Started on	Saturday, 22 May 2021, 8:00 AM
State	Finished
Completed on	Saturday, 22 May 2021, 9:30 AM
Time taken	1 hour 30 mins
Grade	**26.12** out of 40.00 (**65**%)

Question **1**
Incorrect
Mark 0.00 out of 1.00

A 4 GB disk with 1 KB of block size would require these many number of **blocks** for it's free block bitmap:

Answer: 4096 ✖

The correct answer is: 512

Question **2**
Correct
Mark 1.00 out of 1.00

Given that the memory access time is 110 ns, probability of a page fault is 0.5 and page fault handling time is 12 ms,
The effective memory access time in nanoseconds is:

Answer: 6000165 ✔

The correct answer is: 6000055.00

Question **3**
Incorrect
Mark 0.00 out of 1.00

The maximum size of a file in number of blocks of BSIZE in xv6 code is

(write a number only)

Answer: 268 ✖

The correct answer is: 138

Question **4**
Incorrect
Mark 0.00 out of 1.00

Calculate the average waiting time using
Round Robin scheduling with time quantum of 5 time units
for the following workload

assuming that they arrive in the order written below.

Process	Burst Time
P1	5
P2	7
P3	6
P4	2

Write only a number in the answer upto two decimal points.

Answer: 40.75 ✖

The correct answer is: 10.25

Question **5**
Correct
Mark 1.00 out of 1.00

For the reference string

4 2 5 1 0 1 2 5 4 1 2

the number of page faults, including initial ones,

with FIFO replacement and 2 frames are :

Answer: 10

4 -

4 2

5 2

5 1

0 1

-

2 1

2 5

4 5

4 1

2 1

The correct answer is: 10

Question **6**
Correct
Mark 1.00 out of 1.00

Assuming a 16- KB page size, what is the page number for the address 428517 reference in decimal :

(give answer also in decimal)

Answer: 27

The correct answer is: 26

Question **7**

Correct

Mark 1.00 out of 1.00

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(b);
    //some code;
    spinlock(a);
    //some code;
    spinunlock(b);
    spinunlock(a);
}
thread2() {
    spinlock(a);
    //some code;
    spinlock(b);
    //some code;
    spinunlock(b);
    spinunlock(a);
}
```

- a. Deadlock
- b. Self Deadlock
- c. None of these
- d. Deadlock or livelock depending on actual race
- e. Livelock

Your answer is correct.

The correct answer is: Deadlock

Question **8**

Partially correct

Mark 1.33 out of 2.00

Match the snippets of xv6 code with the core functionality they achieve, or problems they avoid.
"..." means some code.

```
static inline uint
xchg(volatile uint *addr, uint newval)
{
  uint result;

  // The + in "+m" denotes a read-modify-write operand.
  asm volatile("lock; xchgl %0, %1" :
             "+m" (*addr), "=a" (result) :
             "1" (newval) :
             "cc");
  return result;
}
```

Atomic compare and swap instruction (to be expanded inline into code) ✔

```
void
sleep(void *chan, struct spinlock *lk)
{
  ...
  if(lk != &ptable.lock){
    acquire(&ptable.lock);
    release(lk);
  }
```

If you don't do this, a process may be running on two processors parallely ✖

```
void
acquire(struct spinlock *lk)
{
...
__sync_synchronize();
```

Tell compiler not to reorder memory access beyond this line ✔

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: static inline uint
xchg(volatile uint *addr, uint newval)
{
  uint result;

  // The + in "+m" denotes a read-modify-write operand.
  asm volatile("lock; xchgl %0, %1" :
             "+m" (*addr), "=a" (result) :
             "1" (newval) :
             "cc");
  return result;
} → Atomic compare and swap instruction (to be expanded inline into code), void
sleep(void *chan, struct spinlock *lk)
{
  ...
  if(lk != &ptable.lock){
    acquire(&ptable.lock);
    release(lk);
  } → Avoid a self-deadlock, void
acquire(struct spinlock *lk)
{
...
__sync_synchronize(); → Tell compiler not to reorder memory access beyond this line

Question **9**

Correct

Mark 1.00 out of 1.00

Predict the output of the program given here.

Assume that all the path names for the programs are correct. For example "/usr/bin/echo" will actually run echo command.

Assume that there is no mixing of printf output on screen if two of them run concurrently.

In the answer replace a new line by a single space.

For example::

`good`

`output`

should be written as `good output`

```
--
main() {
    int i;
    i = fork();
    if(i == 0)
        execl("/usr/bin/echo", "/usr/bin/echo", "hi", 0);
    else
        wait(0);
    fork();
    execl("/usr/bin/echo", "/usr/bin/echo", "one", 0);
}
```

Answer: hi one one ✔

The correct answer is: hi one one

Question **10**

Partially correct

Mark 1.67 out of 2.00

Select all the blocks that may need to be written back to disk (if updated, of-course), as "Yes", when an operation of deleting a file is carried out on ext2 file system.

An option has to be correct entirely to be marked "Yes"

Superblock	Yes	✔
One or multiple data blocks of the parent directory	No	✔
One or more data bitmap blocks for the parent directory	No	✔
Block bitmap(s) for all the blocks of the file	No	✖
Possibly one block bitmap corresponding to the parent directory	Yes	✔
Data blocks of the file	No	✔

Your answer is partially correct.

only one data block of parent directory. multiple blocks not possible. an entry is always contained within one single block

You have correctly selected 5.

The correct answer is: Superblock → Yes, One or multiple data blocks of the parent directory → No, One or more data bitmap blocks for the parent directory → No, Block bitmap(s) for all the blocks of the file → Yes, Possibly one block bitmap corresponding to the parent directory → Yes, Data blocks of the file → No

Question **11**

Correct

Mark 1.00 out of 1.00

Select all the correct statements about bootloader.

Every wrong selection will deduct marks proportional to 1/n where n is total wrong choices in the question.

You will get minimum a zero.

- [x] a. Modern Bootloaders often allow configuring the way an OS boots ✔
- [x] b. Bootloaders allow selection of OS to boot from ✔
- [ ] c. Bootloader must be one sector in length
- [ ] d. The bootloader loads the BIOS
- [x] e. LILO is a bootloader ✔

Your answer is correct.

The correct answers are: LILO is a bootloader, Modern Bootloaders often allow configuring the way an OS boots, Bootloaders allow selection of OS to boot from

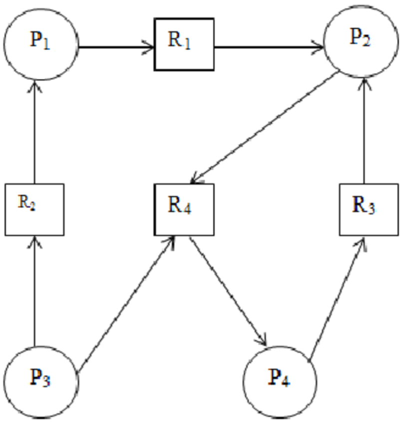
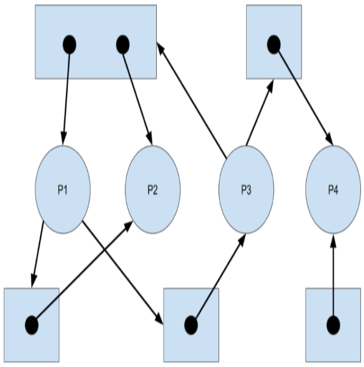
Question **12**

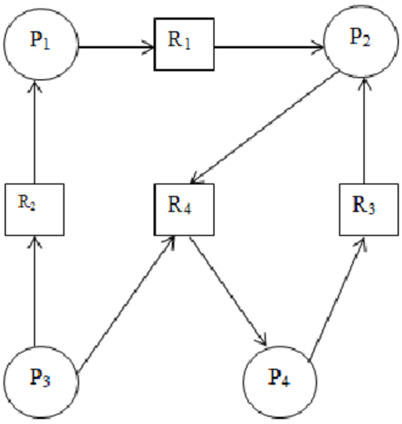
Incorrect

Mark 0.00 out of 1.00

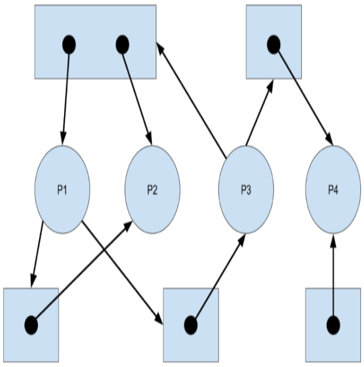
For each of the resource allocation diagram shown,

infer whether the graph contains at least one deadlock or not.

Yes	No		
	◉		✖
◉			✖



: Yes



: No

Question **13**

Partially correct

Mark 0.71 out of 1.00

Mark the statements about device drivers by marking as True or False.

True	False		
○	◉	It's possible that a particular hardware has multiple device drivers available for it.	✗
◉	○	xv6 has device drivers for IDE disk and console.	✓
◉	○	A disk driver converts OS's logical view of disk into physical locations on disk.	✓
◉	○	A device driver code is specific to a hardware device	✓
◉	○	All devices of the same type (e.g. 2 hard disks) can typically use the same device driver	✓
○	◉	Writing a device driver mandatorily demands reading the technical documentation about the hardware.	✗
○	◉	Device driver is an intermediary between the end-user and OS	✓

It's possible that a particular hardware has multiple device drivers available for it.: True
xv6 has device drivers for IDE disk and console.: True
A disk driver converts OS's logical view of disk into physical locations on disk.: True
A device driver code is specific to a hardware device: True
All devices of the same type (e.g. 2 hard disks) can typically use the same device driver: True
Writing a device driver mandatorily demands reading the technical documentation about the hardware.: True
Device driver is an intermediary between the end-user and OS: False

Question **14**

Partially correct

Mark 0.33 out of 1.00

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		
● ✖	○ ✔	F	✖
● ✔	○ ✖	D	✔
● ✖	○ ✔	C	✖
● ✖	○ ✔	A	✖
● ✔	○ ✖	B	✔
● ✖	○ ✔	E	✖

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

Question **15**
Partially correct
Mark 1.43 out of 2.00

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

A process holding one resources and waiting for just one more resource can also be involved in a deadlock.: True
If a resource allocation graph contains a cycle then there is a guarantee of a deadlock: False
The lock ordering to be followed to avoid circular wait is a code in OS that checks for compliance with decided order: False
Circular wait is avoided by enforcing a lock ordering: True
Hold and wait means a thread/process holding some locks and waiting for acquiring some.: 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
Mutual exclusion is a necessary condition for deadlock because it brings in locks on which deadlock happens: True

Question **16**
Correct
Mark 1.00 out of 1.00

Match the left side use(or non-use) of a synchronization primitive with the best option on the right side.

This is the smallest primitive made available in software, using the hardware provided atomic instructions	spinlock	✔
This tool is useful for event-wait scenarios	semaphore	✔
This tool is more useful on multiprocessor systems	spinlock	✔
This tool is quite attractive in solving the main bounded buffer problem	semaphore	✔
This tool is very useful for waiting for 'something'	condition variables	✔

Your answer is correct.

The correct answer is: This is the smallest primitive made available in software, using the hardware provided atomic instructions → spinlock, This tool is useful for event-wait scenarios → semaphore, This tool is more useful on multiprocessor systems → spinlock, This tool is quite attractive in solving the main bounded buffer problem → semaphore, This tool is very useful for waiting for 'something' → condition variables

Question **17**

Correct

Mark 1.00 out of 1.00

The permissions -rwx--x--x on a file mean

- [x] a. The file can be read only by the owner ✔
- [ ] b. 'cat' on the file by owner will not work
- [ ] c. 'cat' on the file by any user will work
- [x] d. 'rm' on the file by any user will work ✔
- [x] e. The file can be executed by anyone ✔
- [x] f. The file can be written only by the owner ✔

Your answer is correct.

The correct answers are: The file can be executed by anyone, The file can be read only by the owner, The file can be written only by the owner, 'rm' on the file by any user will work

Question **18**

Incorrect

Mark 0.00 out of 1.00

Note: for this question you get full marks if you select all and only correct options, you get ZERO if at least one option is wrong or not selected.

Select all the correct statements about log structured file systems.

- [x] a. a transaction is said to be committed when all operations are written to file system ✖
- [x] b. log may be kept on same block device or another block device ✔
- [ ] c. file system recovery may end up losing data
- [x] d. even if file systems followed immediate writes (i.e. non-delayed writes), it could still require recovery ✔
- [x] e. file system recovery recovers all the lost data ✖

Your answer is incorrect.

The correct answers are: file system recovery may end up losing data, log may be kept on same block device or another block device, even if file systems followed immediate writes (i.e. non-delayed writes), it could still require recovery

Question **19**

Incorrect

Mark 0.00 out of 1.00

Consider the structure of directory entry in ext2, as shown in this diagram.

	inode	rec_len	name_len	file_type	name							
0	21	12	1	2	.	\0	\0	\0				
12	22	12	2	2	.	.	\0	\0				
24	53	16	5	2	h	o	m	e	1	\0	\0	\0
40	67	28	3	2	u	s	r	\0				
52	0	16	7	1	o	l	d	f	i	l	e	\0
68	34	12	4	2	s	b	i	n				

Select the correct statements about the directory entry in ext2 file system.

The correct formula for rec_len is (when entries are continuously stored)

- a. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) + (-1) * (strlen(name)%4)
- b. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) + (strlen(name)- 4) %4
- c. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) + 4 - (strlen(name)%4) ✖
- d. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) + (-1) * (strlen(name)- 4)
- e. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) %4
- f. rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + strlen(name)

Your answer is incorrect.

The correct answer is: rec_len = sizeof(inode entry) + sizeof(name len entry) + sizeof(file type entry) + (strlen(name) + (-1) * (strlen(name)- 4)

Question **20**

Partially correct

Mark 0.50 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 executing a system call
Timer interrupt
Generic interrupt handler runs
Scheduler runs
Scheduler selects P2 for execution
P2 returns from timer interrupt handler
Process p2, user code executing
```

This sequence of events is: not-possible ✔

Because

Interrupts are not possible when kernel code is running ✖

Question **21**

Incorrect

Mark 0.00 out of 1.00

The given semaphore implementation faces which problem?

Assume any suitable code for signal()

Note: blocks means waits in a wait queue.

```
struct semaphore {
        int val;
        spinlock lk;
};
sem_init(semaphore *s, int initval) {
        s->val = initval;
        s->sl = 0;
}
wait(semaphore *s) {
        spinlock(&(s->sl));
        while(s->val <=0)
        ;
        (s->val)--;
        spinunlock(&(s->sl));
}
```

- a. blocks holding a spinlock
- b. deadlock
- c. too much spinning, bounded wait not guaranteed
- d. not holding lock after unblock

Your answer is incorrect.

The correct answer is: deadlock

Question **22**

Partially correct

Mark 0.80 out of 1.00

Mark statements True/False w.r.t. change of states of a process.

Reference: The process state diagram (and your understanding of how kernel code works)

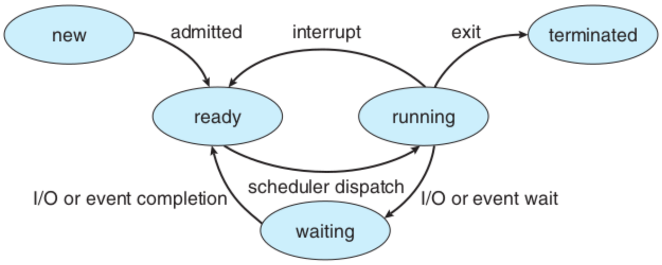


Figure 3.2 Diagram of process state.

True	False		
○ ✗	● ✓	A process in RUNNING state only can become TERMINATED because scheduler moves it to ZOMBIE state	✓
● ✗	○ ✓	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 in WAITING state can not become RUNNING because the event it's waiting for has not occurred	✓
● ✓	○ ✗	Every 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 RUNNING state only can become TERMINATED because scheduler moves it to ZOMBIE state: False
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 in WAITING state can not become RUNNING because the event it's waiting for has not occurred: True
Every 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

Question **23**
Correct
Mark 1.00 out of 1.00

Select T/F for statements about Volume Managers.
Do pay attention to the use of the words physical partition and physical volume.

True	False			
● ✓	○ ✗	The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.	✔	
● ✓	○ ✗	A logical volume can be extended in size but upto the size of volume group	✔	
● ✓	○ ✗	A logical volume may span across multiple physical volumes	✔	
● ✓	○ ✗	The volume manager stores additional metadata on the physical disk partitions	✔	
● ✓	○ ✗	A physical partition should be initialized as a physial volume, before it can be used by volume manager.	✔	
● ✓	○ ✗	A volume group consists of multiple physical volumes	✔	
● ✓	○ ✗	A logical volume may span across multiple physical partitions	✔	since a physical volume is made up of physical partitions, and a volume can span across multiple PVs, it can also span across multiple PP

The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.: True
A logical volume can be extended in size but upto the size of volume group: True
A logical volume may span across multiple physical volumes: True
The volume manager stores additional metadata on the physical disk partitions: True
A physical partition should be initialized as a physial volume, before it can be used by volume manager.: True
A volume group consists of multiple physical volumes: True
A logical volume may span across multiple physical partitions: True

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

Map the block allocation scheme with the problem it suffers from

(Match pairs 1-1, match a scheme with the problem that it suffers from relatively the most, compared to others)

Continuous allocation	need for compaction	✔
Linked allocation	Too many seeks	✔
Indexed Allocation	Overhead of reading metadata blocks	✔

Your answer is correct.

The correct answer is: Continuous allocation → need for compaction, Linked allocation → Too many seeks, Indexed Allocation → Overhead of reading metadata blocks

Question **25**
Correct
Mark 1.00 out of 1.00

This one is not a system call:

○ a. open

○ b. read

○ c. write

● d. scheduler ✔

Your answer is correct.

The correct answer is: scheduler

Question **26**
Correct
Mark 1.00 out of 1.00

Match the pairs.

This question is based on your general knowledge about operating systems/related concepts and their features.

Java thereads	monitors,re-entrant locks, semaphores ✔
Linux threads	atomic-instructions, spinlocks, etc. ✔
POSIX threads	semaphore, mutex, condition variables ✔

Your answer is correct.

The correct answer is: Java thereads → monitors,re-entrant locks, semaphores, Linux threads → atomic-instructions, spinlocks, etc., POSIX threads → semaphore, mutex, condition variables

Question **27**
Correct
Mark 1.00 out of 1.00

Consider the following list of free chunks, in continuous memory management:

7k, 15k, 21k, 14k, 19k, 6k

Suppose there is a request for chunk of size 5k, then the free chunk selected under each of the following schemes will be

Best fit: 6k ✔
First fit: 7k ✔
Worst fit: 21k ✔

Question **28**
Correct
Mark 1.00 out of 1.00

This one is not a scheduling algorithm

- a. Round Robin
- b. SJF
- c. Mergesort ✔
- d. FCFS

Your answer is correct.

The correct answer is: Mergesort

Question **29**
Correct
Mark 1.00 out of 1.00

Mark whether the concept is related to scheduling or not.

Yes	No		
● ✓	○ ✗	timer interrupt	✓
● ✓	○ ✗	context-switch	✓
● ✓	○ ✗	ready-queue	✓
○ ✗	● ✓	file-table	✓
● ✓	○ ✗	runnable process	✓

timer interrupt: Yes
context-switch: Yes
ready-queue: Yes
file-table: No
runnable process: Yes

Question **30**

Partially correct

Mark 1.00 out of 2.00

Map ext2 data structure features with their purpose

Many copies of Superblock	Choose...
Free blocks count in superblock and group descriptor ✖	Redundancy to ensure the most crucial data structure is not lost
Used directories count in group descriptor ✖	is redundant and helps do calculations of directory entries faster
Combining file type and access rights in one variable ✔	saves 1 byte of space
rec_len field in directory entry ✖	Try to keep all the data of a directory and it's file close together in a group
File Name is padded ✔	aligns all memory accesses on word boundary, improving performance
Inode bitmap is one block ✔	limits total number of files that can belong to a group
Block bitmap is one block ✔	Limits the size of a block group, thus improvising on purpose of a group
Mount count in superblock ✔	to enforce file check after certain amount of mounts at boot time
Inode table location in Group Descriptor ✖	is redundant and helps do calculations of directory entries faster
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
A group ✖	Redundancy to ensure the most crucial data structure is not lost

Your answer is partially correct.

You have correctly selected 6.

The correct answer is: Many copies of Superblock → Redundancy to ensure the most crucial data structure is not lost, Free blocks count in superblock and group descriptor → Redundancy to help fsck restore consistency, Used directories count in group descriptor → attempt is made to evenly spread the first-level directories, this count is used there, Combining file type and access rights in one variable → saves 1 byte of space, rec_len field in directory entry → allows holes and linking of entries in directory, File Name is padded → aligns all memory accesses on word boundary, improving performance, Inode bitmap is one block → limits total number of files that can belong to a group, Block bitmap is one block → Limits the size of a block group, thus improvising on purpose of a group, Mount count in superblock → to enforce file check after certain amount of mounts at boot time, 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, A group → Try to keep all the data of a directory and it's file close together in a group

Question **31**
Partially correct
Mark 1.85 out of 2.00

Mark True/False

Statements about scheduling and scheduling algorithms

True	False		
●	○	The nice() system call is used to set priorities for processes	✔
●	○	Aging is used to ensure that low-priority processes do not starve in priority scheduling.	✔
○	●	In non-pre-emptive priority scheduling, the highest priority process is scheduled and runs until it gives up CPU.	✖
●	○	xv6 code does not care about Processor Affinity	✔
●	○	In pre-emptive priority scheduling, priority is implemented by assigning more time quantum to higher priority process.	✔
●	○	A scheduling algorithm is non-premptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.	✔
●	○	Processor Affinity refers to memory accesses of a process being stored on cache of that processor	✔
●	○	Response time will be quite poor on non-interruptible kernels	✔
●	○	Shortest Remaining Time First algorithm is nothing but pre-emptive Shortest Job First algorithm	✔
●	○	On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread	✔
●	○	Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.	✔
●	○	Pre-emptive scheduling leads to many race conditions in kernel code.	✔
●	○	Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.	✔

The nice() system call is used to set priorities for processes: True
Aging is used to ensure that low-priority processes do not starve in priority scheduling.: True
In non-pre-emptive priority scheduling, the highest priority process is scheduled and runs until it gives up CPU.: True
xv6 code does not care about Processor Affinity: True
In pre-emptive priority scheduling, priority is implemented by assigning more time quantum to higher priority process.: True
A scheduling algorithm is non-premptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.: True
Processor Affinity refers to memory accesses of a process being stored on cache of that processor: True
Response time will be quite poor on non-interruptible kernels: True
Shortest Remaining Time First algorithm is nothing but pre-emptive Shortest Job First algorithm: True
On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread: True
Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.: True
Pre-emptive scheduling leads to many race conditions in kernel code.: True
Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.: True

Question **32**
Partially correct
Mark 1.17 out of 2.00

The unix file semantics demand that changes to any open file are visible immediately to any other processes accessing that file at that point in time.

Select the data-structure/programmatic features that ensure the implementation of unix semantics. (Assume there is no mmap())

Yes	No		
	●	All processes accessing the same file share the file descriptor among themselves	✔
	●	The pointer entry in the file descriptor array entry points to the data of the file directly	✔
●		There is only one global file structure per on-disk file.	✖
	●	All file accesses are made using only global variables	✔
●		The 'file offset' is shared among all the processes that access the file.	✖
	●	No synchronization is implemented so that changes are made available immediately.	✔
●		A single spinlock is to be used to protect the unique global 'file structure' representing the file, thus synchronizing access, and making other processes wait for earlier process to finish writing so that writes get visible immediately.	✖
●		There is only one in-memory copy of the on disk file's contents in kernel memory/buffers	✔
●		The file descriptors in every PCB are pointers to the same global file structure.	✖
	●	The file descriptor array is external to PCB and all processes that share a file, have pointers to same file-descriptors' array	✔
●		All file structures representing any open file, give access to the same in-memory copy of the file's contents	✔
●		The 'file offset' index is stored outside the file-structure to which file-descriptor array points	✖

All processes accessing the same file share the file descriptor among themselves: No
The pointer entry in the file descriptor array entry points to the data of the file directly: No
There is only one global file structure per on-disk file.: No
All file accesses are made using only global variables: No
The 'file offset' is shared among all the processes that access the file.: No
No synchronization is implemented so that changes are made available immediately.: No
A single spinlock is to be used to protect the unique global 'file structure' representing the file, thus synchronizing access, and making other processes wait for earlier process to finish writing so that writes get visible immediately.: No
There is only one in-memory copy of the on disk file's contents in kernel memory/buffers: Yes
The file descriptors in every PCB are pointers to the same global file structure.: No
The file descriptor array is external to PCB and all processes that share a file, have pointers to same file-descriptors' array: No
All file structures representing any open file, give access to the same in-memory copy of the file's contents: Yes
The 'file offset' index is stored outside the file-structure to which file-descriptor array points: No

Question **33**

Partially correct

Mark 0.33 out of 2.00

Map the function in xv6's file system code, to it's perceived logical layer.

namei	inode	✖
filestat()	Choose...	
dirlookup	directory	✔
ialloc	file descriptor	✖
stati	Choose...	
ideintr	buffer cache	✖
bread	Choose...	
balloc	file descriptor	✖
sys_chdir()	system call	✔
skipelem	system call	✖
commit	system call	✖
bmap	system call	✖

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: namei → pathname lookup, filestat() → file descriptor, dirlookup → directory, ialloc → inode, stati → inode, ideintr → disk driver, bread → buffer cache, balloc → block allocation on disk, sys_chdir() → system call, skipelem → pathname lookup, commit → logging, bmap → inode

◄ Course Exit Feedback

Jump to...

xv6-public-master ►