 7:56:13 PM
Time taken	55 mins 6 secs
Grade	**10.00** out of 11.00 (**91**%)

Question **1**

Not answered

Marked out of 0.50

Match the pairs of which action is taken by whom

Answer:

The correct answer is: kernel

Question **2**

Complete

Mark 1.00 out of 1.00

ELF Magic number is

- a. 0xELFELFELF
- b. 0
- c. 0x464C457FL
- d. 0xELF
- [✓ e. 0x464C457FU]
- f. 0x0x464CELF
- g. 0xFFFFFFFFF

The correct answer is: 0x464C457FU

Question **3**

Complete

Mark 1.00 out of 1.00

which of the following is not a difference between real mode and protected mode

- a. processor starts in real mode
- b. in real mode general purpose registers are 16 bit, in protected mode they are 32 bit
- c. in real mode the addressable memory is less than in protected mode
- d. in real mode the segment is multiplied by 16, in protected mode segment is used as index in GDT
- [✓ e. in real mode the addressable memory is more than in protected mode]

The correct answer is: in real mode the addressable memory is more than in protected mode

Question **4**

Complete

Mark 1.00 out of 1.00

The kernel is loaded at Physical Address

- a. 0x0010000
- b. 0x80100000
- [✓ c. 0x00100000]
- d. 0x80000000

The correct answer is: 0x00100000

Question **5**

Complete

Mark 1.00 out of 1.00

The kernel ELF file contains how many Program headers?

- a. 10
- [✓ b. 3]
- c. 9
- d. 4
- e. 2

The correct answer is: 3

Question **6**

Complete

Mark 1.00 out of 1.00

x86 provides which of the following type of memory management options?

- a. segmentation and two level paging
- b. segmentation or paging
- c. segmentation or one or two level paging
- d. segmentation and one level paging
- e. segmentation only
- [✓ f. segmentation and one or two level paging]

The correct answer is: segmentation and one or two level paging

Question **7**

Complete

Mark 1.00 out of 1.00

The variable $stack in entry.S is

- a. located at 0x7c00
- b. located at 0
- c. located at less than 0x7c00
- [✓ d. a memory region allocated as a part of entry.S]
- e. located at the value given by %esp as setup by bootmain()

The correct answer is: a memory region allocated as a part of entry.S

Question **8**

Complete

Mark 1.00 out of 1.00

The right side of line of code "entry = (void(*)(void))(elf->entry)" means

- a. Convert the "entry" in ELF structure into void
- b. Get the "entry" in ELF structure and convert it into a void pointer
- [✓ c. Get the "entry" in ELF structure and convert it into a function pointer accepting no arguments and returning nothing]
- d. Get the "entry" in ELF structure and convert it into a function void pointer

The correct answer is: Get the "entry" in ELF structure and convert it into a function pointer accepting no arguments and returning nothing

Question **9**

Complete

Mark 1.00 out of 1.00

The number of GDT entries setup during boot process of xv6 is

- [✓ a. 3]
- b. 2
- c. 4
- d. 256
- e. 255
- f. 0

The correct answer is: 3

Question **10**

Complete

Mark 1.00 out of 1.00

Why is the code of entry() in Assembly and not in C?

- a. There is no particular reason, it could also be in C
- [✓ b. Because it needs to setup paging]
- c. Because the kernel code must begin in assembly
- d. Because the symbol entry() is inside the ELF file

The correct answer is: Because it needs to setup paging

Question **11**

Not answered

Marked out of 0.50

code line, MMU setting: Match the line of xv6 code with the MMU setup employed

Answer:

The correct answer is: inb $0x64,%al

Question **12**

Complete

Mark 1.00 out of 1.00

The ljmp instruction in general does

- ○ a. change the CS and EIP to 32 bit mode
- ● b. change the CS and EIP to 32 bit mode, and jumps to new value of EIP [✓]
- ○ c. change the CS and EIP to 32 bit mode, and jumps to next line of code
- ○ d. change the CS and EIP to 32 bit mode, and jumps to kernel code

The correct answer is: change the CS and EIP to 32 bit mode, and jumps to new value of EIP

◄ Homework questions: Basics of MM, xv6 booting

Jump to...

(Code) Files, redirection, dup, (IPC)pipe ►

Dashboard / My courses / Computer Engineering & IT / CEIT-Even-sem-21-22 / OS-even-sem-21-22 / 14 February - 20 February / Topic-wise Quiz-3 (processes, trap handling, scheduler)

Started on	Monday, 21 February 2022, 7:00:49 PM
State	Finished
Completed on	Monday, 21 February 2022, 7:54:23 PM
Time taken	53 mins 34 secs
Grade	**8.53** out of 10.00 (**85**%)

Question **1**

Complete

Mark 1.00 out of 1.00

Select the odd one out

- a. Kernel stack of new process to Process stack of new process
- b. Process stack of running process to kernel stack of running process
- c. Kernel stack of scheduler to kernel stack of new process
- d. Kernel stack of running process to kernel stack of scheduler
- [✓ e. Kernel stack of new process to kernel stack of scheduler]

The correct answer is: Kernel stack of new process to kernel stack of scheduler

Question **2**

Complete

Mark 0.50 out of 0.50

Match the File descriptors to their meaning

1	Standard output
2	Standard error
0	Standard Input

The correct answer is: 1 → Standard output, 2 → Standard error, 0 → Standard Input

Question **3**

Complete

Mark 1.00 out of 1.00

What will be the output of this program

```
int main() {
  int fd;
  printf("%d ",   open("/etc/passwd", O_RDONLY));
  close(1);
  fd = printf("%d ", open("/etc/passwd", O_RDONLY));
  close(fd);
   fd = printf("%d ", open("/etc/passwd", O_RDONLY));
}
```

[✓ a. 3 1 1]

b. 3 1 2

c. 3 4 5

d. 3 3 3

e. 2 2 2

f. 1 1 1

The correct answer is: 3 1 1

Question **4**

Complete

Mark 0.70 out of 1.00

Match the elements of C program to their place in memory

#define MACROS	No memory needed
Malloced Memory	Heap
Code of main()	Code
Arguments	Stack
Local Variables	Stack
Global variables	Data
Local Static variables	Stack
Function code	Code
#include files	No Memory needed
Global Static variables	Data

The correct answer is: #define MACROS → No Memory needed, Malloced Memory → Heap, Code of main() → Code, Arguments → Stack, Local Variables → Stack, Global variables → Data, Local Static variables → Data, Function code → Code, #include files → No memory needed, Global Static variables → Data

Question **5**

Complete

Mark 0.00 out of 0.50

Which of the following state transitions are not possible?

- [✓ a. Ready -> Waiting]
- b. Running -> Waiting
- [✓ c. Ready -> Terminated]
- [✓ d. Waiting -> Terminated]

The correct answers are: Ready -> Terminated, Waiting -> Terminated, Ready -> Waiting

Question **6**

Complete

Mark 0.50 out of 1.00

Which of the following is not a task of the code of swtch() function

- a. Save the old context
- b. Save the return value of the old context code
- c. Load the new context
- [✓ d. Change the kernel stack location]
- e. Jump to next context EIP
- f. Switch stacks

The correct answers are: Save the return value of the old context code, Change the kernel stack location

Question **7**

Complete

Mark 1.00 out of 1.00

A process blocks itself means

- [✓ a. The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler]
- b. The kernel code of an interrupt handler, moves the process to a waiting queue and calls scheduler
- c. The application code calls the scheduler
- d. The kernel code of system call calls scheduler

The correct answer is: The kernel code of system call, called by the process, moves the process to a waiting queue and calls scheduler

Question **8**

Complete

Mark 1.00 out of 1.00

Arrange in correct order, the files involved in execution of system call

vectors.S	2
trapasm.S	3
usys.S	1
trap.c	4

The correct answer is: vectors.S → 2, trapasm.S → 3, usys.S → 1, trap.c → 4

Question **9**

Complete

Mark 0.50 out of 0.50

Match the names of PCB structures with kernel

linux	struct task_struct
xv6	struct proc

The correct answer is: linux → struct task_struct, xv6 → struct proc

Question **10**

Complete

Mark 0.33 out of 0.50

Match the MACRO with it's meaning

KERNLINK	2.1 GB
PHYSTOP	224 MB
KERNBASE	2 GB

The correct answer is: KERNLINK → 2.224 GB, PHYSTOP → 224 MB, KERNBASE → 2 GB

Question **11**

Complete

Mark 1.00 out of 1.00

The trapframe, in xv6, is built by the

- [✓ a. hardware, vectors.S, trapasm.S]
- b. vectors.S, trapasm.S
- c. hardware, trapasm.S
- d. hardware, vectors.S
- e. hardware, vectors.S, trapasm.S, trap()

The correct answer is: hardware, vectors.S, trapasm.S

Question **12**

Complete

Mark 1.00 out of 1.00

The "push 0" in vectors.S is

- a. To indicate that it's a system call and not a hardware interrupt
- b. A placeholder to match the size of struct trapframe
- c. To be filled in as the return value of the system call
- [✓ d. Place for the error number value]

The correct answer is: Place for the error number value

◄ Description of some possible course mini projects

Jump to...

(Code) mmap related programs ►

Dashboard / My courses / Computer Engineering & IT / CEIT-Even-sem-21-22 / OS-even-sem-21-22 / 21 February - 27 February / Topic-wise Quiz-4 (Virtual Memory)

Started on	Saturday, 26 February 2022, 5:17:38 PM
State	Finished
Completed on	Saturday, 26 February 2022, 6:32:37 PM
Time taken	1 hour 14 mins
Grade	**13.23** out of 15.00 (**88**%)

Question **1**

Complete

Mark 0.75 out of 1.00

Order the following events, related to page fault handling, in correct order

1. MMU detects that a page table entry is marked "invalid"
2. Page fault interrupt is generated
3. Page fault handler in kernel starts executing
4. Page fault handler detects that it's a page fault and not illegal memory access
5. Empty frame is found
6. Disk read is issued
7. Page faulting process is made to wait in a queue
8. Other processes scheduled by scheduler
9. Disk Interrupt occurs
10. Disk interrupt handler runs
11. Page faulted process is moved to ready-queue
12. Page table of page faulted process is updated

The correct order for these items is as follows:

1. MMU detects that a page table entry is marked "invalid"
2. Page fault interrupt is generated
3. Page fault handler in kernel starts executing
4. Page fault handler detects that it's a page fault and not illegal memory access
5. Empty frame is found
6. Disk read is issued
7. Page faulting process is made to wait in a queue
8. Other processes scheduled by scheduler
9. Disk Interrupt occurs
10. Disk interrupt handler runs
11. Page table of page faulted process is updated
12. Page faulted process is moved to ready-queue

Question **2**

Complete

Mark 1.00 out of 1.00

Given below is the "maps" file for a particular instance of "vim.basic" process.

Mark the given statements as True or False, w.r.t. the contents of the map file.

```
55a43501b000-55a435049000 r--p 00000000 103:05 917529                    /usr/bin/vim.basic
55a435049000-55a435248000 r-xp 0002e000 103:05 917529                    /usr/bin/vim.basic
55a435248000-55a4352b6000 r--p 0022d000 103:05 917529                    /usr/bin/vim.basic
55a4352b7000-55a4352c5000 r--p 0029b000 103:05 917529                    /usr/bin/vim.basic
55a4352c5000-55a4352e2000 rw-p 002a9000 103:05 917529                    /usr/bin/vim.basic
55a4352e2000-55a4352f0000 rw-p 00000000 00:00 0
55a436bc9000-55a436e5b000 rw-p 00000000 00:00 0                          [heap]
7f275b0a3000-7f275b0a6000 r--p 00000000 103:05 917901                    /usr/lib/x86_64-linux-
gnu/libnss_files-2.31.so
7f275b0a6000-7f275b0ad000 r-xp 00003000 103:05 917901                    /usr/lib/x86_64-linux-
gnu/libnss_files-2.31.so
7f275b0ad000-7f275b0af000 r--p 0000a000 103:05 917901                    /usr/lib/x86_64-linux-
gnu/libnss_files-2.31.so
7f275b0af000-7f275b0b0000 r--p 0000b000 103:05 917901                    /usr/lib/x86_64-linux-
gnu/libnss_files-2.31.so
7f275b0b0000-7f275b0b1000 rw-p 0000c000 103:05 917901                    /usr/lib/x86_64-linux-
gnu/libnss_files-2.31.so
7f275b0b1000-7f275b0b7000 rw-p 00000000 00:00 0
7f275b0b7000-7f275b8f5000 r--p 00000000 103:05 925247                    /usr/lib/locale/locale-archive
7f275b8f5000-7f275b8fa000 rw-p 00000000 00:00 0
7f275b8fa000-7f275b8fc000 r--p 00000000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b8fc000-7f275b901000 r-xp 00002000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b901000-7f275b904000 r--p 00007000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b904000-7f275b905000 ---p 0000a000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b905000-7f275b906000 r--p 0000a000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b906000-7f275b907000 rw-p 0000b000 103:05 924216                    /usr/lib/x86_64-linux-
gnu/libogg.so.0.8.4
7f275b907000-7f275b90a000 r--p 00000000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b90a000-7f275b921000 r-xp 00003000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b921000-7f275b932000 r--p 0001a000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b932000-7f275b933000 ---p 0002b000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b933000-7f275b934000 r--p 0002b000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b934000-7f275b935000 rw-p 0002c000 103:05 924627                    /usr/lib/x86_64-linux-
gnu/libvorbis.so.0.4.8
7f275b935000-7f275b937000 rw-p 00000000 00:00 0
7f275b937000-7f275b938000 r--p 00000000 103:05 917914                    /usr/lib/x86_64-linux-
gnu/libutil-2.31.so
7f275b938000-7f275b939000 r-xp 00001000 103:05 917914                    /usr/lib/x86_64-linux-
gnu/libutil-2.31.so
7f275b939000-7f275b93a000 r--p 00002000 103:05 917914                    /usr/lib/x86_64-linux-
gnu/libutil-2.31.so
7f275b93a000-7f275b93b000 r--p 00002000 103:05 917914                    /usr/lib/x86_64-linux-
gnu/libutil-2.31.so
7f275b93b000-7f275b93c000 rw-p 00003000 103:05 917914                    /usr/lib/x86_64-linux-
```

```
gnu/libutil-2.31.so
7f275b93c000-7f275b93e000 r--p 00000000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b93e000-7f275b94f000 r-xp 00002000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b94f000-7f275b955000 r--p 00013000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b955000-7f275b956000 ---p 00019000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b956000-7f275b957000 r--p 00019000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b957000-7f275b958000 rw-p 0001a000 103:05 915906                    /usr/lib/x86_64-linux-
gnu/libz.so.1.2.11
7f275b958000-7f275b95c000 r--p 00000000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b95c000-7f275b978000 r-xp 00004000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b978000-7f275b982000 r--p 00020000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b982000-7f275b983000 ---p 0002a000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b983000-7f275b985000 r--p 0002a000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b985000-7f275b986000 rw-p 0002c000 103:05 923645                    /usr/lib/x86_64-linux-
gnu/libexpat.so.1.6.11
7f275b986000-7f275b988000 r--p 00000000 103:05 924057                    /usr/lib/x86_64-linux-
gnu/libltdl.so.7.3.1
7f275b988000-7f275b98d000 r-xp 00002000 103:05 924057                    /usr/lib/x86_64-linux-
gnu/libltdl.so.7.3.1
7f275b98d000-7f275b98f000 r--p 00007000 103:05 924057                    /usr/lib/x86_64-linux-
gnu/libltdl.so.7.3.1
7f275b98f000-7f275b990000 r--p 00008000 103:05 924057                    /usr/lib/x86_64-linux-
gnu/libltdl.so.7.3.1
7f275b990000-7f275b991000 rw-p 00009000 103:05 924057                    /usr/lib/x86_64-linux-
gnu/libltdl.so.7.3.1
7f275b991000-7f275b995000 r--p 00000000 103:05 921934                    /usr/lib/x86_64-linux-
gnu/libtdb.so.1.4.3
7f275b995000-7f275b9a3000 r-xp 00004000 103:05 921934                    /usr/lib/x86_64-linux-
gnu/libtdb.so.1.4.3
7f275b9a3000-7f275b9a9000 r--p 00012000 103:05 921934                    /usr/lib/x86_64-linux-
gnu/libtdb.so.1.4.3
7f275b9a9000-7f275b9aa000 r--p 00017000 103:05 921934                    /usr/lib/x86_64-linux-
gnu/libtdb.so.1.4.3
7f275b9aa000-7f275b9ab000 rw-p 00018000 103:05 921934                    /usr/lib/x86_64-linux-
gnu/libtdb.so.1.4.3
7f275b9ab000-7f275b9ad000 rw-p 00000000 00:00 0
7f275b9ad000-7f275b9af000 r--p 00000000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9af000-7f275b9b4000 r-xp 00002000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9b4000-7f275b9b5000 r--p 00007000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9b5000-7f275b9b6000 ---p 00008000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9b6000-7f275b9b7000 r--p 00008000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9b7000-7f275b9b8000 rw-p 00009000 103:05 924631                    /usr/lib/x86_64-linux-
gnu/libvorbisfile.so.3.3.7
7f275b9b8000-7f275b9ba000 r--p 00000000 103:05 924277                    /usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0
7f275b9ba000-7f275ba1e000 r-xp 00002000 103:05 924277                    /usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0
```

```
7f275ba1e000-7f275ba46000 r--p 00066000 103:05 924277                    /usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0
7f275ba46000-7f275ba47000 r--p 0008d000 103:05 924277                    /usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0
7f275ba47000-7f275ba48000 rw-p 0008e000 103:05 924277                    /usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0
7f275ba48000-7f275ba6d000 r--p 00000000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275ba6d000-7f275bbe5000 r-xp 00025000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275bbe5000-7f275bc2f000 r--p 0019d000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275bc2f000-7f275bc30000 ---p 001e7000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275bc30000-7f275bc33000 r--p 001e7000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275bc33000-7f275bc36000 rw-p 001ea000 103:05 917893                    /usr/lib/x86_64-linux-
gnu/libc-2.31.so
7f275bc36000-7f275bc3a000 rw-p 00000000 00:00 0
7f275bc3a000-7f275bc41000 r--p 00000000 103:05 917906                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f275bc41000-7f275bc52000 r-xp 00007000 103:05 917906                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f275bc52000-7f275bc57000 r--p 00018000 103:05 917906                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f275bc57000-7f275bc58000 r--p 0001c000 103:05 917906                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f275bc58000-7f275bc59000 rw-p 0001d000 103:05 917906                    /usr/lib/x86_64-linux-
gnu/libpthread-2.31.so
7f275bc59000-7f275bc5d000 rw-p 00000000 00:00 0
7f275bc5d000-7f275bcce000 r--p 00000000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275bcce000-7f275bf29000 r-xp 00071000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275bf29000-7f275c142000 r--p 002cc000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275c142000-7f275c143000 ---p 004e5000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275c143000-7f275c149000 r--p 004e5000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275c149000-7f275c190000 rw-p 004eb000 103:05 917016                    /usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0
7f275c190000-7f275c1b3000 rw-p 00000000 00:00 0
7f275c1b3000-7f275c1b4000 r--p 00000000 103:05 917894                    /usr/lib/x86_64-linux-
gnu/libdl-2.31.so
7f275c1b4000-7f275c1b6000 r-xp 00001000 103:05 917894                    /usr/lib/x86_64-linux-
gnu/libdl-2.31.so
7f275c1b6000-7f275c1b7000 r--p 00003000 103:05 917894                    /usr/lib/x86_64-linux-
gnu/libdl-2.31.so
7f275c1b7000-7f275c1b8000 r--p 00003000 103:05 917894                    /usr/lib/x86_64-linux-
gnu/libdl-2.31.so
7f275c1b8000-7f275c1b9000 rw-p 00004000 103:05 917894                    /usr/lib/x86_64-linux-
gnu/libdl-2.31.so
7f275c1b9000-7f275c1bb000 rw-p 00000000 00:00 0
7f275c1bb000-7f275c1c0000 r-xp 00000000 103:05 923815                    /usr/lib/x86_64-linux-
gnu/libgpm.so.2
7f275c1c0000-7f275c3bf000 ---p 00005000 103:05 923815                    /usr/lib/x86_64-linux-
gnu/libgpm.so.2
7f275c3bf000-7f275c3c0000 r--p 00004000 103:05 923815                    /usr/lib/x86_64-linux-
gnu/libgpm.so.2
7f275c3c0000-7f275c3c1000 rw-p 00005000 103:05 923815                    /usr/lib/x86_64-linux-
gnu/libgpm.so.2
```

```
7f275c3c1000-7f275c3c3000 r--p 00000000 103:05 923315                    /usr/lib/x86_64-linux-
gnu/libacl.so.1.1.2253
7f275c3c3000-7f275c3c8000 r-xp 00002000 103:05 923315                    /usr/lib/x86_64-linux-
gnu/libacl.so.1.1.2253
7f275c3c8000-7f275c3ca000 r--p 00007000 103:05 923315                    /usr/lib/x86_64-linux-
gnu/libacl.so.1.1.2253
7f275c3ca000-7f275c3cb000 r--p 00008000 103:05 923315                    /usr/lib/x86_64-linux-
gnu/libacl.so.1.1.2253
7f275c3cb000-7f275c3cc000 rw-p 00009000 103:05 923315                    /usr/lib/x86_64-linux-
gnu/libacl.so.1.1.2253
7f275c3cc000-7f275c3cf000 r--p 00000000 103:05 923446                    /usr/lib/x86_64-linux-
gnu/libcanberra.so.0.2.5
7f275c3cf000-7f275c3d9000 r-xp 00003000 103:05 923446                    /usr/lib/x86_64-linux-
gnu/libcanberra.so.0.2.5
7f275c3d9000-7f275c3dd000 r--p 0000d000 103:05 923446                    /usr/lib/x86_64-linux-
gnu/libcanberra.so.0.2.5
7f275c3dd000-7f275c3de000 r--p 00010000 103:05 923446                    /usr/lib/x86_64-linux-
gnu/libcanberra.so.0.2.5
7f275c3de000-7f275c3df000 rw-p 00011000 103:05 923446                    /usr/lib/x86_64-linux-
gnu/libcanberra.so.0.2.5
7f275c3df000-7f275c3e5000 r--p 00000000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c3e5000-7f275c3fe000 r-xp 00006000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c3fe000-7f275c405000 r--p 0001f000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c405000-7f275c406000 ---p 00026000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c406000-7f275c407000 r--p 00026000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c407000-7f275c408000 rw-p 00027000 103:05 924431                    /usr/lib/x86_64-linux-
gnu/libselinux.so.1
7f275c408000-7f275c40a000 rw-p 00000000 00:00 0
7f275c40a000-7f275c418000 r--p 00000000 103:05 924540                    /usr/lib/x86_64-linux-
gnu/libtinfo.so.6.2
7f275c418000-7f275c427000 r-xp 0000e000 103:05 924540                    /usr/lib/x86_64-linux-
gnu/libtinfo.so.6.2
7f275c427000-7f275c435000 r--p 0001d000 103:05 924540                    /usr/lib/x86_64-linux-
gnu/libtinfo.so.6.2
7f275c435000-7f275c439000 r--p 0002a000 103:05 924540                    /usr/lib/x86_64-linux-
gnu/libtinfo.so.6.2
7f275c439000-7f275c43a000 rw-p 0002e000 103:05 924540                    /usr/lib/x86_64-linux-
gnu/libtinfo.so.6.2
7f275c43a000-7f275c449000 r--p 00000000 103:05 917895                    /usr/lib/x86_64-linux-
gnu/libm-2.31.so
7f275c449000-7f275c4f0000 r-xp 0000f000 103:05 917895                    /usr/lib/x86_64-linux-
gnu/libm-2.31.so
7f275c4f0000-7f275c587000 r--p 000b6000 103:05 917895                    /usr/lib/x86_64-linux-
gnu/libm-2.31.so
7f275c587000-7f275c588000 r--p 0014c000 103:05 917895                    /usr/lib/x86_64-linux-
gnu/libm-2.31.so
7f275c588000-7f275c589000 rw-p 0014d000 103:05 917895                    /usr/lib/x86_64-linux-
gnu/libm-2.31.so
7f275c589000-7f275c58b000 rw-p 00000000 00:00 0
7f275c5ae000-7f275c5af000 r--p 00000000 103:05 917889                    /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f275c5af000-7f275c5d2000 r-xp 00001000 103:05 917889                    /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f275c5d2000-7f275c5da000 r--p 00024000 103:05 917889                    /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f275c5db000-7f275c5dc000 r--p 0002c000 103:05 917889                    /usr/lib/x86_64-linux-gnu/ld-
2.31.so
```

```
7f275c5dc000-7f275c5dd000 rw-p 0002d000 103:05 917889                   /usr/lib/x86_64-linux-gnu/ld-2.31.so
7f275c5dd000-7f275c5de000 rw-p 00000000 00:00 0
7ffd22d2f000-7ffd22d50000 rw-p 00000000 00:00 0                          [stack]
7ffd22db0000-7ffd22db4000 r--p 00000000 00:00 0                          [vvar]
7ffd22db4000-7ffd22db6000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
```

True	False	
○	●	The size of the stack is one page
●	○	vim.basic uses the math library
●	○	The 5th entry 55a4352c5000-55a4352e2000 **may** correspond to "data" of the vim.basic
○	●	The size of the heap is one page
●	○	This is a virtual memory map (not physical memory map)

The size of the stack is one page: False
vim.basic uses the math library: True
The 5th entry 55a4352c5000-55a4352e2000 **may** correspond to "data" of the vim.basic: True
The size of the heap is one page: False
This is a virtual memory map (not physical memory map): True

Question **3**
Complete
Mark 1.00 out of 1.00

which of the following, do you think, are valid concerns for making the kernel pageable?

[✓ a. The page fault handler should not be pageable]
b. No part of kernel code should be pageable.
[✓ c. The kernel must have some dedicated frames for it's own work]
[✓ d. The kernel's own page tables should not be pageable]
[✓ e. The disk driver and disk interrupt handler should not be pageable]
f. No data structure of kernel should be pageable

The correct answers are: The kernel's own page tables should not be pageable, The page fault handler should not be pageable, The kernel must have some dedicated frames for it's own work, The disk driver and disk interrupt handler should not be pageable

Question **4**

Complete

Mark 0.50 out of 0.50

Map the technique with it's feature/problem

static loading	wastage of physical memory
dynamic loading	allocate memory only if needed
dynamic linking	small executable file
static linking	large executable file

The correct answer is: static loading → wastage of physical memory, dynamic loading → allocate memory only if needed, dynamic linking → small executable file, static linking → large executable file

Question **5**

Complete

Mark 0.50 out of 1.00

Select all the correct statements, w.r.t. Copy on Write

- [✓] a. Fork() used COW technique to improve performance of new process creation.
- [ ] b. use of COW during fork() is useless if child called exit()
- [✓] c. use of COW during fork() is useless if exec() is called by the child
- [✓] d. If either parent or child modifies a COW-page, then a copy of the page is made and page table entry is updated
- [✓] e. COW helps us save memory
- [✓] f. Vfork() assumes that there will be no write, but rather exec()

The correct answers are: Fork() used COW technique to improve performance of new process creation., If either parent or child modifies a COW-page, then a copy of the page is made and page table entry is updated, COW helps us save memory, Vfork() assumes that there will be no write, but rather exec()

Question **6**

Complete

Mark 0.86 out of 1.00

Compare paging with demand paging and select the correct statements.

Select one or more:

- ☐ a. TLB hit ration has zero impact in effective memory access time in demand paging.
- ☑ b. With demand paging, it's possible to have user programs bigger than physical memory.
- ☑ c. The meaning of valid-invalid bit in page table is different in paging and demand-paging.
- ☐ d. Demand paging always increases effective memory access time.
- ☑ e. Paging requires some hardware support in CPU
- ☐ f. Paging requires NO hardware support in CPU
- ☑ g. Demand paging requires additional hardware support, compared to paging.
- ☑ h. Both demand paging and paging support shared memory pages.
- ☐ i. With paging, it's possible to have user programs bigger than physical memory.
- ☑ j. Calculations of number of bits for page number and offset are same in paging and demand paging.

The correct answers are: Demand paging requires additional hardware support, compared to paging., Both demand paging and paging support shared memory pages., With demand paging, it's possible to have user programs bigger than physical memory., Demand paging always increases effective memory access time., Paging requires some hardware support in CPU, Calculations of number of bits for page number and offset are same in paging and demand paging., The meaning of valid-invalid bit in page table is different in paging and demand-paging.

Question **7**

Complete

Mark 1.00 out of 1.00

For the reference string
3 4 3 5 2

the number of page faults (including initial ones) using

FIFO replacement and 2 page frames is :

4

FIFO replacement and 3 page frames is :

4

Question **8**

Complete

Mark 1.00 out of 1.00

Page sizes are a power of 2 because

Select one:

- [✓ a. Certain bits are reserved for offset in logical address. Hence page size = 2^(no.of offset bits)]
- b. MMU only understands numbers that are power of 2
- c. operating system calculations happen using power of 2
- d. Power of 2 calculations are highly efficient
- e. Certain bits are reserved for offset in logical address. Hence page size = 2^(32 - no.of offset bits)

The correct answer is: Certain bits are reserved for offset in logical address. Hence page size = 2^(no.of offset bits)

Question **9**

Complete

Mark 1.00 out of 1.00

Shared memory is possible with which of the following memory management schemes ?

Select one or more:

- [✓ a. paging]
- [✓ b. segmentation]
- [✓ c. demand paging]
- d. continuous memory management

The correct answers are: paging, segmentation, demand paging

Question **10**

Complete

Mark 1.00 out of 1.00

Given below is the output of the command "ps -eo min_flt,maj_flt,cmd" on a Linux Desktop system. Select the statements that are consistent with the output

```
626729 482768 /usr/lib/firefox/firefox -contentproc -parentBuildID 20220202182137 -prefsLen 9256 -
prefMapSize 264738 -appDir /usr/lib/firefox/browser 6094 true rdd

2167 687 /usr/sbin/apache2 -k start

1265185 222 /usr/bin/gnome-shell

102648 111 /usr/sbin/mysqld

9813 0 bash

15497 370 /usr/bin/gedit --gapplication-service
```

- ✓ a. Apache web-server has not been doing much work
- ✓ b. All of the processes here exihibit some good locality of reference
- ✓ c. The bash shell is mostly busy doing work within a particular locality
- ✓ d. Firefox has likely been running for a large amount of time

The correct answers are: Firefox has likely been running for a large amount of time, Apache web-server has not been doing much work, The bash shell is mostly busy doing work within a particular locality, All of the processes here exihibit some good locality of reference

Question **11**

Complete

Mark 1.00 out of 1.00

W.r.t the figure given below, mark the given statements as True or False.

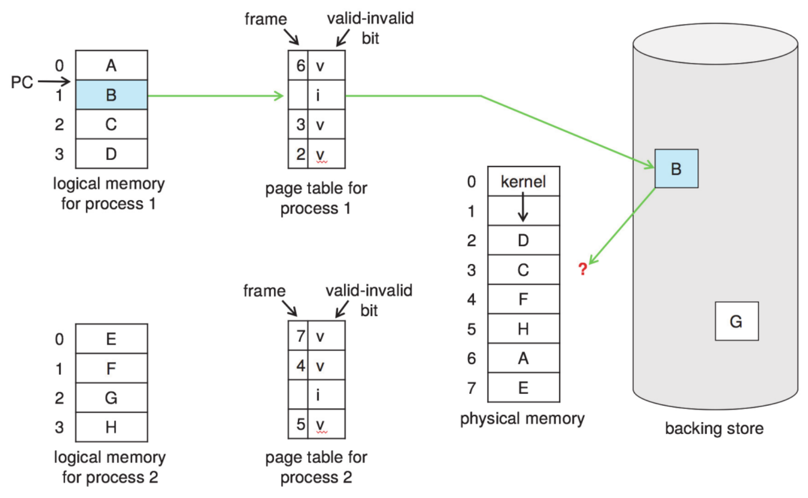

Figure 10.9 Need for page replacement.

True	False	
◉	○	Local replacement means chose any of the frames 2, 3, 6
◉	○	Handling this scenario demands two disk I/Os
◉	○	Page 1 of process 1 needs a replacement
◉	○	Kernel occupies two page frames
○	◉	Global replacement means chose any of the frame from 0 to 7
○	◉	Local replacement means chose any of the frame from 2 to 7
◉	○	Global replacement means chose any of the frame from 2 to 7
◉	○	The kernel's pages can not used for replacement if kernel is not pageable.

Local replacement means chose any of the frames 2, 3, 6: True
Handling this scenario demands two disk I/Os: True
Page 1 of process 1 needs a replacement: True
Kernel occupies two page frames: True
Global replacement means chose any of the frame from 0 to 7: False
Local replacement means chose any of the frame from 2 to 7: False
Global replacement means chose any of the frame from 2 to 7: True
The kernel's pages can not used for replacement if kernel is not pageable.: True

Question **12**

Complete

Mark 0.50 out of 0.50

Map the parts of a C code to the memory regions they are related to

functions	code
local variables	stack
global un-initialized variables	bss
global initialized variables	data
function arguments	stack
static variables	data
malloced memory	heap

The correct answer is: functions → code, local variables → stack, global un-initialized variables → bss, global initialized variables → data, function arguments → stack, static variables → data, malloced memory → heap

Question **13**

Complete

Mark 0.29 out of 1.00

Suppose two processes share a library between them. The library consists of 5 pages, and these 5 pages are mapped to frames 9, 15, 23, 4, 7 respectively. Process P1 has got 6 pages, first 3 of which consist of process's own code/data and 3 correspond to library's pages 0, 2, 4. Process P2 has got 7 pages, first 3 of which consist of processe's own code/data and remaining 4 correspond to library's pages 0, 1, 3, 4. Fill in the blanks for page table entries of P1 and P2.

Page table of P1, Page 4	15
Page table of P2, Page 0	6
Page table of P1, Page 3	9
Page table of P2, Page 3	9
Page table of P2, Page 4	15
Page table of P1, Page 5	7
Page table of P2, Page 1	3

The correct answer is: Page table of P1, Page 4 → 23, Page table of P2, Page 0 → 9, Page table of P1, Page 3 → 9, Page table of P2, Page 3 → 4, Page table of P2, Page 4 → 7, Page table of P1, Page 5 → 7, Page table of P2, Page 1 → 15

Question **14**

Complete

Mark 1.00 out of 1.00

Assuming a 8- KB page size, what is the page numbers for the address 1093943 reference in decimal :

(give answer also in decimal)

Answer: 133

The correct answer is: 134

Question **15**

Complete

Mark 0.83 out of 1.00

Given six memory partitions of 300 KB , 600 KB , 350 KB , 200 KB , 750 KB , and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB and 500 KB (in order)?

first fit 500 KB	600 KB
best fit 115 KB	125 KB
worst fit 115 KB	750 KB
first fit 115 KB	300 KB
best fit 500 KB	600 KB
worst fit 500 KB	600 KB

The correct answer is: first fit 500 KB → 600 KB, best fit 115 KB → 125 KB, worst fit 115 KB → 750 KB, first fit 115 KB → 300 KB, best fit 500 KB → 600 KB, worst fit 500 KB → 635 KB

Question **16**

Complete

Mark 1.00 out of 1.00

Calculate the EAT in NANO-seconds (upto 2 decimal points) w.r.t. a page fault, given

Memory access time = 183 ns

Average page fault service time = 11 ms

Page fault rate = 0.4

Answer: 4400000.0

The correct answer is: 4400109.80

◄ (Code) mmap related programs

Jump to...

Points from Mid-term feedback ►

Started on	Monday, 7 March 2022, 7:00:32 PM
State	Finished
Completed on	Monday, 7 March 2022, 8:09:22 PM
Time taken	1 hour 8 mins
Grade	**9.50** out of 15.00 (**63**%)

Question **1**

Complete

Mark 0.00 out of 2.00

exec() does this: curproc->tf->eip = elf.entry, but userinit() does this: p->tf->eip = 0; Select all the statements from below, that collectively explain this

- ✓ a. elf.entry is anyways 0, so both statements mean the same
- ✓ b. In userinit() the function inituvm() has mapped the code of 'initcode' to be starting at virtual address 0
- ✓ c. exec() loads from ELF file and the address of first instruction to be executed is given by 'entry'
- ☐ d. the code of 'initcode' is loaded at physical address 0
- ✓ e. the 'entry' in initcode is anyways 0
- ☐ f. the initcode is created using objcopy, which discards all relocation information and symbols (like entry)

The correct answers are: exec() loads from ELF file and the address of first instruction to be executed is given by 'entry', In userinit() the function inituvm() has mapped the code of 'initcode' to be starting at virtual address 0, the initcode is created using objcopy, which discards all relocation information and symbols (like entry)

Question **2**

Complete

Mark 1.50 out of 1.50

Which of the following is DONE by allocproc() ?

- ✓ a. setup the trapframe and context pointers appropriately
- ☐ b. setup kernel memory mappings for the process
- ☐ c. setup the contents of the trapframe of the process properly
- ✓ d. allocate kernel stack for the process
- ✓ e. ensure that the process starts in forkret()
- ✓ f. Select an UNUSED struct proc for use
- ☐ g. ensure that the process starts in trapret()
- ✓ h. allocate PID to the process

The correct answers are: Select an UNUSED struct proc for use, allocate PID to the process, allocate kernel stack for the process, setup the trapframe and context pointers appropriately, ensure that the process starts in forkret()

Question **3**

Complete

Mark 1.00 out of 1.00

The variable 'end' used as argument to kinit1 has the value

- a. 80110000
- [✓ b. 801154a8]
- c. 80000000
- d. 80102da0
- e. 81000000
- f. 8010a48c

The correct answer is: 801154a8

Question **4**

Complete

Mark 1.00 out of 1.00

Does exec() code around clearptau() lead to wastage of one page frame?

- a. no
- [✓ b. yes]

The correct answer is: yes

Question **5**

Complete

Mark 0.75 out of 1.00

Map the virtual address to physical address in xv6

0xFE000000	0x80000000
80108000	0x108000
KERNLINK	0x100000
KERNBASE	0

The correct answer is: 0xFE000000 → 0xFE000000, 80108000 → 0x108000, KERNLINK → 0x100000, KERNBASE → 0

Question **6**

Complete

Mark 1.50 out of 1.50

Arrange the following in the correct order of execution (w.r.t. 'init')

userinit() is called	1
scheduler() schedules initcode() process	5
initcode() returns from trapret()	7
'initcode' process is marked RUNNABLE	3
mpmain() calls scheduler()	4
initcode() returns in forkret()	6
initcode() calls exec("/init", ...)	8
'initcode' struct proc is created	2

The correct answer is: userinit() is called → 1, scheduler() schedules initcode() process → 5, initcode() returns from trapret() → 7, 'initcode' process is marked RUNNABLE → 3, mpmain() calls scheduler() → 4, initcode() returns in forkret() → 6, initcode() calls exec("/init", ...) → 8, 'initcode' struct proc is created → 2

Question **7**

Complete

Mark 0.08 out of 1.00

Select all the correct statements about initcode

- ☐ a. code of 'initcode' is loaded along with the kernel during booting
- ☑ b. code of initcode is loaded in memory by the kernel during userinit()
- ☑ c. the size of 'initcode' is 2c
- ☑ d. code of initcode is loaded at virtual address 0
- ☑ e. The data and stack of initcode is mapped to one single page in userinit()
- ☐ f. initcode is the 'init' process
- ☑ g. initcode essentially calls exec("/init",...)

The correct answers are: code of 'initcode' is loaded along with the kernel during booting, the size of 'initcode' is 2c, The data and stack of initcode is mapped to one single page in userinit(), initcode essentially calls exec("/init",...)

Question **8**

Complete

Mark 0.67 out of 1.00

Which of the following is done by mappages()?

- ☐ a. create page table mappings to the range given by "pa" and "pa + size"
- ☐ b. allocate page frame if required
- ☑ c. create page table mappings for the range given by "va" and "va + size"
- ☑ d. allocate page table if required
- ☐ e. allocate page directory if required

The correct answers are: create page table mappings for the range given by "va" and "va + size", allocate page table if required, create page table mappings to the range given by "pa" and "pa + size"

Question **9**

Complete

Mark 1.00 out of 1.00

Why is there a call to kinit2? Why is it not merged with knit1?

- ○ a. When kinit1() is called there is a need for few page frames, but later knit2() is called to serve need of more page frames
- ● b. knit2 refers to virtual addresses beyond 4MB, which are not mapped before kvalloc() is called
- ○ c. Because there is a limit on the values that the argumets to knit1() can take.
- ○ d. call to seginit() makes it possible to actually use PHYSTOP in argument to kinit2()

The correct answer is: knit2 refers to virtual addresses beyond 4MB, which are not mapped before kvalloc() is called

Question **10**

Complete

Mark 0.00 out of 1.00

The approximate number of page frames created by kinit1 is

- ○ a. 2000
- ○ b. 4000
- ○ c. 3000
- ● d. 1000
- ○ e. 16
- ○ f. 10
- ○ g. 4

The correct answer is: 3000

Question **11**

Complete

Mark 1.00 out of 1.00

What does userinit() do ?

- ○ a. initializes the process 'init' and starts executing it
- ○ b. sets up the 'init' process to start execution in forkret()
- ○ c. initializes the users
- [✓ d. sets up the 'initcode' process to start execution in forkret()]
- ○ e. sets up the 'initcode' process to start execution in trapret()
- ○ f. sets up the 'initcode' process to start execution in forkret ()

The correct answer is: sets up the 'initcode' process to start execution in forkret()

Question **12**

Complete

Mark 0.00 out of 1.00

Select the statement that most correctly describes what setupkvm() does

- [✓ a. creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray and makes kpgdir point to it]
- ○ b. creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray
- ○ c. creates a 1-level page table for the use by the kernel, as specified in kmap[] global array
- ○ d. creates a 2-level page table for the use of the kernel, as specified in gdtdesc

The correct answer is: creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray

Question **13**

Complete

Mark 1.00 out of 1.00

What does seginit() do?

- ○ a. Nothing significant, just repetition of earlier GDT setup but with 2-level paging setup done
- ○ b. Adds two additional entries to GDT corresponding to Code and Data segments, but to be used in privilege level 0
- ○ c. Nothing significant, just repetition of earlier GDT setup but with kernel page table allocated now
- ○ d. Nothing significant, just repetition of earlier GDT setup but with free frames list created now
- [✓ e. Adds two additional entries to GDT corresponding to Code and Data segments, but to be used in privilege level 3]

The correct answer is: Adds two additional entries to GDT corresponding to Code and Data segments, but to be used in privilege level 3

◄ Questions for test on kalloc/kfree/kvmalloc, etc.

Jump to...

(Optional Assignment) Slab allocator in xv6 ►

Started on	Saturday, 12 February 2022, 10:00:55 AM
State	Finished
Completed on	Saturday, 12 February 2022, 11:57:03 AM
Time taken	1 hour 56 mins
Grade	**5.20** out of 10.00 (**52**%)

Question **1**

Complete

Mark 0.50 out of 0.50

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

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

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

Question **2**

Complete

Mark 0.00 out of 0.50

What's the trapframe in xv6?

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

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 **3**

Complete

Mark 0.21 out of 0.50

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

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 **4**

Complete

Mark 0.18 out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

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 **5**

Complete

Mark 0.10 out of 0.50

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

readseg((uchar*)elf, 4096, 0); in bootmain.c	Immaterial as the stack is not used here
cli in bootasm.S	The 4KB area in kernel image, loaded in memory, named as 'stack'
ljmp $(SEG_KCODE<<3), $start32 in bootasm.S	0x7c00 to 0x10000
jmp *%eax in entry.S	0x10000 to 0x7c00
call bootmain in bootasm.S	0x7c00 to 0

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

Question **6**

Not answered

Marked out of 1.00

Which parts of the xv6 code in bootasm.S bootmain.c , entry.S and in the codepath related to scheduler() and trap handling() can also be written in some other way, and still ensure that xv6 works properly?

Writing code is not necessary. You only need to comment on which part of the code could be changed to something else or written in another fashion.

Maximum two points to be written.

Question **7**

Complete

Mark 0.25 out of 0.50

The bootmain() function has this code

```
elf = (struct elfhdr*)0x10000;  // scratch space
readseg((uchar*)elf, 4096, 0);
```

Mark the statements as True or False with respect to this code.

In these statements 0x1000 is referred to as ADDRESS

True	False	
○	●	It the value of ADDRESS is changed to a higher number (upto a limit), the program could still work
○	●	If the value of ADDRESS is changed, then the program will not work
●	○	It the value of ADDRESS is changed to a lower number (upto a limit), the program could still work
●	○	The value ADDRESS is changed to a 0 the program could still work
●	○	This line effectively loads the ELF header and the program headers at ADDRESS
○	●	This line loads the kernel code at ADDRESS

It the value of ADDRESS is changed to a higher number (upto a limit), the program could still work: True
If the value of ADDRESS is changed, then the program will not work: False
It the value of ADDRESS is changed to a lower number (upto a limit), the program could still work: True
The value ADDRESS is changed to a 0 the program could still work: False
This line effectively loads the ELF header and the program headers at ADDRESS: False
This line loads the kernel code at ADDRESS: False

Question **8**

Complete

Mark 0.50 out of 0.50

Suppose a program does a scanf() call.

Essentially the scanf does a read() system call.

This call will obviously "block" waiting for the user input.

In terms of OS data structures and execution of code, what does it mean?

Select one:

- a. read() returns and process calls scheduler()
- b. read() will return and process will be taken to a wait queue
- [✓ c. OS code for read() will move PCB of current process to a wait queue and call scheduler]
- d. OS code for read() will call scheduler
- e. OS code for read() will move the PCB of this process to a wait queue and return from the system call

The correct answer is: OS code for read() will move PCB of current process to a wait queue and call scheduler

Question **9**

Complete

Mark 0.80 out of 1.00

Mark the statements, w.r.t. the scheduler of xv6 as True or False

True	False	
○	●	`sched()` *calls* `scheduler()` and `scheduler()` *calls* `sched()`
○	●	The variable c->scheduler on first processor uses the stack allocated entry.S
○	●	The work of selecting and scheduling a process is done only in `scheduler()` and not in `sched()`
●	○	`swtch` is a function that does not return to the caller
○	●	the control returns to `mycpu()->intena = intena; ();` after `swtch(&p->context, mycpu()->scheduler);` in `sched()`
○	●	the control returns to `switchkvm();` after `swtch(&(c->scheduler), p->context);` in `scheduler()`
●	○	The function `scheduler()` executes using the kernel-only stack
●	○	`swtch` is a function that saves old context, loads new context, and returns to last EIP in the new context
●	○	`sched()` and `scheduler()` are co-routines
●	○	When a process is scheduled for execution, it resumes execution in `sched()` after the call to `swtch()`

`sched()` *calls* `scheduler()` and `scheduler()` *calls* `sched()`: False
The variable c->scheduler on first processor uses the stack allocated entry.S: True
The work of selecting and scheduling a process is done only in `scheduler()` and not in `sched()`: True
`swtch` is a function that does not return to the caller: True
the control returns to `mycpu()->intena = intena; ();` after `swtch(&p->context, mycpu()->scheduler);` in `sched()`: False
the control returns to `switchkvm();` after `swtch(&(c->scheduler), p->context);` in `scheduler()`: False
The function `scheduler()` executes using the kernel-only stack: True
`swtch` is a function that saves old context, loads new context, and returns to last EIP in the new context: True
`sched()` and `scheduler()` are co-routines: True
When a process is scheduled for execution, it resumes execution in `sched()` after the call to `swtch()`
: True

Question **10**

Complete

Mark 0.36 out of 0.50

Order the events that occur on a timer interrupt:

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

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

Question **11**

Complete

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

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

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 12

Complete

Mark 0.50 out of 0.50

Consider the following programs

exec1.c

```
#include <unistd.h>
#include <stdio.h>
int main() {
    execl("./exec2", "./exec2", NULL);
}
```

exec2.c

```
#include <unistd.h>
#include <stdio.h>
int main() {
    execl("/bin/ls", "/bin/ls", NULL);

   printf("hello\n");
}
```

Compiled as

```
cc    exec1.c   -o exec1
cc    exec2.c   -o exec2
```

And run as

```
$ ./exec1
```

Explain the output of the above command (./exec1)

Assume that /bin/ls , i.e. the 'ls' program exists.

Select one:

- a. Execution fails as the call to execl() in exec2 fails
- b. Execution fails as the call to execl() in exec1 fails
- [✓ c. "ls" runs on current directory]
- d. Program prints hello
- e. Execution fails as one exec can't invoke another exec

The correct answer is: "ls" runs on current directory

Question **13**

Complete

Mark 0.50 out of 0.50

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

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

Question **14**

Complete

Mark 0.00 out of 1.00

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

(Note: non-interruptible kernel code means, if the kernel code is executing, then interrupts will be disabled).

Note: A possible sequence may have some missing steps in between. An impossible sequence will will have n and n+1th steps such that n+1th step can not follow n'th step.

Select one or more:

☐ a. 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

☑ b. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running

☐ 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
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

☐ e.
P1 running
P1 makes sytem call
Scheduler
P2 running
P2 makes sytem call and blocks
Scheduler
P1 running again

☑ f. P1 running

P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheuler
P1 running
P1's system call return

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 **15**

Complete

Mark 0.40 out of 0.50

Select all the correct statements about zombie processes

Select one or more:

- [ ] a. A process can become zombie if it finishes, but the parent has finished before it
- [✓] b. A zombie process occupies space in OS data structures
- [✓] c. If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent
- [✓] d. init() typically keeps calling wait() for zombie processes to get cleaned up
- [ ] e. A process becomes zombie when it's parent finishes
- [✓] f. A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it
- [ ] g. A zombie process remains zombie forever, as there is no way to clean it up
- [ ] h. Zombie processes are harmless even if OS is up for long time

The correct answers are: A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it, A process can become zombie if it finishes, but the parent has finished before it, A zombie process occupies space in OS data structures, If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent, init() typically keeps calling wait() for zombie processes to get cleaned up

Question **16**

Complete

Mark 0.40 out of 0.50

Select Yes if the mentioned element should be a part of PCB

Select No otherwise.

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

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

◄ Extra Reading on Linkers: A writeup by Ian Taylor (keep changing url string from 38 to 39, and so on)

Jump to...

(Code) IPC - Shm, Messages ►

Started on	Tuesday, 22 March 2022, 1:59:36 PM
State	Finished
Completed on	Tuesday, 22 March 2022, 5:20:58 PM
Time taken	3 hours 21 mins
Grade	**25.84** out of 40.00 (**65**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Select the most common causes of use of IPC by processes

- [✓ a. More modular code] ✓
- [✓ b. Sharing of information of common interest] ✓
- c. More security checks
- [✓ d. Breaking up a large task into small tasks and speeding up computation, on multiple core machines] ✓
- e. Get the kernel performance statistics

The correct answers are: Sharing of information of common interest, Breaking up a large task into small tasks and speeding up computation, on multiple core machines, More modular code

Question **2**

Partially correct

Mark 0.80 out of 1.00

Mark the statements about named and un-named pipes as True or False

True	False		
● ✓	○ ✗	Named pipes can exist beyond the life-time of processes using them.	✓
○ ✗	● ✓	The pipe() system call can be used to create either a named or un-named pipe.	✓
○ ✗	● ✓	Named pipes can be used for communication between only "related" processes.	✓
● ✓	○ ✗	Un-named pipes are inherited by a child process from parent.	✓
○ ✓	● ✗	Both types of pipes provide FIFO communication.	✗
● ✓	○ ✗	Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.	✓
● ✓	○ ✗	Both types of pipes are an extension of the idea of "message passing".	✓
○ ✗	● ✓	A named pipe has a name decided by the kernel.	✓
● ✗	○ ✓	The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.	✗
● ✓	○ ✗	Named pipe exists as a file	✓

Named pipes can exist beyond the life-time of processes using them.: True
The pipe() system call can be used to create either a named or un-named pipe.: False
Named pipes can be used for communication between only "related" processes.: False
Un-named pipes are inherited by a child process from parent.: True
Both types of pipes provide FIFO communication.: True
Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.: True
Both types of pipes are an extension of the idea of "message passing".: True
A named pipe has a name decided by the kernel.: False
The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.: False
Named pipe exists as a file: True

Question **3**

Correct

Mark 1.00 out of 1.00

For the reference string
3 4 3 5 2

using LRU replacement policy for pages,

consider the number of page faults for 2, 3 and 4 page frames.
Select the most correct statement.

Select one:

[✓ a. LRU will never exhibit Balady's anomaly] ✓

b. Exhibit Balady's anomaly between 2 and 3 frames

c. This example does not exhibit Balady's anomaly

d. Exhibit Balady's anomaly between 3 and 4 frames

Your answer is correct.

The correct answer is: LRU will never exhibit Balady's anomaly

Question **4**

Correct

Mark 1.00 out of 1.00

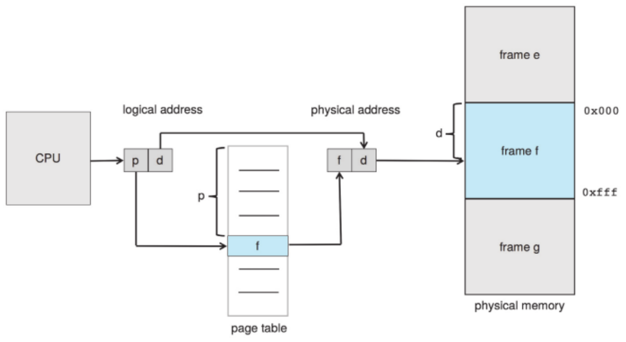

Figure 9.8 Paging hardware.

Mark the statements as True or False, w.r.t. the above diagram (note that the diagram does not cover all details of what actually happens!)

True	False		
● ✓	○ ✗	The logical address issued by CPU is the same one generated by compiler	✓
○ ✗	● ✓	There are total 3 memory references in this diagram	✓
● ✓	○ ✗	The page table is in physical memory and must be continuous	✓
● ✓	○ ✗	The combining of f and d is done by MMU	✓
○ ✗	● ✓	Using the offset d in the physical page-frame is done by MMU	✓
● ✓	○ ✗	The split of logical address into p and d is done by MMU	✓

The logical address issued by CPU is the same one generated by compiler: True
There are total 3 memory references in this diagram: False
The page table is in physical memory and must be continuous: True
The combining of f and d is done by MMU: True
Using the offset d in the physical page-frame is done by MMU: False
The split of logical address into p and d is done by MMU: True

Question **5**

Correct

Mark 1.00 out of 1.00

Map the functionality/use with function/variable in xv6 code.

Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices	setupkvm() ✓
Setup kernel part of a page table, and switch to that page table	kvmalloc() ✓
Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed	mappages() ✓
Array listing the kernel memory mappings, to be used by setupkvm()	kmap[] ✓
return a free page, if available; 0, otherwise	kalloc() ✓
Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary	walkpgdir() ✓

Your answer is correct.

The correct answer is: Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices → setupkvm(), Setup kernel part of a page table, and switch to that page table → kvmalloc(), Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed → mappages(), Array listing the kernel memory mappings, to be used by setupkvm() → kmap[], return a free page, if available; 0, otherwise → kalloc(), Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary → walkpgdir()

Question **6**

Partially correct

Mark 0.60 out of 1.00

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.
- ☑ b. Dynamic linking essentially results in relocatable code. ✓
- ☐ c. Static linking leads to non-relocatable code
- ☑ d. Loader is part of the operating system ✓
- ☐ e. Dynamic linking and loading is not possible without demand paging or demand segmentation.
- ☑ f. Continuous memory management schemes can support static linking and static loading. (may be inefficiently) ✓
- ☐ g. Continuous memory management schemes can support dynamic linking and dynamic loading.
- ☐ h. Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently)
- ☐ i. Loader is last stage of the linker program

Your answer is partially correct.

You have correctly selected 3.
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 **7**

Partially correct

Mark 0.60 out of 1.00

Mark the statements as True or False, w.r.t. thrashing

True	False		
○ ✗	● ✓	Thrashing occurs because some process is doing lot of disk I/O.	✓
● ✓	○ ✗	Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.	✓
○ ✓	● ✗	Thrashing is particular to demand paging systems, and does not apply to pure paging systems.	✗
○ ✗	● ✓	Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.	✓
● ✗	○ ✓	mmap() solves the problem of thrashing.	✗
● ✓	○ ✗	The working set model is an attempt at approximating the locality of a process.	✓
● ✓	○ ✗	Thrashing can be limited if local replacement is used.	✓
● ✓	○ ✗	During thrashing the CPU is under-utilised as most time is spent in I/O	✓
● ✗	○ ✓	Thrashing can occur even if entire memory is not in use.	✗
○ ✓	● ✗	Thrashing occurs when the total size of all processe's locality exceeds total memory size.	✗

Thrashing occurs because some process is doing lot of disk I/O.: False
Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.: True
Thrashing is particular to demand paging systems, and does not apply to pure paging systems.: True
Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.: False
mmap() solves the problem of thrashing.: False
The working set model is an attempt at approximating the locality of a process.: True
Thrashing can be limited if local replacement is used.: True
During thrashing the CPU is under-utilised as most time is spent in I/O: True
Thrashing can occur even if entire memory is not in use.: False
Thrashing occurs when the total size of all processe's locality exceeds total memory size.: True

Question **8**

Partially correct

Mark 0.80 out of 1.00

Mark statements True/False w.r.t. change of states of a process. Note that a statement is true only if the claim and argument both are true.

Reference: The process state diagram (and your understanding of how kernel code works). Note - the diagram does not show zombie state!

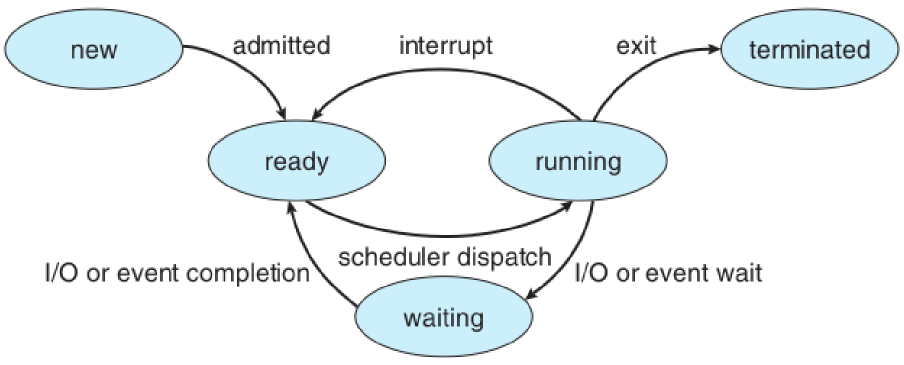

Figure 3.2 Diagram of process state.

True	False		
● ✓	○ ✗	Every forked process has to go through ZOMBIE state, at least for a small duration.	✓
● ✓	○ ✗	Only a process in READY state is considered by scheduler	✓
● ✗	○ ✓	A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.	✗
○ ✗	● ✓	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first	✓
● ✓	○ ✗	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet	✓

Every forked process has to go through ZOMBIE state, at least for a small duration.: True
Only a process in READY state is considered by scheduler: True
A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.: False
A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False
A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

Question **9**

Partially correct

Mark 0.75 out of 1.00

Select the correct points of comparison between POSIX and System V shared memory.

- ☐ a. POSIX allows giving name to shared memory, System V does not
- ☑ b. POSIX shared memory is "thread safe", System V is not ✓
- ☑ c. System V is more prevalent than POSIX even today ✓
- ☑ d. POSIX shared memory is newer than System V shared memory ✓

The correct answers are: POSIX shared memory is newer than System V shared memory, POSIX shared memory is "thread safe", System V is not, POSIX allows giving name to shared memory, System V does not, System V is more prevalent than POSIX even today

Question **10**

Correct

Mark 1.00 out of 1.00

Given that a kernel has 1000 KB of total memory, and holes of sizes (in that order) 300 KB, 200 KB, 100 KB, 250 KB. For each of the requests on the left side, match it with the chunk chosen using the specified algorithm.

Consider each request as first request.

200 KB, first fit	300 KB	✓
100 KB, worst fit	300 KB	✓
150 KB, best fit	200 KB	✓
150 KB, first fit	300 KB	✓
50 KB, worst fit	300 KB	✓
220 KB, best fit	250 KB	✓

The correct answer is: 200 KB, first fit → 300 KB, 100 KB, worst fit → 300 KB, 150 KB, best fit → 200 KB, 150 KB, first fit → 300 KB, 50 KB, worst fit → 300 KB, 220 KB, best fit → 250 KB

Question **11**

Partially correct

Mark 0.89 out of 1.00

Mark the statements as True or False, w.r.t. mmap()

True	False		
○✗	●✓	mmap() results in changes to buffer-cache of the kernel.	✓
●✓	○✗	mmap() can be implemented on both demand paged and non-demand paged systems.	✓
●✓	○✗	mmap() is a system call	✓
○✗	●✓	on failure mmap() returns NULL	✓
○✗	●✓	MAP_SHARED leads to a mapping that is copy-on-write	✓
●✓	○✗	mmap() results in changes to page table of a process.	✓
●✓	○✗	MAP_PRIVATE leads to a mapping that is copy-on-write	✓
●✓	○✗	on failure mmap() returns (void *)-1	✓
●✗	○✓	MAP_FIXED guarantees that the mapping is always done at the specified address	✗

mmap() results in changes to buffer-cache of the kernel.: False
mmap() can be implemented on both demand paged and non-demand paged systems.: True
mmap() is a system call: True
on failure mmap() returns NULL: False
MAP_SHARED leads to a mapping that is copy-on-write: False
mmap() results in changes to page table of a process.: True
MAP_PRIVATE leads to a mapping that is copy-on-write: True
on failure mmap() returns (void *)-1: True
MAP_FIXED guarantees that the mapping is always done at the specified address: False

Question **12**

Partially correct

Mark 0.80 out of 1.00

Choice of the global or local replacement strategy is a subjective choice for kernel programmers. There are advantages and disadvantages on either side. Out of the following statements, that advocate either global or local replacement strategy, select those statements that have a logically CONSISTENT argument. (That is any statement that is logically correct about either global or local replacement)

Consistent	Inconsistent		
● ✓	○ ✗	Global replacement can be preferred when greater throughput (number of processes completing per unit time) is a concern, because each process tries to complete at the expense of others, thus leading to overall more processes completing (unless thrashing occurs).	✓
○ ✓	● ✗	Local replacement can lead to under-utilisation of memory, because a process may not use all the pages allocated to it all the time.	✗
● ✓	○ ✗	Local replacement results in more predictable per-process completion time because number of page faults can be better predicted.	✓
● ✓	○ ✗	Local replacement can be preferred when avoiding thrashing is a major concern because with local replacement and minimum number of frames allocated, a process is always able to progress and cascading inter-process page faults are avoided.	✓
● ✓	○ ✗	Global replacement may give highly variable per process completion time because number of page faults become un-predictable.	✓

Global replacement can be preferred when greater throughput (number of processes completing per unit time) is a concern, because each process tries to complete at the expense of others, thus leading to overall more processes completing (unless thrashing occurs).: Consistent
Local replacement can lead to under-utilisation of memory, because a process may not use all the pages allocated to it all the time.: Consistent
Local replacement results in more predictable per-process completion time because number of page faults can be better predicted.: Consistent
Local replacement can be preferred when avoiding thrashing is a major concern because with local replacement and minimum number of frames allocated, a process is always able to progress and cascading inter-process page faults are avoided.: Consistent
Global replacement may give highly variable per process completion time because number of page faults become un-predictable.: Consistent

Question **13**

Partially correct

Mark 0.50 out of 1.00

Select all correct statements w.r.t. Major and Minor page faults on Linux

- [✓] a. Major page faults are likely to occur in more numbers at the beginning of the process ✓
- [ ] b. Thrashing is possible only due to major page faults
- [ ] c. Minor page fault may occur because the page was freed, but still tagged and available in the free page list
- [✓] d. Minor page fault may occur because the page was a shared memory page ✓
- [ ] e. Minor page fault may occur because of a page fault during fork(), on code of an already running process
- [✓] f. Minor page faults are an improvement of the page buffering techniques ✓

The correct answers are: Minor page fault may occur because the page was a shared memory page, Minor page fault may occur because of a page fault during fork(), on code of an already running process, Minor page fault may occur because the page was freed, but still tagged and available in the free page list, Major page faults are likely to occur in more numbers at the beginning of the process, Thrashing is possible only due to major page faults, Minor page faults are an improvement of the page buffering techniques

Question **14**

Partially correct

Mark 0.91 out of 2.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB onlyMark statements True or False

True	False		
● ✓	○ ✗	The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context	✓
● ✓	○ ✗	The process's address space gets mapped on frames, obtained from ~2MB:224MB range	✓
● ✗	○ ✓	The switchkvm() call in scheduler() changes CR3 to use page directory of new process	✗
● ✓	○ ✗	The stack allocated in entry.S is used as stack for scheduler's context for first processor	✓
○ ✓	● ✗	The kernel code and data take up less than 2 MB space	✗
○ ✗	● ✓	The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context	✓
○ ✓	● ✗	The free page-frame are created out of nearly 222 MB	✗
● ✗	○ ✓	The kernel's page table given by kpgdir variable is used as stack for scheduler's context	✗
○ ✓	● ✗	xv6 uses physical memory upto 224 MB only	✗
○ ✓	● ✗	The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir	✗
● ✓	○ ✗	PHYSTOP can be increased to some extent, simply by editing memlayout.h	✓

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True
The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True
The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False
The stack allocated in entry.S is used as stack for scheduler's context for first processor: True
The kernel code and data take up less than 2 MB space: True
The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False
The free page-frame are created out of nearly 222 MB: True
The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False
xv6 uses physical memory upto 224 MB only: True
The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True
PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

Question **15**

Incorrect

Mark 0.00 out of 1.00

Given below is a sequence of reference bits on pages before the second chance algorithm runs. Before the algorithm runs, the counter is at the page marked (x). Write the sequence of reference bits after the second chance algorithm has executed once. In the answer write PRECISELY one space BETWEEN each number and do not mention (x).

0 0 1(x) 1 0 1 1

Answer: 0 1 1 1 1 1 ✗

The correct answer is: 0 0 0 0 0 1 1

Question **16**

Correct

Mark 1.00 out of 1.00

Select all the correct statements about MMU and it's functionality (on a non-demand paged system)

Select one or more:

- [x] a. MMU is inside the processor ✓
- [x] b. Illegal memory access is detected in hardware by MMU and a trap is raised ✓
- [ ] c. Logical to physical address translations in MMU are done with specific machine instructions
- [ ] d. MMU is a separate chip outside the processor
- [x] e. Logical to physical address translations in MMU are done in hardware, automatically ✓
- [x] f. The Operating system sets up relevant CPU registers to enable proper MMU translations ✓
- [ ] g. Illegal memory access is detected by operating system
- [ ] h. The operating system interacts with MMU for every single address translation

Your answer is correct.

The correct answers are: MMU is inside the processor, Logical to physical address translations in MMU are done in hardware, automatically, The Operating system sets up relevant CPU registers to enable proper MMU translations, Illegal memory access is detected in hardware by MMU and a trap is raised

Question **17**

Partially correct

Mark 0.88 out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

Your answer is partially correct.

You have correctly selected 7.
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 **18**

Partially correct

Mark 0.50 out of 1.00

Consider a demand-paging system with the following time-measured utilizations:
CPU utilization : 20%
Paging disk: 97.7%
Other I/O devices: 5%

For each of the following, indicate whether it will (or is likely to) improve CPU utilization (even if by a small amount). Explain your answers.

a. Install a faster CPU : Yes ✗

b. Install a bigger paging disk. : No ✓

c. Increase the degree of multiprogramming. : Yes ✗

d. Decrease the degree of multiprogramming. : No ✗

e. Install more main memory.: Yes ✓

f. Install a faster hard disk or multiple controllers with multiple hard disks. : Yes ✓

g. Add prepaging to the page-fetch algorithms. :
Yes ✓

h. Increase the page size. : No ✗

Question **19**

Correct

Mark 1.00 out of 1.00

The complete range of virtual addresses (after main() in main.c is over), from which the free pages used by kalloc() and kfree() is derived,are:

- a. end, PHYSTOP
- b. end, 4MB
- c. P2V(end), P2V(PHYSTOP)
- d. P2V(end), PHYSTOP
- e. end, P2V(4MB + PHYSTOP)
- f. end, (4MB + PHYSTOP)
- [✓ g. end, P2V(PHYSTOP)] ✓

Your answer is correct.

The correct answer is: end, P2V(PHYSTOP)

Question **20**

Incorrect

Mark 0.00 out of 1.00

Mark whether the given sequence of events is possible or not-possible. Also, select the reason for your answer.

For each sequence it's a not-possible sequence if some important event is not mentioned in the sequence.

Assume that the kernel code is non-interruptible and uniprocessor system.

```
Process P1, user code executing
Timer interrupt
Context changes to kernel context
Generic interrupt handler runs
Generic interrupt handler calls Scheduler
Scheduler selects P2 for execution
After scheduler, Process P2 user code executing
```

This sequence of events is: possible ✗

Because

✗

Question **21**

Correct

Mark 1.00 out of 1.00

Suppose a kernel uses a buddy allocator. The smallest chunk that can be allocated is of size 32 bytes. One bit is used to track each such chunk, where 1 means allocated and 0 means free. The chunk looks like this as of now:

10011010

Now, there is a request for a chunk of 50 bytes.

After this allocation, the bitmap, indicating the status of the buddy allocator will be

Answer: 11111010 ✓

The correct answer is: 11111010

Question **22**

Partially correct

Mark 0.55 out of 1.00

Select all the correct statements about process states.

Note that in this question you lose marks for every incorrect choice that you make, proportional to actual number of incorrect choices.

- [ ] a. A process becomes ZOMBIE when it calls exit()
- [x] b. Process state is stored in the PCB ✓
- [x] c. The scheduler can change state of a process from RUNNALBE to RUNNING ✓
- [ ] d. The scheduler can change state of a process from RUNNALBE to RUNNING and vice-versa
- [ ] e. Process state is stored in the processor
- [ ] f. Process state is changed only by interrupt handlers
- [x] g. Process state can be implemented as just a number ✓
- [ ] h. Process state is implemented as a string
- [x] i. A process becomes ZOMBIE when another process bites into it's memory ✗

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: Process state is stored in the PCB, Process state can be implemented as just a number, The scheduler can change state of a process from RUNNALBE to RUNNING, A process becomes ZOMBIE when it calls exit()

Question **23**

Partially correct

Mark 1.56 out of 2.00

Match the description of a memory management function with the name of the function that provides it, in xv6

Load contents from ELF into pages after allocating the pages first	loaduvm()	✖
Create a copy of the page table of a process	copyuvm()	✓
Switch to kernel page table	switchkvm()	✓
setup the kernel part in the page table	setupkvm()	✓
Setup and load the user page table for initcode process	inituvm()	✓
Mark the page as in-accessible	clearpteu()	✓
Switch to user page table	switchuvm()	✓
Load contents from ELF into existing pages	No such function	✖
Copy the code pages of a process	No such function	✓

The correct answer is: Load contents from ELF into pages after allocating the pages first → No such function, Create a copy of the page table of a process → copyuvm(), Switch to kernel page table → switchkvm(), setup the kernel part in the page table → setupkvm(), Setup and load the user page table for initcode process → inituvm(), Mark the page as in-accessible → clearpteu(), Switch to user page table → switchuvm(), Load contents from ELF into existing pages → loaduvm(), Copy the code pages of a process → No such function

Question **24**

Partially correct

Mark 0.83 out of 1.00

W.r.t. xv6 code, match the state of a process with a code that sets the state

SLEEPING	sleep(), called by any process blocking itself	✓
EMBRYO	fork()->allocproc() before setting up the UVM	✓
UNUSED	wait(), called by parent process	✓
RUNNABLE	wakeup(), called by an interrupt handler	✓
RUNNING	scheduler()	✓
ZOMBIE	exit(), called by an interrupt handler	✖

The correct answer is: SLEEPING → sleep(), called by any process blocking itself, EMBRYO → fork()->allocproc() before setting up the UVM, UNUSED → wait(), called by parent process, RUNNABLE → wakeup(), called by an interrupt handler, RUNNING → scheduler(), ZOMBIE → exit(), called by process itself

Question **25**

Partially correct

Mark 0.30 out of 2.00

Order the following events, in the creation of init() process in xv6:

1. ✗ userinit() is called
2. ✓ kernel stack is allocated for initcode process
3. ✗ trapframe and context pointers are set to proper location
4. ✗ empty struct proc is obtained for initcode
5. ✗ Arguments on setup on process stack for /init
6. ✓ the header of "/init" ELF file is ready by kernel
7. ✗ memory mappings are created for "/init" process
8. ✗ trap() runs
9. ✗ initcode process runs
10. ✗ sys_exec runs
11. ✗ initcode calls exec system call
12. ✗ Stack is allocated for "/init" process
13. ✓ kernel memory mappings are created for initcode
14. ✗ values are set in the trapframe of initcode
15. ✗ page table mappings of 'initcode' are replaced by makpings of 'init'
16. ✗ initcode is selected by scheduler for execution
17. ✗ name of process "/init" is copied in struct proc
18. ✗ function pointer from syscalls[] array is invoked
19. ✗ initcode process is set to be runnable
20. ✗ code is set to start in forkret() when process gets scheduled

Your answer is partially correct.

Grading type: Relative to the next item (including last)

Grade details: 3 / 20 = 15%

Here are the scores for each item in this response:

1. 0 / 1 = 0%
2. 1 / 1 = 100%
3. 0 / 1 = 0%
4. 0 / 1 = 0%
5. 0 / 1 = 0%
6. 1 / 1 = 100%
7. 0 / 1 = 0%
8. 0 / 1 = 0%
9. 0 / 1 = 0%
10. 0 / 1 = 0%
11. 0 / 1 = 0%
12. 0 / 1 = 0%
13. 1 / 1 = 100%

14. 0 / 1 = 0%
15. 0 / 1 = 0%
16. 0 / 1 = 0%
17. 0 / 1 = 0%
18. 0 / 1 = 0%
19. 0 / 1 = 0%
20. 0 / 1 = 0%

The correct order for these items is as follows:

1. userinit() is called
2. empty struct proc is obtained for initcode
3. kernel stack is allocated for initcode process
4. trapframe and context pointers are set to proper location
5. code is set to start in forkret() when process gets scheduled
6. kernel memory mappings are created for initcode
7. values are set in the trapframe of initcode
8. initcode process is set to be runnable
9. initcode is selected by scheduler for execution
10. initcode process runs
11. initcode calls exec system call
12. trap() runs
13. function pointer from syscalls[] array is invoked
14. sys_exec runs
15. the header of "/init" ELF file is ready by kernel
16. memory mappings are created for "/init" process
17. Stack is allocated for "/init" process
18. Arguments on setup on process stack for /init
19. name of process "/init" is copied in struct proc
20. page table mappings of 'initcode' are replaced by makpings of 'init'

Question **26**

Partially correct

Mark 0.60 out of 1.00

Select all the correct statements w.r.t user and kernel threads

Select one or more:

- [✓] a. many-one model can be implemented even if there are no kernel threads ✓
- [✓] b. A process blocks in many-one model even if a single thread makes a blocking system call ✓
- [ ] c. all three models, that is many-one, one-one, many-many , require a user level thread library
- [✓] d. many-one model gives no speedup on multicore processors ✓
- [ ] e. A process may not block in many-one model, if a thread makes a blocking system call
- [ ] f. one-one model increases kernel's scheduling load
- [ ] g. one-one model can be implemented even if there are no kernel threads

Your answer is partially correct.

You have correctly selected 3.
The correct answers are: many-one model can be implemented even if there are no kernel threads, all three models, that is many-one, one-one, many-many , require a user level thread library, one-one model increases kernel's scheduling load, many-one model gives no speedup on multicore processors, A process blocks in many-one model even if a single thread makes a blocking system call

Question **27**

Incorrect

Mark 0.00 out of 1.00

If one thread opens a file with read privileges then

Select one:

- ● a. any other thread cannot read from that file ✗
- ○ b. other threads in the another process can also read from that file
- ○ c. none of these
- ○ d. other threads in the same process can also read from that file

Your answer is incorrect.

The correct answer is: other threads in the same process can also read from that file

Question **28**

Partially correct

Mark 0.25 out of 1.00

Select all the correct statements about signals

Select one or more:

- ☐ a. The signal handler code runs in kernel mode of CPU
- ☑ b. SIGKILL definitely kills a process because it's code runs in kernel mode of CPU ✗
- ☑ c. Signals are delivered to a process by kernel ✓
- ☐ d. SIGKILL definitely kills a process because it can't be caught or ignored, and it's default action terminates the process
- ☐ e. Signal handlers once replaced can't be restored
- ☑ f. A signal handler can be invoked asynchronously or synchronously depending on signal type ✓
- ☑ g. Signals are delivered to a process by another process ✗
- ☑ h. The signal handler code runs in user mode of CPU ✓

Your answer is partially correct.

You have selected too many options.
The correct answers are: Signals are delivered to a process by kernel, A signal handler can be invoked asynchronously or synchronously depending on signal type, The signal handler code runs in user mode of CPU, SIGKILL definitely kills a process because it can't be caught or ignored, and it's default action terminates the process

Question **29**

Partially correct

Mark 0.38 out of 1.00

Mark the statements as True or False, w.r.t. passing of arguments to system calls in xv6 code.

True	False		
◉ ✗	○ ✓	Integer arguments are stored in eax, ebx, ecx, etc. registers	✗
◉ ✗	○ ✓	String arguments are first copied to trapframe and then from trapframe to kernel's other variables.	✗
◉ ✓	○ ✗	Integer arguments are copied from user memory to kernel memory using argint()	✓
◉ ✓	○ ✗	The arguments are accessed in the kernel code using esp on the trapframe.	✓
○ ✓	◉ ✗	The arguments to system call originally reside on process stack.	✗
○ ✓	◉ ✗	String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer	✗
◉ ✓	○ ✗	The functions like argint(), argstr() make the system call arguments available in the kernel.	✓
◉ ✗	○ ✓	The arguments to system call are copied to kernel stack in trapasm.S	✗

Integer arguments are stored in eax, ebx, ecx, etc. registers: False
String arguments are first copied to trapframe and then from trapframe to kernel's other variables.: False
Integer arguments are copied from user memory to kernel memory using argint(): True
The arguments are accessed in the kernel code using esp on the trapframe.: True
The arguments to system call originally reside on process stack.: True
String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer: True
The functions like argint(), argstr() make the system call arguments available in the kernel.: True
The arguments to system call are copied to kernel stack in trapasm.S: False

Question **30**

Correct

Mark 1.00 out of 1.00

The data structure used in kalloc() and kfree() in xv6 is

- [✓ a. Singly linked NULL terminated list] ✓
- b. Double linked NULL terminated list
- c. Doubly linked circular list
- d. Singly linked circular list

Your answer is correct.

The correct answer is: Singly linked NULL terminated list

Question **31**

Correct

Mark 1.00 out of 1.00

Consider a computer system with a 32-bit logical address and 4- KB page size. The system supports up to 512 MB of physical memory. How many entries are there in each of the following?

Write answer as a decimal number.

A conventional, single-level page table:

1048576

✓

An inverted page table:

131072

✓

Question **32**

Incorrect

Mark 0.00 out of 2.00

For the reference string
3 4 3 5 2

using FIFO replacement policy for pages,

consider the number of page faults for 2, 3 and 4 page frames.
Select the correct statement.

Select one:

○ a. Do not exhibit Balady's anomaly

[✓ b. Exhibit Balady's anomaly between 3 and 4 frames] ✖

○ c. Exhibit Balady's anomaly between 2 and 3 frames

Your answer is incorrect.

The correct answer is: Do not exhibit Balady's anomaly

Question **33**

Partially correct

Mark 0.86 out of 1.00

After virtual memory is implemented

(select T/F for each of the following)One Program's size can be larger than physical memory size

True	False		
◉ ✓	○ ✗	One Program's size can be larger than physical memory size	✓
◉ ✓	○ ✗	Cumulative size of all programs can be larger than physical memory size	✓
○ ✗	◉ ✓	Virtual access to memory is granted to all processes	✓
◉ ✓	○ ✗	Relatively less I/O may be possible during process execution	✓
◉ ✗	○ ✓	Virtual addresses become available to executing process	✗
◉ ✓	○ ✗	Logical address space could be larger than physical address space	✓
◉ ✓	○ ✗	Code need not be completely in memory	✓

One Program's size can be larger than physical memory size: True
Cumulative size of all programs can be larger than physical memory size: True
Virtual access to memory is granted to all processes: False
Relatively less I/O may be possible during process execution: True
Virtual addresses become available to executing process: False
Logical address space could be larger than physical address space: True
Code need not be completely in memory: True

Question **34**

Correct

Mark 2.00 out of 2.00

Consider the reference string
6 4 2 0 1 2 6 9 2 0 5

If the number of page frames is 3, then total number of page faults (including initial), using FIFO replacement is:

Answer: 10 ✓

#6# 6,4# 6,4,2 #0,4,2# 0,1,2 #0,1,6 #9,1,6# 9,2,6# 9,2,0 #5,2,0

The correct answer is: 10

Question **35**

Partially correct

Mark 0.50 out of 1.00

For each function/code-point, select the status of segmentation setup in xv6

bootmain()	gdt setup with 3 entries, right from first line of code of bootloader	✖
after seginit() in main()	gdt setup with 5 entries (0 to 4) on one processor	✓
entry.S	gdt setup with 3 entries, right from first line of code of bootloader	✖
bootasm.S	gdt setup with 3 entries, at start32 symbol of bootasm.S	✓
kvmalloc() in main()	gdt setup with 3 entries, right from first line of code of bootloader	✖
after startothers() in main()	gdt setup with 5 entries (0 to 4) on all processors	✓

Your answer is partially correct.

You have correctly selected 3.
The correct answer is: bootmain() → gdt setup with 3 entries, at start32 symbol of bootasm.S, after seginit() in main() → gdt setup with 5 entries (0 to 4) on one processor, entry.S → gdt setup with 3 entries, at start32 symbol of bootasm.S, bootasm.S → gdt setup with 3 entries, at start32 symbol of bootasm.S, kvmalloc() in main() → gdt setup with 3 entries, at start32 symbol of bootasm.S, after startothers() in main() → gdt setup with 5 entries (0 to 4) on all processors

◄ (Optional Assignment) lseek system call in xv6

Jump to...

Feedback on Quiz-2 ►