Started on	Saturday, 16 March 2024, 1:32 PM
State	Finished
Completed on	Saturday, 16 March 2024, 3:33 PM
Time taken	2 hours 1 min
Grade	**13.35** out of 15.00 (**88.97**%)

Question **1**

Correct

Mark 0.50 out of 0.50

Select the correct statements about sched() and scheduler() in xv6 code

☑ a. Each call to sched() or scheduler() involves change of one stack inside swtch()✓

☑ b. When either sched() or scheduler() is called, it does not return immediately to caller✓

☑ c. sched() switches to the scheduler's context✓

☑ d. sched() and scheduler() are co-routines✓

☑ e. When either sched() or scheduler() is called, it results in a context switch✓

☑ f. After call to swtch() in scheduler(), the control moves to code in sched()✓

☑ g. scheduler() switches to the selected process's context✓

☑ h. After call to swtch() in sched(), the control moves to code in scheduler()✓

Your answer is correct.

The correct answers are: sched() and scheduler() are co-routines, When either sched() or scheduler() is called, it does not return immediately to caller, When either sched() or scheduler() is called, it results in a context switch, sched() switches to the scheduler's context, scheduler() switches to the selected process's context, After call to swtch() in scheduler(), the control moves to code in sched(), After call to swtch() in sched(), the control moves to code in scheduler(), Each call to sched() or scheduler() involves change of one stack inside swtch()

Question **2**

Correct

Mark 0.50 out of 0.50

The variable $stack in entry.S is

○ a. located at 0x7c00

○ b. located at the value given by %esp as setup by bootmain()

● c. a memory region allocated as a part of entry.S✓

○ d. located at less than 0x7c00

○ e. located at 0

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

Question **3**

Correct

Mark 0.50 out of 0.50

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

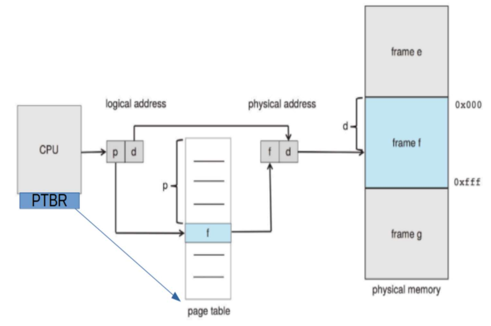


Figure 9.8 Paging hardware.

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

True	False		
○ ✗	● ☑	Size of page table is always determined by the size of RAM	✓
● ☑	○ ✗	The PTBR is present in the CPU as a register	✓
● ☑	○ ✗	Maximum Size of page table is determined by number of bits used for page number	✓
○ ✗	● ☑	The page table is indexed using frame number	✓
● ☑	○ ✗	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	✓
○ ✗	● ☑	The locating of the page table using PTBR also involves paging translation	✓

Size of page table is always determined by the size of RAM: False
The PTBR is present in the CPU as a register: True
Maximum Size of page table is determined by number of bits used for page number: True
The page table is indexed using frame number: 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
The locating of the page table using PTBR also involves paging translation: False

Question **4**

Correct

Mark 0.50 out of 0.50

Mark statements as True/False w.r.t. the creation of free page list in xv6.

True	False			
● ✓	○ ✗	the kmem.lock is used by kfree() and kalloc() only.	✓	
○ ✗	● ✓	free page list is a singly circular linked list.	✓	it's singly linked NULL terminated list.

● ✓	○ ✗	if(kmem.use_lock)
		acquire(&kmem.lock);
		is not done when called from kinit1() because there is no need to take the lock when kinit1() is running because interrupts are disabled and only one processor is running
			✓	

● ✓	○ ✗	kmem.use_lock is set to 1 after free page list is created, so that kmem.lock is taken before accessing kmem.freelist.	✓	

○ ✗	● ✓	if(kmem.use_lock)
		acquire(&kmem.lock);
		this "if" condition is true, when kinit2() runs because multi-processor support has been enabled by now.
			✓	No. kinit2() calls kfree() and then initializes use_lock.

● ✓	○ ✗	The pointers that link the pages together are in the first 4 bytes of the pages themselves	✓	

the kmem.lock is used by kfree() and kalloc() only.: True
free page list is a singly circular linked list.: False
if(kmem.use_lock)
    acquire(&kmem.lock);
is not done when called from kinit1() because there is no need to take the lock when kinit1() is running because interrupts are disabled and only one processor is running: True
kmem.use_lock is set to 1 after free page list is created, so that kmem.lock is taken before accessing kmem.freelist.: True
if(kmem.use_lock)
    acquire(&kmem.lock);
this "if" condition is true, when kinit2() runs because multi-processor support has been enabled by now.: False
The pointers that link the pages together are in the first 4 bytes of the pages themselves: True

Question **5**

Correct

Mark 0.50 out of 0.50

Map ext2 data structure features with their purpose

Mount count in superblock ✓	to enforce file check after certain amount of mounts at boot time
A group ✓	Try to keep all the data of a directory and it's file close together in a group
Combining file type and access rights in one variable ✓	saves 1 byte of space
Block bitmap is one block ✓	Limits the size of a block group, thus improvising on purpose of a group
Inode bitmap is one block ✓	limits total number of files that can belong to a group
File Name is padded ✓	aligns all memory accesses on word boundary, improving performance
rec_len field in directory entry ✓	allows holes and linking of entries in directory
Inode table location in Group Descriptor ✓	Obvious, as it's per group and not per file-system
Inode table ✓	All inodes are kept together so that one disk read leads to reading many inodes together, effectively doing a buffering of subsequent inode reads, and to save
Used directories count in group descriptor ✓	attempt is made to evenly spread the first-level directories, this count is used there
Free blocks count in superblock and group descriptor ✓	Redundancy to help fsck restore consistency
Many copies of Superblock ✓	Redundancy to ensure the most crucial data structure is not lost

Your answer is correct.

The correct answer is: Mount count in superblock → to enforce file check after certain amount of mounts at boot time, A group → Try to keep all the data of a directory and it's file close together in a group, Combining file type and access rights in one variable → saves 1 byte of space, Block bitmap is one block → Limits the size of a block group, thus improvising on purpose of a group, Inode bitmap is one block → limits total number of files that can belong to a group, File Name is padded → aligns all memory accesses on word boundary, improving performance, rec_len field in directory entry → allows holes and linking of entries in directory, Inode table location in Group Descriptor → Obvious, as it's per group and not per file-system, Inode table → All inodes are kept together so that one disk read leads to reading many inodes together, effectively doing a buffering of subsequent inode reads, and to save space on disk, Used directories count in group descriptor →

attempt is made to evenly spread the first-level directories, this count is used there, Free blocks count in superblock and group descriptor → Redundancy to help fsck restore consistency, Many copies of Superblock → Redundancy to ensure the most crucial data structure is not lost

Question **6**

Partially correct

Mark 0.43 out of 0.50

Given below are statements about concurrency and parallelism

Select T/F

A concurrent system can allow more than one task to progress, whereas a parallel system can perform more than one task at the same time.

True	**False**		
○ ✗	● ✓	Parallel systems allow more than one task to progress while concurrent systems do not.	✓
● ✓	○ ✗	It is possible to have concurrency without parallelism	✓
● ✓	○ ✗	A concurrent system can allow more than one task to progress, whereas a parallel system can perform more than one task at the same time.	✓
● ✗	○ ✓	A concurrent system allows more than one task to progress while a parallel system does not.	✗
○ ✗	● ✓	Both concurrency and parallelism are the same.	✓
○ ✗	● ✓	It is possible to have parallelism without concurrency	✓
○ ✗	● ✓	It is not possible to have concurrency without parallelism.	✓

Parallel systems allow more than one task to progress while concurrent systems do not.: False
It is possible to have concurrency without parallelism: True
A concurrent system can allow more than one task to progress, whereas a parallel system can perform more than one task at the same time.: True
A concurrent system allows more than one task to progress while a parallel system does not.: False
Both concurrency and parallelism are the same.: False
It is possible to have parallelism without concurrency: False
It is not possible to have concurrency without parallelism.: False

Question **7**

Correct

Mark 0.50 out of 0.50

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

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

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

Question **8**

Correct

Mark 0.50 out of 0.50

Which of the following is DONE by allocproc() ?

☐ a. ensure that the process starts in trapret()

☑ b. ensure that the process starts in forkret() ✓

☑ c. allocate PID to the process ✓

☑ d. Select an UNUSED struct proc for use ✓

☐ e. setup the contents of the trapframe of the process properly

☑ f. allocate kernel stack for the process ✓

☑ g. setup the trapframe and context pointers appropriately ✓

☐ h. setup kernel memory mappings for 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 **9**

Correct

Mark 0.50 out of 0.50

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

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

Question **10**

Incorrect

Mark 0.00 out of 0.50

In the code below assume that each function can be executed concurrently by many threads/processes.
Ignore syntactical issues, and focus on the semantics.

This program is an example of

```
spinlock a, b;  // assume initialized
thread1() {
    spinlock(a);
    //some code;
    spinlock(b);
    //some code;
    spinunlock(b);
    spinunlock(a);
}
thread2() {
    spinlock(a);
    //some code;
    spinlock(b);
    //some code;
    spinunlock(b);
    spinunlock(a);
}
```

○ a. Deadlock or livelock depending on actual race

○ b. Self Deadlock

○ c. Livelock

● d. Deadlock ✗

○ e. None of these

Your answer is incorrect.

The correct answer is: None of these

Question **11**

Correct

Mark 0.50 out of 0.50

Select the correct statements about paging (not demand paging) mechanism

Select one or more:

□ a. Page table is accessed by the OS as part of execuation of an instruction

☑ b. OS creates the page table for every process✓

□ c. User process can update it's own PTBR

☑ d. Page table is accessed by the MMU as part of execution of an instruction✓

☑ e. An invalid entry on a page means, it was an illegal memory reference✓

□ f. An invalid entry on a page means, either it was illegal memory reference or the page was not present in memory.

☑ g. The PTBR is loaded by the OS✓

□ h. User process can update it's own page table entries

Your answer is correct.

The correct answers are: OS creates the page table for every process, The PTBR is loaded by the OS, Page table is accessed by the MMU as part of execution of an instruction, An invalid entry on a page means, it was an illegal memory reference

Question **12**

Correct

Mark 0.50 out of 0.50

Why V2P_WO is used in entry.S and not V2P ?

○ a. It's a mistake. They could have used the same macro in both places.

○ b. The two macros are different. They lead to different calculations.

○ c. Because the processor can not do a type casting at run time

● d. Because entry.S is an assembly code file and assemblers do not know about data types and type casting.✓

○ e. The typecasting has the effect of creating virtual address, while without typecast we get physical address.

Your answer is correct.

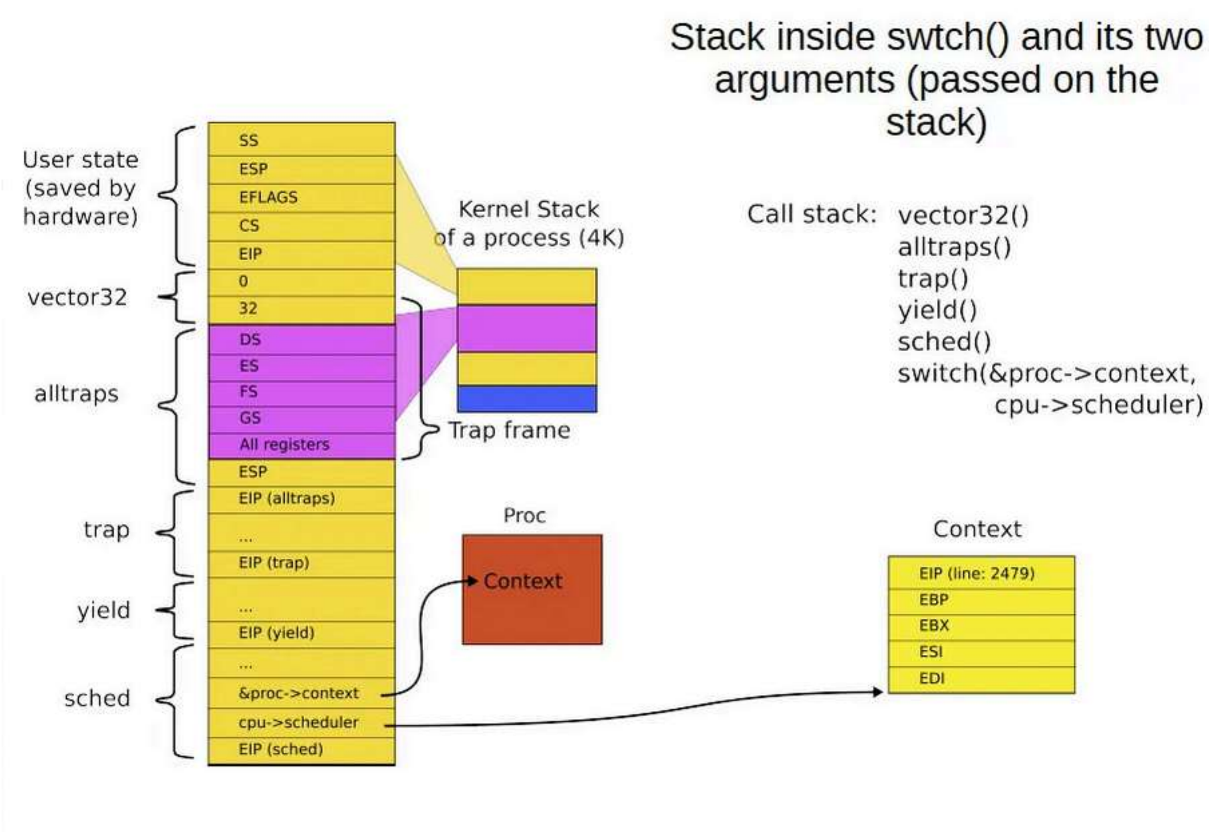
The correct answer is: Because entry.S is an assembly code file and assemblers do not know about data types and type casting.

Question **13**

Correct

Mark 0.50 out of 0.50

Mark statements as True/False, w.r.t. the given diagram



Stack inside swtch() and its two arguments (passed on the stack)

Call stack: vector32()
alltraps()
trap()
yield()
sched()
switch(&proc->context,
cpu->scheduler)

True	False			
○ ✗	● ✓	The diagram is wrong because it shows the user stack and kernel stack together (continuous), but in practice they are separate	✓	diagram shows only kernel stack
● ✓	○ ✗	The diagram is correct	✓	
○ ✗	● ✓	The "context" yellow coloured box, pointed to by cpu->scheduler is on the kernel stack of the scheduler.	✓	
● ✓	○ ✗	The blue shaded part in "kernel stack of a process(4k)" refers to remaining part of stack (not used yet)	✓	
○ ✗	● ✓	This is a diagram of swtch() called from scheduler()	✓	No. diagram of swtch() called from sched()
● ✓	○ ✗	The "ESP" (second entry from top) is stack pointer of user-stack of process, while the "ESP" (first entry below pink region) is the trapframe pointer on kernel stack of process.	✓	

The diagram is wrong because it shows the user stack and kernel stack together (continuous), but in practice they are separate: False
The diagram is correct: True
The "context" yellow coloured box, pointed to by cpu->scheduler is on the kernel stack of the scheduler.: False
The blue shaded part in "kernel stack of a process(4k)" refers to remaining part of stack (not used yet): True
This is a diagram of swtch() called from scheduler(): False
The "ESP" (second entry from top) is stack pointer of user-stack of process, while the "ESP" (first entry below pink region) is the trapframe pointer on kernel stack of process.: True

Question **14**

Correct

Mark 0.50 out of 0.50

Mark the statements as True/False, with respect to the use of the variable "chan" in struct proc.

True	**False**		
○ ✗	● ☑	chan is the head pointer to a linked list of processes, waiting for a particular event to occur	✓
● ☑	○ ✗	When chan is not NULL, the 'state' in struct proc must be SLEPING	✓
● ☑	○ ✗	chan stores the address of the variable, representing a condition, for which the process is waiting.	✓
● ☑	○ ✗	The value of 'chan' is changed only in sleep()	✓
● ☑	○ ✗	in xv6, the address of an appropriate variable is used as a "condition" for a waiting process.	✓
○ ✗	● ☑	Changing the state of a process automatically changes the value of 'chan'	✓
● ☑	○ ✗	'chan' is used only by the sleep() and wakeup1() functions.	✓
○ ✗	● ☑	when chan is NULL, the 'state' in proc must be RUNNABLE.	✓

chan is the head pointer to a linked list of processes, waiting for a particular event to occur: False
When chan is not NULL, the 'state' in struct proc must be SLEPING: True
chan stores the address of the variable, representing a condition, for which the process is waiting.: True
The value of 'chan' is changed only in sleep(): True
in xv6, the address of an appropriate variable is used as a "condition" for a waiting process.: True
Changing the state of a process automatically changes the value of 'chan': False
'chan' is used only by the sleep() and wakeup1() functions.: True
when chan is NULL, the 'state' in proc must be RUNNABLE.: False

Question **15**

Correct

Mark 0.50 out of 0.50

Doing a lookup on the pathname /a/b/b/c/d for opening the file "d" requires reading [6] ✓ no. of inodes. Assume that there are no hard/soft links on the path.

Write the answer as a number.

The correct answer is: 6

Question **16**

Partially correct

Mark 0.25 out of 0.50

Suppose a file is to be created in an ext2 file system, in an existing directory /a/b/. Select from below, the list of blocks that may need modification.

Select one or more:

☑ a. inode of /a/b/✓

☐ b. inode of /a/

☐ c. link count on /a/b/ inode

☐ d. inode bitmap in some block group

☑ e. group descriptor(s)✓

☐ f. inode bitmap referrring to /a/b/

☐ g. new data block in some block group

☑ h. superblock✓

☐ i. data blocks of /a/

☐ j. inode table in some block group

☑ k. block bitmap in some block group✓

☐ l. existing data blocks of /a/b/

Your answer is partially correct.

You have correctly selected 4.
The correct answers are: superblock, group descriptor(s), inode of /a/b/, existing data blocks of /a/b/, inode table in some block group, inode bitmap in some block group, block bitmap in some block group, new data block in some block group

Question **17**

Partially correct

Mark 0.33 out of 0.50

Select the correct statements about hard and soft links

Select one or more:

□ a. Soft link shares the inode of actual file

□ b. Deleting a soft link deletes only the actual file

□ c. Deleting a soft link deletes the link, not the actual file

☑ d. Deleting a soft link deletes both the link and the actual file ✗

☑ e. Hard links share the inode ✓

☑ f. Hard links enforce separation of filename from it's metadata in on-disk data structures. ✓

☑ g. Soft links can span across partitions while hard links can't ✓

□ h. Hard links can span across partitions while soft links can't

☑ i. Deleting a hard link deletes the file, only if link count was 1 ✓

□ j. Soft links increase the link count of the actual file inode

☑ k. Hard links increase the link count of the actual file inode ✓

□ l. Deleting a hard link always deletes the file

Your answer is partially correct.

You have correctly selected 5.
The correct answers are: Soft links can span across partitions while hard links can't, Hard links increase the link count of the actual file inode, Deleting a soft link deletes the link, not the actual file, Deleting a hard link deletes the file, only if link count was 1, Hard links share the inode, Hard links enforce separation of filename from it's metadata in on-disk data structures.

Question **18**

Correct

Mark 0.50 out of 0.50

It is proposed that when a process does an illegal memory access, xv6 terminate the process by printing the error message "Illegal Memory Access". Select all the changes that need to be done to xv6 for this as True (Note that the changes proposed here may not cover the exhaustive list of all changes required) and the un-necessary/wrong changes as False.

Required	**Un-necessary/Wrong**		
● ☑	○ ☒	Change in the Makefile and instruct cc/ld to start the code of each program at some address other than 0	✓
● ☑	○ ☒	Change allocuvm() to call mappages() with proper permissions on each page table entry	✓
○ ☒	● ☑	Change mappages() to set specified permissions on each page table entry	✓
○ ☒	● ☑	Mark each page as readonly in the page table mappings	✓
● ☑	○ ☒	Ensure that the address 0 is mapped to invalid	✓
● ☑	○ ☒	Handle the Illegal memory acceess trap in trap() function, and terminate the currently running process.	✓
● ☑	○ ☒	Change exec to treat text/data sections separately and call allocuvm() with proper flags for page table entries	✓
○ ☒	● ☑	Add code that checks if the illegal memory access trap was due to an actual illegal memory access.	✓

Change in the Makefile and instruct cc/ld to start the code of each program at some address other than 0: Required
Change allocuvm() to call mappages() with proper permissions on each page table entry: Required
Change mappages() to set specified permissions on each page table entry: Un-necessary/Wrong
Mark each page as readonly in the page table mappings: Un-necessary/Wrong
Ensure that the address 0 is mapped to invalid: Required
Handle the Illegal memory acceess trap in trap() function, and terminate the currently running process.: Required
Change exec to treat text/data sections separately and call allocuvm() with proper flags for page table entries: Required
Add code that checks if the illegal memory access trap was due to an actual illegal memory access.: Un-necessary/Wrong

Question **19**

Correct

Mark 0.50 out of 0.50

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

○ a. 8010a48c
○ b. 80110000
○ c. 80000000
● d. 801154a8 ✓
○ e. 80102da0
○ f. 81000000

The correct answer is: 801154a8

Question **20**

Partially correct

Mark 0.33 out of 0.50

Mark statements about deadlocks as True or false

True	**False**		
○ ✗	● ☑	Deadlocks are the same as livelocks	✓
● ☑	○ ✗	A deadlock is possible only if all the 4 conditions of mutual exclusion, cyclic wait, hold and wait, and no preemption are satisfied	✓
● ☑	○ ✗	A deadlock necessarily requires a cycle in the resource allocation graph	✓
● ☑	○ ✗	Cycle in the resource allocation graph does not necessarily mean a deadlock	✓
● ✗	○ ☑	A deadlock must involve at least two processes	✗
○ ☑	● ✗	Deadlocks are not possible if there is no race	✗

Deadlocks are the same as livelocks: False
A deadlock is possible only if all the 4 conditions of mutual exclusion, cyclic wait, hold and wait, and no preemption are satisfied: True
A deadlock necessarily requires a cycle in the resource allocation graph: True
Cycle in the resource allocation graph does not necessarily mean a deadlock: True
A deadlock must involve at least two processes: False
Deadlocks are not possible if there is no race: True

Question **21**

Correct

Mark 0.50 out of 0.50

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

○ a. creates a 1-level page table for the use by the kernel, as specified in kmap[] global array

● b. creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray✓

○ c. creates a 2-level page table setup with virtual->physical mappings specified in the kmap[] global arrray and makes kpgdir point to it

○ 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 **22**

Correct

Mark 0.50 out of 0.50

Mark statements as T/F

All statements are in the context of preventing deadlocks.

True	False		
●✓	○✗	A process holding one resources and waiting for just one more resource can also be involved in a deadlock.	✓
●✓	○✗	Mutual exclusion is a necessary condition for deadlock because it brings in locks on which deadlock happens	✓
○✗	●✓	The lock ordering to be followed to avoid circular wait is a code in OS that checks for compliance with decided order	✓
●✓	○✗	Hold and wait means a thread/process holding some locks and waiting for acquiring some.	✓
○✗	●✓	If a resource allocation graph contains a cycle then there is a guarantee of a deadlock	✓
●✓	○✗	Circular wait is avoided by enforcing a lock ordering	✓
●✓	○✗	Deadlock is possible if all the conditions are met at the same time: Mutual exclusion, hold and wait, no pre-emption, circular wait.	✓

A process holding one resources and waiting for just one more resource can also be involved in a deadlock.: True
Mutual exclusion is a necessary condition for deadlock because it brings in locks on which deadlock happens: True
The lock ordering to be followed to avoid circular wait is a code in OS that checks for compliance with decided order: False
Hold and wait means a thread/process holding some locks and waiting for acquiring some.: True
If a resource allocation graph contains a cycle then there is a guarantee of a deadlock: False
Circular wait is avoided by enforcing a lock ordering: True
Deadlock is possible if all the conditions are met at the same time: Mutual exclusion, hold and wait, no pre-emption, circular wait.: True

Question **23**

Correct

Mark 0.50 out of 0.50

Will this code work for a spinlock() operation? The intention here is to call compare-and-swap() only if the lock is not held (the if condition checks for the same).

```
void spinlock(int *lock) {
{
    while (true) {
        if (*lock == 0) {
            /* lock appears to be available */
            if (!compare_and_swap(lock, 0, 1))
            break
        }
    }
}
```

○ a. No, because in the case of both processes succeeding in the "if" condition, both may end up acquiring the lock.

● b. Yes, because no matter in which order the if-check and compare-and-swap run in multiple processes, only one process will succeed in compare-and-swap() and others will keep looping in while-loop. ✓

○ c. Yes, because there is no race to update the lock variable

○ d. No, because this breaks the atomicity requirement of compare-and-test.

Your answer is correct.

The correct answer is: Yes, because no matter in which order the if-check and compare-and-swap run in multiple processes, only one process will succeed in compare-and-swap() and others will keep looping in while-loop.

Question **24**

Partially correct

Mark 0.38 out of 0.50

The kernel ELF file contains these headers

```
Program Header:
    LOAD off    0x00001000 vaddr 0x80100000 paddr 0x00100000 align 2**12
         filesz 0x00007aab memsz 0x00007aab flags r-x
    LOAD off    0x00009000 vaddr 0x80108000 paddr 0x00108000 align 2**12
         filesz 0x00002516 memsz 0x0000d4a8 flags rw-
   STACK off    0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**4
         filesz 0x00000000 memsz 0x00000000 flags rwx
```

mark the statemetns as True/False

True	False		
● ✓	○ ✗	Second header is for Data/Globals	✓
● ✓	○ ✗	First header is for the code/text	✓
○ ✓	● ✗	in bootmain() the third header leads to allocation of no-memory.	✗
● ✓	○ ✗	Third header is for stack	✓

Second header is for Data/Globals: True
First header is for the code/text: True
in bootmain() the third header leads to allocation of no-memory.: True
Third header is for stack: True

Question **25**

Correct

Mark 0.50 out of 0.50

Consider this program.
Some statements are identified using the // comment at the end.

Assume that = is an atomic operation.

```
#include <stdio.h>
#include <pthread.h>
long c = 0, c1 = 0, c2 = 0, run = 1;
void *thread1(void *arg) {
    while(run == 1) {//E
        c = 10; //A
        c1 = c2 + 5; //B
    }
}
void *thread2(void *arg) {
    while(run == 1) {//F
        c = 20;//C
        c2 = c1 + 3;//D
    }
}
int main() {
    pthread_t th1, th2;
    pthread_create(&th1, NULL, thread1, NULL);
    pthread_create(&th2, NULL, thread2, NULL);
    sleep(2);
    run = 0;
    fprintf(stdout, "c = %ld c1+c2 = %ld c1 = %ld c2 = %ld \n", c, c1+c2, c1, c2);
    fflush(stdout);
}
```

Which statements are part of the critical Section?

Yes	No		
○ ✗	● ✓	C	✓
● ✓	○ ✗	D	✓
○ ✗	● ✓	A	✓
○ ✗	● ✓	F	✓
● ✓	○ ✗	B	✓
○ ✗	● ✓	E	✓

C: No
D: Yes
A: No
F: No
B: Yes
E: No

Question **26**

Partially correct

Mark 0.38 out of 0.50

Match the code with it's functionality

S = 0
P1:
Statement1;
Signal(S)

Execution order P1, then P2 ✓

P2:
Wait(S)
Statment2;

S1 = 0; S2 = 0;
P2:
Statement1;
Signal(S2);

P1:
Wait(S2);
Statemetn2;
Signal(S1);

Execution order P3, P2, P1 ✗

P3:
Wait(S1);
Statement S3;

S = 1
Wait(S)
Critical Section
Signal(S);

Binary Semaphore for mutual exclusion ✓

S = 5
Wait(S)
Critical Section
Signal(S)

Counting semaphore ✓

Your answer is partially correct.

You have correctly selected 3.
The correct answer is: S = 0
P1:
Statement1;
Signal(S)

P2:
Wait(S)
Statment2; → Execution order P1, then P2, S1 = 0; S2 = 0;
P2:
Statement1;
Signal(S2);

P1:
Wait(S2);
Statemetn2;
Signal(S1);

P3:
Wait(S1);
Statement S3; → Execution order P2, P1, P3, S = 1
Wait(S)
Critical Section
Signal(S); → Binary Semaphore for mutual exclusion, S = 5
Wait(S)

Critical Section
Signal(S) → Counting semaphore

Question **27**

Correct

Mark 0.50 out of 0.50

In an ext2 file system, if the block size is 4KB and partition size is 128 GB, then the number of block groups will be:

Answer: 1024 ✓

size * 1024 * 1024 / 4 --> no of blocks

each group = 8 * 4 * 1024 blocks = 32768 blocks

so size * 1024 * 1024 / (4 * 32768) number of groups

The correct answer is: 1024.00

Question **28**

Correct

Mark 0.50 out of 0.50

The "push 0" in vectors.S is

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

The correct answer is: Place for the error number value

Question **29**

Partially correct

Mark 0.25 out of 0.50

Match pairs

mutex	atomic test and set with loop	✗
peterson	per process flag, global turn variable	✓
semaphore	wait() and signal()	✓
spinlock	lock() and unlock()	✗

Your answer is partially correct.

You have correctly selected 2.
The correct answer is: mutex → lock() and unlock(), peterson → per process flag, global turn variable, semaphore → wait() and signal(), spinlock → atomic test and set with loop

Question **30**

Correct

Mark 0.50 out of 0.50

Which of the following is done by mappages()?

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

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"

◄ Quiz-1 (15 Marks)

Jump to...

ESE(60 Marks) ►