Started on	Sunday, 18 February 2024, 2:00 PM
State	Finished
Completed on	Sunday, 18 February 2024, 3:56 PM
Time taken	1 hour 55 mins
Grade	**13.45** out of 15.00 (**89.69**%)

Question **1**

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

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

Question **2**

Partially correct

Mark 0.40 out of 0.50

Match the elements of C program to their place in memory

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

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

Question **3**

Correct

Mark 1.00 out of 1.00

Consider the image given below, which explains how paging works.

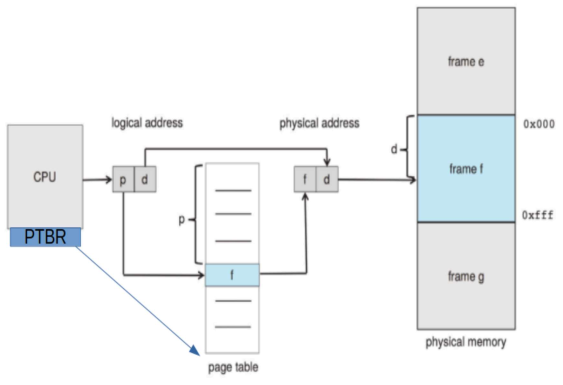

Figure 9.8 Paging hardware.

Mention whether each statement is True or False, with respect to this image.

True	False		
○ ✗	[● ✓]	The page table is indexed using frame number	✓
○ ✗	[● ✓]	Size of page table is always determined by the size of RAM	✓
[● ✓]	○ ✗	The PTBR is present in the CPU as a register	✓
○ ✗	[● ✓]	The locating of the page table using PTBR also involves paging translation	✓
[● ✓]	○ ✗	The page table is itself present in Physical memory	✓
[● ✓]	○ ✗	The page table is indexed using page number	✓
[● ✓]	○ ✗	The physical address may not be of the same size (in bits) as the logical address	✓
[● ✓]	○ ✗	Maximum Size of page table is determined by number of bits used for page number	✓

The page table is indexed using frame number: False
Size of page table is always determined by the size of RAM: False
The PTBR is present in the CPU as a register: True
The locating of the page table using PTBR also involves paging translation: False
The page table is itself present in Physical memory: True
The page table is indexed using page number: True
The physical address may not be of the same size (in bits) as the logical address: True
Maximum Size of page table is determined by number of bits used for page number: True

Question **4**

Partially correct

Mark 0.75 out of 1.00

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

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

Your answer is partially correct.

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

Question **5**

Correct

Mark 0.50 out of 0.50

What's the trapframe in xv6?

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

Your answer is correct.

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

Question **6**

Correct

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 **7**

Incorrect

Mark 0.00 out of 0.50

The ljmp instruction in general does

[✓ a.] change the CS and EIP to 32 bit mode, and jumps to next line of code ✖

b. change the CS and EIP to 32 bit mode

c. change the CS and EIP to 32 bit mode, and jumps to new value of EIP

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

Question **8**

Correct

Mark 1.00 out of 1.00

Mark the statements as True/False w.r.t. the basic concepts of memory management.

True	False		
○ ✗	● ✓	The kernel refers to the page table for converting each virtual address to physical address.	✓
● ✓	○ ✗	The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.	✓
○ ✗	● ✓	The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.	✓
○ ✗	● ✓	When a process is executing, each virtual address is converted into physical address by the kernel directly.	✓
● ✓	○ ✗	The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.	✓
○ ✗	● ✓	The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.	✓
● ✓	○ ✗	When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.	✓

The kernel refers to the page table for converting each virtual address to physical address.: False
The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True
The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.: False
When a process is executing, each virtual address is converted into physical address by the kernel directly.: False
The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.: True
The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.: False
When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.: True

Question **9**

Correct

Mark 1.00 out of 1.00

Suppose a processor supports base(relocation register) + limit scheme of MMU.

Assuming this, mark the statements as True/False

True	False		
● ✓	○ ✗	The OS sets up the relocation and limit registers when the process is scheduled	✓
● ✓	○ ✗	The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;	✓
○ ✗	● ✓	The OS detects any memory access beyond the limit value and raises an interrupt	✓
○ ✗	● ✓	The compiler generates machine code assuming appropriately sized semgments for code, data and stack.	✓
○ ✗	● ✓	The process sets up it's own relocation and limit registers when the process is scheduled	✓
● ✓	○ ✗	The hardware detects any memory access beyond the limit value and raises an interrupt	✓
● ✓	○ ✗	The OS may terminate the process while handling the interrupt of memory violation	✓
○ ✗	● ✓	The hardware may terminate the process while handling the interrupt of memory violation	✓

The OS sets up the relocation and limit registers when the process is scheduled: True
The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;: True
The OS detects any memory access beyond the limit value and raises an interrupt: False
The compiler generates machine code assuming appropriately sized semgments for code, data and stack.: False
The process sets up it's own relocation and limit registers when the process is scheduled: False
The hardware detects any memory access beyond the limit value and raises an interrupt: True
The OS may terminate the process while handling the interrupt of memory violation: True
The hardware may terminate the process while handling the interrupt of memory violation: False

Question **10**

Correct

Mark 0.50 out of 0.50

Match the register pairs

IP	CS	✓
BP	SS	✓
DI	DS	✓
SI	DS	✓
SP	SS	✓

The correct answer is: IP → CS, BP → SS, DI → DS, SI → DS, SP → SS

Question **11**

Correct

Mark 0.50 out of 0.50

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

- ☐ 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; but calling convention was applicable all the time
- [✓] c. The setting up of the most essential memory management infrastructure needs assembly code ✓
- [✓] d. 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 ✓

Your answer is correct.

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

Question **12**

Correct

Mark 0.50 out of 0.50

A process blocks itself means

- a. The kernel code of an interrupt handler, moves the process to a waiting queue and calls scheduler
- b. The application code calls the scheduler
- [✓ c. The kernel code of system call, called by the process, moves the process to a waiting queue and calls 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 **13**

Correct

Mark 0.50 out of 0.50

The kernel is loaded at Physical Address

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

The correct answer is: 0x00100000

Question **14**

Correct

Mark 0.50 out of 0.50

```
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 **15**

Incorrect

Mark 0.00 out of 0.50

The variable $stack in entry.S is

- ○ a. a memory region allocated as a part of entry.S
- ● b. located at less than 0x7c00 ✗
- ○ c. located at 0x7c00
- ○ d. located at 0
- ○ 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 **16**

Correct

Mark 0.50 out of 0.50

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

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

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

Question **17**

Partially correct

Mark 0.30 out of 0.50

xv6.img: bootblock kernel

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

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

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

Your answer is partially correct.

You have correctly selected 3.
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 **18**

Correct

Mark 0.50 out of 0.50

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

Your answer is correct.

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

Question **19**

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:

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

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 **20**

Correct

Mark 0.50 out of 0.50

The trapframe, in xv6, is built by the

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

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

Question **21**

Correct

Mark 0.50 out of 0.50

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

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

The correct answer is: 801154a8

Question **22**

Correct

Mark 0.50 out of 0.50

Select all the correct statements about zombie processes

Select one or more:

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

Your answer is correct.

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 **23**

Correct

Mark 0.50 out of 0.50

Which of the following state transitions are not possible?

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

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

Question **24**

Correct

Mark 0.50 out of 0.50

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

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

The correct answer is: 3

◄ Homework questions: Basics of MM, xv6 booting

Jump to...

Quiz-2 (15 Marks) ►