Started on	Saturday, 4 May 2024, 1:06 PM
State	Finished
Completed on	Saturday, 4 May 2024, 5:58 PM
Time taken	4 hours 52 mins
Grade	Not yet graded

Question **1**

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 quite attractive in solving the main bounded buffer problem	semaphore ✔
This tool is very useful for waiting for 'something'	condition variables ✔
This tool is useful for event-wait scenarios	semaphore ✔
This tool is more useful on multiprocessor systems	spinlock ✔

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 quite attractive in solving the main bounded buffer problem → semaphore, This tool is very useful for waiting for 'something' → condition variables, This tool is useful for event-wait scenarios → semaphore, This tool is more useful on multiprocessor systems → spinlock

Question **2**

Partially correct

Mark 0.43 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		
◉☑	○✖	A physical partition(made available as physical volume) can belong to only one volume group	✔
◉✖	○☑	A volume group consists of multiple physical partitions	✖
○✖	◉☑	A volume group can contain physical partitions(as physical volumes) from only one physical disk	✔
○☑	◉✖	Extending the size of a logical volume does not automatically change the size of the file system	✖
○✖	◉☑	A logical volume can span across two volume groups	✔
◉✖	○☑	A volume group consists of multiple physical partitions	✖
◉✖	○☑	A logical volume can be extended in size but upto the size of the maximum sized physical volume only	✖

A physical partition(made available as physical volume) can belong to only one volume group: True
A volume group consists of multiple physical partitions: False
A volume group can contain physical partitions(as physical volumes) from only one physical disk: False
Extending the size of a logical volume does not automatically change the size of the file system: True
A logical volume can span across two volume groups: False
A volume group consists of multiple physical partitions: False
A logical volume can be extended in size but upto the size of the maximum sized physical volume only: False

Question **3**

Correct

Mark 0.50 out of 0.50

Match the parts of the code, with their description

```
void *thread1(void *arg) {
   while(run == 1) {
      pthread_mutex_lock(&lock);
      c++;
      pthread_mutex_unlock(&lock);
      c1++;
   }
}
```

pthread_mutex_unlock(&lock);	exit section ✔
c1++;	reminder ✔
c++;	critical section ✔
pthread_mutex_lock(&lock);	entry section ✔

Your answer is correct.

The correct answer is: pthread_mutex_unlock(&lock); → exit section, c1++; → reminder, c++; → critical section, pthread_mutex_lock(&lock); → entry section

Question **4**

Partially correct

Mark 0.78 out of 1.00

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

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

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 LRU 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

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		
◉☑	○✖	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.	✔
◉☑	○✖	Local replacement can lead to under-utilisation of memory, because a process may not use all the pages allocated to it all the time.	✔
◉☑	○✖	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 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.: Consistent
Global replacement may give highly variable per process completion time because number of page faults become un-predictable.: 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
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 results in more predictable per-process completion time because number of page faults can be better predicted.: Consistent

Question **7**

Correct

Mark 1.00 out of 1.00

Select all the actions taken by iget()

- [ ] a. Returns the inode with inode-cache lock held
- [ ] b. Panics if inode does not exist in cache
- [x] c. Returns the inode with reference count incremented ✔
- [ ] d. Returns a valid inode if not found in cache
- [ ] e. Returns the inode locked
- [x] f. Returns an inode with given dev+inode-number from cache, if it exists in cache ✔
- [x] g. Returns a free-inode , with dev+inode-number set, if not found in cache ✔

Your answer is correct.

The correct answers are: Returns an inode with given dev+inode-number from cache, if it exists in cache, Returns the inode with reference count incremented, Returns a free-inode , with dev+inode-number set, if not found in cache

Question **8**

Partially correct

Mark 0.86 out of 1.00

Select T/F w.r.t physical disk handling in xv6 code

True	False		
✗	☑	only direct blocks are supported	✗
✗	☑	device files are not supported	✓
☑	✗	The code supports IDE, and not SATA/SCSI	✓
☑	✗	disk driver handles only one buffer at a time	✓
☑	✗	only 2 disks are handled by default	✓
☑	✗	log is kept on the same device as the file system	✓
☑	✗	the superblock does not contain number of free blocks	✓

only direct blocks are supported: False
device files are not supported: False
The code supports IDE, and not SATA/SCSI: True
disk driver handles only one buffer at a time: True
only 2 disks are handled by default: True
log is kept on the same device as the file system: True
the superblock does not contain number of free blocks: True

Question **9**

Correct

Mark 1.00 out of 1.00

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

8k, 1k, 2k, 14k, 13k, 7k, 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: 8k ✓

Worst fit: 14k ✓

Question **10**

Correct

Mark 1.00 out of 1.00

Select T/F for the disk block allocation scheme related statements

True	False		
◉ ☑	○ ✖	FAT uses linked allocation	✔
◉ ☑	○ ✖	NVM storage devices are pushing for search for new block allocation schemes	✔
◉ ☑	○ ✖	Continuous allocation leads to faster file access	✔
◉ ☑	○ ✖	Continuous allocation allows to fetch any block with just one seek	✔
◉ ☑	○ ✖	Continuous allocation may involve a costly search for free space	✔
◉ ☑	○ ✖	The unix inode is based on indexed allocation	✔
◉ ☑	○ ✖	Maximum file size limit is determined by disk block allocation scheme, as one of the factors.	✔
◉ ☑	○ ✖	Linked allocation does away with file size limitation to a large extent	✔

FAT uses linked allocation: True
NVM storage devices are pushing for search for new block allocation schemes: True
Continuous allocation leads to faster file access: True
Continuous allocation allows to fetch any block with just one seek: True
Continuous allocation may involve a costly search for free space: True
The unix inode is based on indexed allocation: True
Maximum file size limit is determined by disk block allocation scheme, as one of the factors.: True
Linked allocation does away with file size limitation to a large extent: True

Question **11**

Correct

Mark 2.00 out of 2.00

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 : No ✔

b. Install a bigger paging disk. : No ✔

c. Increase the degree of multiprogramming. : No ✔

d. Decrease the degree of multiprogramming. : Yes ✔

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. : Yes ✔

Question **12**

Complete

Marked out of 2.00

Which changes should be made to xv6 code to implement the vfork() system call ?

List your suggestions in a bullet-point fashion. Each point should mention

(a) pseudo-code of new function to be added

(b) prototype of any new function or new system call to be added

(c) pseudo-code of changes to an existing function, describing lines to be removed, and lines to be added

(d) precise declaration of new data structures to be added in C, or changes to the existing data structure

(e) Name and a one-line description of new userland functionality to be added

(f) Changes to Makefile

(g) Any other change in a maximum of 20 words per change.

void
vfork()

Question **13**

Correct

Mark 1.00 out of 1.00

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

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 ✔
bootmain()	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 ✔
entry.S	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 ✔

Your answer is correct.

The correct answer is: 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, bootmain() → 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, entry.S → 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

Question **14**

Correct

Mark 0.50 out of 0.50

You must have seen the error message "Segmentation fault, core dumped" very often.

With respect to this error message, mark the statements as True/False.

True	False			
✗	✓	On Linux, the message is printed only because the memory management scheme is segmentation	✓	No, it's just a term used, even if paging is used for memory management.
✓	✗	The process has definitely performed illegal memory access.	✓	
✓	✗	The core file can be analysed later using a debugger, to determine what went wrong.	✓	use gdb ./core ./executable-filename
✗	✓	The term "core" refers to the core code of the kernel.	✓	core means memory, all memory for the process.
✓	✗	The image of the process is stored in a file called "core", if the ulimit allows so.	✓	see ulimit -a
✗	✓	The illegal memory access was detected by the kernel and the process was punished by kernel.	✓	"detection" is done by CPU, not kernel.
✓	✗	On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.	✓	

On Linux, the message is printed only because the memory management scheme is segmentation: False
The process has definitely performed illegal memory access.: True
The core file can be analysed later using a debugger, to determine what went wrong.: True
The term "core" refers to the core code of the kernel.: False
The image of the process is stored in a file called "core", if the ulimit allows so.: True
The illegal memory access was detected by the kernel and the process was punished by kernel.: False
On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.: True

Question **15**

Correct

Mark 0.50 out of 0.50

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

- [x] a. exec() loads from ELF file and the address of first instruction to be executed is given by 'entry' ✔
- [ ] b. the 'entry' in initcode is anyways 0
- [ ] c. the code of 'initcode' is loaded at physical address 0
- [x] d. In userinit() the function inituvm() has mapped the code of 'initcode' to be starting at virtual address 0 ✔
- [ ] e. elf.entry is anyways 0, so both statements mean the same
- [x] 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 **16**

Correct

Mark 0.50 out of 0.50

Select the compiler's view of the process's address space, for each of the following MMU schemes:
(Assume that each scheme,e.g. paging/segmentation/etc is effectively utilised)

Relocation + Limit	one continuous chunk	✔
Segmentation	many continuous chunks of variable size	✔
Segmentation, then paging	many continuous chunks of variable size	✔
Paging	one continuous chunk	✔

Your answer is correct.

The correct answer is: Relocation + Limit → one continuous chunk, Segmentation → many continuous chunks of variable size, Segmentation, then paging → many continuous chunks of variable size, Paging → one continuous chunk

Question **17**

Correct

Mark 1.00 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.
- [x] b. The meaning of valid-invalid bit in page table is different in paging and demand-paging. ✔
- [x] c. Demand paging always increases effective memory access time. ✔
- [x] d. Demand paging requires additional hardware support, compared to paging. ✔
- [x] e. Paging requires some hardware support in CPU ✔
- [x] f. Calculations of number of bits for page number and offset are same in paging and demand paging. ✔
- [ ] g. With paging, it's possible to have user programs bigger than physical memory.
- [x] h. Both demand paging and paging support shared memory pages. ✔
- [x] i. With demand paging, it's possible to have user programs bigger than physical memory. ✔
- [ ] j. Paging requires NO hardware support in CPU

Your answer is correct.

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 **18**

Correct

Mark 0.50 out of 0.50

Which of the following statements are correct about xv6 and multiprocessor support ?

- [x] a. At any point in time, after main(), the kernel may be parallely executing on any of the processors. ✔
- [x] b. Each processor on xv6 starts code execution in mpenter() when first processor sets "started" to 1. ✔
- [x] c. In xv6 on x86, the first processor configures other processors in mpinit() ✔
- [x] d. xv6 supports only SMP ✔
- [x] e. xv6 supports a variable number of processors (upto 8) and adjusts it's data structures according to number of processors ✔

The correct answers are: xv6 supports only SMP, xv6 supports a variable number of processors (upto 8) and adjusts it's data structures according to number of processors, Each processor on xv6 starts code execution in mpenter() when first processor sets "started" to 1., In xv6 on x86, the first processor configures other processors in mpinit(), At any point in time, after main(), the kernel may be parallely executing on any of the processors.

Question **19**

Correct

Mark 0.50 out of 0.50

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

Your answer is correct.

Question **20**

Correct

Mark 1.00 out of 1.00

The given semaphore implementation faces which problem?

Note: blocks means waits in a wait queue.

```
struct semaphore {
        int val;
        spinlock lk;
        list l;
};
sem_init(semaphore *s, int initval) {
        s->val = initval;
        s->sl = 0;
}
block(semaphore *s) {
        listappend(s->l, current);
        spinunlock(&(s->sl));
        schedule();
}
wait(semaphore *s) {
        spinlock(&(s->sl));
        while(s->val <=0) {
                block(s);
        }
        (s->val)--;
        spinunlock(&(s->sl));
}
signal(seamphore *s) {
        spinlock(*(s->sl));
        (s->val)++;
        x = dequeue(s->sl) and enqueue(readyq, x);
        spinunlock(*(s->sl));
}
```

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

Your answer is correct.

The correct answer is: not holding lock after unblock

Question **21**

Correct

Mark 1.00 out of 1.00

Select the proper order of execution enforced by the given semaphore code

P1, P2, P3 are processes and S1 , S2 are binary semaphores.

```
S1 = 0; S2 = 0;

P1:
Wait(S2);
Statement2;

P2:
Statement1;
Signal(S1);

P3:
Wait(S1)
Statement S3;
Signal(S2);
```

Answer:

P2, P3, P1 ✔

Question **22**

Correct

Mark 1.00 out of 1.00

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

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

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 **23**

Correct

Mark 1.00 out of 1.00

Calculate the average waiting time using
FCFS scheduling
for the following workload

assuming that they arrive in this order during the first time unit:

Process	Burst Time
P1	2
P2	6
P3	2
P4	3

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

Answer: 5 ✔

P2 waits for 2 units

P3 waits for 2+6 units

P4 waits for 2 + 6 +2 units of time

Total waiting = 2 + 2 + 6 + 2 + 6 + 2 = 20 units

Average waiting time = 20/4 = 5

The correct answer is: 5

Question **24**

Correct

Mark 1.00 out of 1.00

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

Select one or more:

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

Your answer is correct.

The correct answers are: paging, segmentation, demand paging

Question **25**

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

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

Your answer is correct.

The correct answer is: None of these

Question **26**

Correct

Mark 1.00 out of 1.00

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 2 and 3 frames
- c. Exhibit Balady's anomaly between 3 and 4 frames

Your answer is correct.

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

Question **27**

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:

11010010

Now, there is a request for a chunk of 45 bytes.

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

Answer: 11011110 ✔

The correct answer is: 11011110

Question **28**

Correct

Mark 0.50 out of 0.50

What does seginit() do?

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

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

Question **29**

Complete

Marked out of 2.00

Write all changes required to xv6 to add a buddy allocator.

Every change should be mentioned in terms of either of the following:

(a) pseudo-code of new function to be added

(b) prototype of any new function or new system call to be added

(c) pseudo-code of changes to an existing function, describing lines to be removed, and lines to be added

(d) **precise** declaration of new data structures to be added in C, or changes to the existing data structure

(e) Name and a one-line description of new userland functionality to be added

(f) Changes to Makefile

(g) Any other change in a maximum of 20 words per change.

```
//Buddy structure
struct buddy
{ void *base;
size_t size;
struct buddy *next;
}
//free list array
struct buddy *freelist[MAX_ORDER];
//new functions
init_allocator(void *memory, size_t size)
void *allocater(size_t size)
void free(void *ptr)
void split_block(struct buddy *block, size_t order)
struct buddy  *merge_blocks(struct buddy *block1, struct buddy *block2)
other changes would be to to update memory management routines in xv6 to use buddy allocator functions for allocation and deallocation
```

Question **30**

Correct

Mark 1.00 out of 1.00

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"

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

Your answer is correct.

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

Question **31**

Correct

Mark 1.00 out of 1.00

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

dirlookup	directory ✔
stati	inode ✔
filestat()	file descriptor ✔
bmap	inode ✔
commit	logging ✔
bread	buffer cache ✔
ideintr	disk driver ✔
sys_chdir()	system call ✔
balloc	block allocation on disk ✔
ialloc	inode ✔
skipelem	pathname lookup ✔
namei	pathname lookup ✔

Your answer is correct.

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

Question **32**

Correct

Mark 0.50 out of 0.50

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

(give answer also in decimal)

Answer: 134 ✔

The correct answer is: 134

Question **33**

Correct

Mark 1.00 out of 1.00

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

True	False		
○ ✗	◉ ☑	Named pipes can be used for communication between only "related" processes.	✔
◉ ☑	○ ✗	Both types of pipes are an extension of the idea of "message passing".	✔
◉ ☑	○ ✗	Both types of pipes provide FIFO communication.	✔
○ ✗	◉ ☑	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.	✔
○ ✗	◉ ☑	The pipe() system call can be used to create either a named or un-named pipe.	✔
◉ ☑	○ ✗	Un-named pipes are inherited by a child process from parent.	✔
◉ ☑	○ ✗	Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.	✔
○ ✗	◉ ☑	A named pipe has a name decided by the kernel.	✔

Named pipes can be used for communication between only "related" processes.: False
Both types of pipes are an extension of the idea of "message passing".: True
Both types of pipes provide FIFO communication.: True
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
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
Un-named pipes are inherited by a child process from parent.: True
Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.: True
A named pipe has a name decided by the kernel.: False

Question **34**

Correct

Mark 2.00 out of 2.00

Suppose it is required to add the chown() (without notion of a group, just the notion of owner and others) system call to xv6. Select the changes, from the options given below, which are an absolute must to effect the addition of this system call.

Changes w.r.t. other system calls should be selected if those system calls are necessary for the implementation of the chown() system call.

Must	Not-Must		
○ ✗	◉ ☑	Create an application program su.c which itself is a SUID program, and calls setuid() and setuid() with specified user-ID and then does exec() of a shell.	✔
◉ ☑	○ ✗	Add a uid field representing the owner, inside dinode	✔
○ ✗	◉ ☑	Add few extra files belonging to different users, using mkfs.c	✔
◉ ☑	○ ✗	Add the chown() system call which checks if the user with id "0" is calling this system call and then change the ownership of the on-disk and in-memory inode.	✔
◉ ☑	○ ✗	Add a mode field representing file permissions, inside dinode	✔
○ ✗	◉ ☑	Add the username field to the on disk inode, that is dinode.	✔
○ ✗	◉ ☑	Add a setuid(), seteuid() system call, callable only by the user with ID or EUID equal to "0" to set the new user id of the process	✔
○ ✗	◉ ☑	Mandatorily create a file like "passwd" which maps user-names to user-IDs and modify other utilities (like "ls") to show user-name instead of user-ID for the files.	✔
○ ✗	◉ ☑	Inherit the UID and EUID of the parent in fork()	✔
○ ✗	◉ ☑	Modify exec() to check SUID bit on the executable file and set EUID of the process to UID of the executable	✔
○ ✗	◉ ☑	Add a EUID field in the struct proc representing Effective user ID	✔
○ ✗	◉ ☑	Add a UID field in the struct proc represenging the USER-ID of the user who started this process	✔
◉ ☑	○ ✗	Modify mkfs.c to add owner and permissions to each file created	✔

Create an application program su.c which itself is a SUID program, and calls setuid() and setuid() with specified user-ID and then does exec() of a shell.: Not-Must
Add a uid field representing the owner, inside dinode: Must
Add few extra files belonging to different users, using mkfs.c: Not-Must
Add the chown() system call which checks if the user with id "0" is calling this system call and then change the ownership of the on-disk and in-memory inode.: Must
Add a mode field representing file permissions, inside dinode: Must
Add the username field to the on disk inode, that is dinode.: Not-Must
Add a setuid(), seteuid() system call, callable only by the user with ID or EUID equal to "0" to set the new user id of the process: Not-Must
Mandatorily create a file like "passwd" which maps user-names to user-IDs and modify other utilities (like "ls") to show user-name instead of user-ID for the files.: Not-Must
Inherit the UID and EUID of the parent in fork(): Not-Must
Modify exec() to check SUID bit on the executable file and set EUID of the process to UID of the executable: Not-Must
Add a EUID field in the struct proc representing Effective user ID: Not-Must
Add a UID field in the struct proc represenging the USER-ID of the user who started this process: Not-Must
Modify mkfs.c to add owner and permissions to each file created: Must

Question **35**

Correct

Mark 1.00 out of 1.00

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

"..." means some code.

```
struct proc*
myproc(void) {
...
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
...
}
```

Disable interrupts to avoid another process's pointer being returned ✔

```
void
yield(void)
{
...
release(&ptable.lock);
}
```

Release the lock held by some another process ✔

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

Tell compiler not to reorder memory access beyond this line ✔

```
void
acquire(struct spinlock *lk)
{
  pushcli();
```

Disable interrupts to avoid deadlocks ✔

```
void
acquire(struct spinlock *lk)
{
...
  getcallerpcs(&lk, lk->pcs);
```

Traverse ebp chain to get sequence of instructions followed in functions calls ✔

```
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
panic(char *s)
{
...
  panicked = 1;
```

Ensure that no printing happens on other processors ✔

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

Avoid a self-deadlock ✔

Your answer is correct.

The correct answer is: struct proc*

```
myproc(void) {
...
  pushcli();
  c = mycpu();
  p = c->proc;
  popcli();
...
}
```

→ Disable interrupts to avoid another process's pointer being returned, void

```
yield(void)
{
...
release(&ptable.lock);
}
```

→ Release the lock held by some another process, void

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

→ Tell compiler not to reorder memory access beyond this line, void

```
acquire(struct spinlock *lk)
{
  pushcli();
```

```
  → Disable interrupts to avoid deadlocks, void
acquire(struct spinlock *lk)
{
...
  getcallerpcs(&lk, lk->pcs);



  → Traverse ebp chain to get sequence of instructions followed in functions calls, 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
panic(char *s)
{
...
  panicked = 1; → Ensure that no printing happens on other processors, void
sleep(void *chan, struct spinlock *lk)
{
  ...
  if(lk != &ptable.lock){
    acquire(&ptable.lock);
    release(lk);
  } → Avoid a self-deadlock
```

Question **36**

Partially correct

Mark 1.80 out of 2.00

Following code claims to implement the command

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

Fill in the blanks to make the code work.

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

```
int main(int argc, char *argv[]) {
   int pid1, pid2;
   int pfd[ 2 ✔ ][2];

   pipe( pfd[0] ✔ );
   pid1 = fork() ✔ ;
   if(pid1 != 0) {
      close(pfd[0] [0] ✔ );
      close( 1 ✔ );
      dup( pfd[0][1] ✔ );
      execl("/bin/ls", "/bin/ls", " -1 ✖ ", NULL);
   }
   pipe( pfd[1] ✔ );
   pid2 ✔ = fork();
   if(pid2 == 0) {
      close( pfd[1][1] ✖ ;
      close(0);
      dup( pfd[0][0] ✔ );
      close(pfd[1] [0] ✔ );
      close( 1 ✔ );
      dup( pfd[1][1] ✔ );
      execl("/usr/bin/head", "/usr/bin/head", " -3 ✔ ", NULL);
   } else {
      close(pfd [1][1] ✔ );
      close( 0 ✔ );
      dup( pfd[1][0] ✔ );
      close(pfd [0][0] ✔ );
      execl("/usr/bin/tail", "/usr/bin/tail", " -1 ✔ ", NULL);
  }
}
```

Question **37**

Correct

Mark 1.00 out of 1.00

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

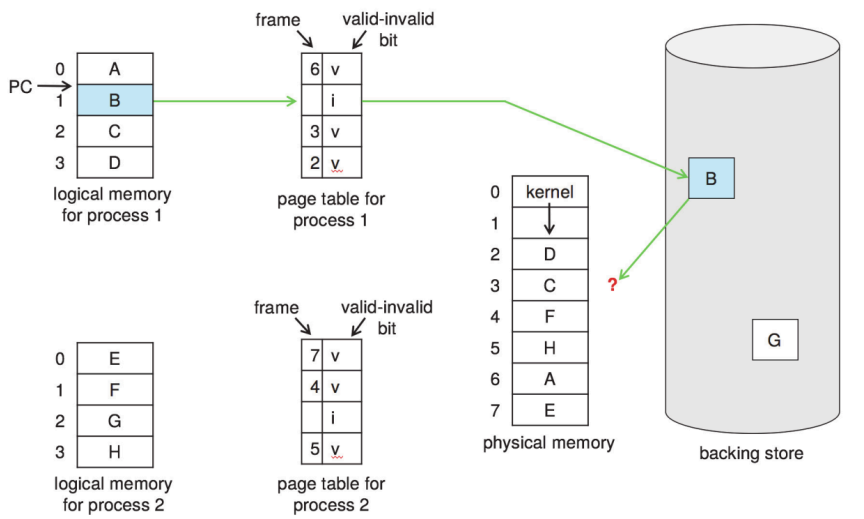


Figure 10.9 Need for page replacement.

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

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

Question **38**

Correct

Mark 0.50 out of 0.50

The approximate number of page frames created by kinit1 is

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

The correct answer is: 3000

Question **39**

Correct

Mark 1.00 out of 1.00

In xv6, The struct context is given as

```
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

Select all the reasons that explain why only these 5 registers are included in the struct context.

- [x] a. esp is not saved in context, because context{} is on stack and it's address is always argument to swtch() ✔
- [ ] b. esp is not saved in context, because it's not part of the context
- [x] c. The segment registers are same across all contexts, hence they need not be saved ✔
- [x] d. eax, ecx, edx are caller save, hence no need to save ✔
- [ ] e. xv6 tries to minimize the size of context to save memory space

Your answer is correct.

The correct answers are: The segment registers are same across all contexts, hence they need not be saved, eax, ecx, edx are caller save, hence no need to save, esp is not saved in context, because context{} is on stack and it's address is always argument to swtch()

Question **40**

Partially correct

Mark 0.17 out of 1.00

Select correct statements about mounting

Select one or more:

- [ ] a. The existing name-space at the mount-point is no longer visible after mounting
- [x] b. The mount point must be a directory ✔
- [ ] c. Mounting deletes all data at the mount-point
- [ ] d. Mounting is attaching a disk-partition with a filesystem on it, into another file system name-space
- [ ] e. The mount point can be a file as well
- [ ] f. On Linuxes mounting can be done only while booting the OS
- [ ] g. Even in operating systems with a pluggable kernel module for file systems, the code for mounting any particular file system must be already present in the operating system system kernel
- [ ] h. Mounting makes all disk partitions available as one name space
- [ ] i. In operating systems with a pluggable kernel module for file systems, the code for mounting a particular file system is provided by the module of that file system.
- [ ] j. It's possible to mount a partition on one computer, into namespace of another computer.

Your answer is partially correct.

You have correctly selected 1.
The correct answers are: Mounting is attaching a disk-partition with a filesystem on it, into another file system name-space, The mount point must be a directory, The existing name-space at the mount-point is no longer visible after mounting, Mounting makes all disk partitions available as one name space, In operating systems with a pluggable kernel module for file systems, the code for mounting a particular file system is provided by the module of that file system., It's possible to mount a partition on one computer, into namespace of another computer.

Question **41**

Correct

Mark 0.50 out of 0.50

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

Your answer is correct.

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 **42**

Correct

Mark 0.50 out of 0.50

Match the pair

Hashed page table	Linear search on collsion done by OS (e.g. SPARC Solaris) typically ✔
Inverted Page table	Linear/Parallel search using page number in page table ✔
Hierarchical Paging	More memory access time per hierarchy ✔

Your answer is correct.

The correct answer is: Hashed page table → Linear search on collsion done by OS (e.g. SPARC Solaris) typically, Inverted Page table → Linear/Parallel search using page number in page table, Hierarchical Paging → More memory access time per hierarchy

Question **43**

Partially correct

Mark 0.40 out of 0.50

Select all the complications added due to concurrent processing in operating systems.

Select one or more:

- [x] a. Context of each process needs to be restored when it's scheduled ✔
- [x] b. It requires a timer interrupt in hardware ✔
- [x] c. Context of each process needs to be saved when it's interrupted by timer ✔
- [ ] d. OS needs to decide whether it's own code is interruptible or not
- [x] e. Scheduling becomes more time consuming due to increased number of processes ✔

Your answer is partially correct.

You have correctly selected 4.

The correct answers are: It requires a timer interrupt in hardware, Scheduling becomes more time consuming due to increased number of processes, OS needs to decide whether it's own code is interruptible or not, Context of each process needs to be saved when it's interrupted by timer, Context of each process needs to be restored when it's scheduled

Question **44**

Correct

Mark 1.00 out of 1.00

Mark statements as T/F

All statements are in the context of preventing deadlocks.

True	False		
○ ✗	◉ ☑	If a resource allocation graph contains a cycle then there is a guarantee of 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	✔
◉ ☑	○ ✗	Circular wait is avoided by enforcing a lock ordering	✔
◉ ☑	○ ✗	If a resource allocation graph contains a cycle then there is a possibility of a deadlock	✔
◉ ☑	○ ✗	Deadlock is not possible if any of these conditions is not met: Mutual exclusion, hold and wait, no pre-emption, circular wait.	✔
◉ ☑	○ ✗	The lock ordering to be followed to avoid circular wait is a protocol to be followed by programmers.	✔

If a resource allocation graph contains a cycle then there is a guarantee of a deadlock: False
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
Circular wait is avoided by enforcing a lock ordering: True
If a resource allocation graph contains a cycle then there is a possibility of a deadlock: True
Deadlock is not possible if any of these conditions is not met: Mutual exclusion, hold and wait, no pre-emption, circular wait.: True
The lock ordering to be followed to avoid circular wait is a protocol to be followed by programmers.: True

Question **45**

Correct

Mark 1.00 out of 1.00

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

```
void
panic(char *s)
{
...
  panicked = 1;
```

Ensure that no printing happens on other processors ✔

```
void
acquire(struct spinlock *lk)
{
...
  getcallerpcs(&lk, lk->pcs);
```

Traverse ebp chain to get sequence of instructions followed in functions calls ✔

```
void
yield(void)
{
...
release(&ptable.lock);
}
```

Release the lock held by some another process ✔

Your answer is correct.

The correct answer is: void
panic(char *s)
{
...
  panicked = 1; → Ensure that no printing happens on other processors, void
acquire(struct spinlock *lk)
{
...
  getcallerpcs(&lk, lk->pcs); → Traverse ebp chain to get sequence of instructions followed in functions calls, void
yield(void)
{
...
release(&ptable.lock);
} → Release the lock held by some another process

Question **46**

Correct

Mark 1.00 out of 1.00

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

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

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

Question **47**

Correct

Mark 1.00 out of 1.00

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

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

Program 1

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

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

Program 2

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

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

Select all the correct statements about the programs

Select one or more:

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

Your answer is correct.

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

Question **48**

Correct

Mark 1.00 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 0 0 0 0 1 1 ✔

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

Question **49**

Correct

Mark 0.50 out of 0.50

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)

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

Your answer is correct.

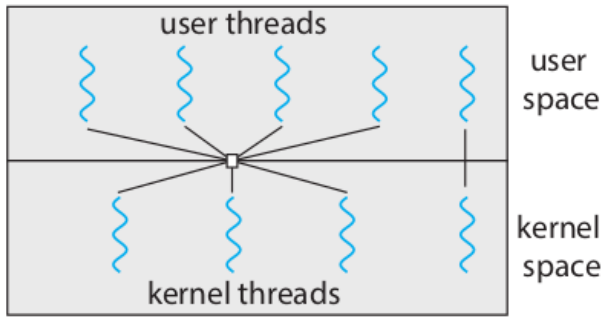
The correct answer is: Indexed Allocation → Overhead of reading metadata blocks, Linked allocation → Too many seeks, Continuous allocation → need for compaction

Question **50**

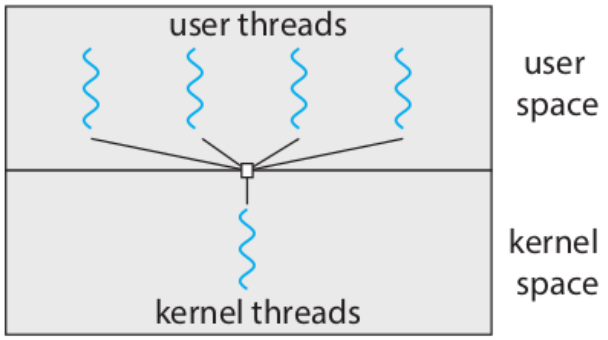
Correct

Mark 0.50 out of 0.50

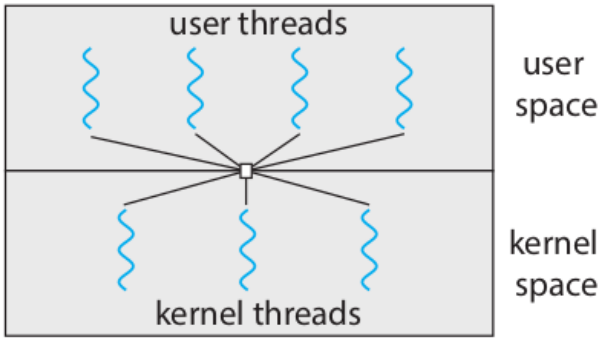
Match the diagram with the threading model



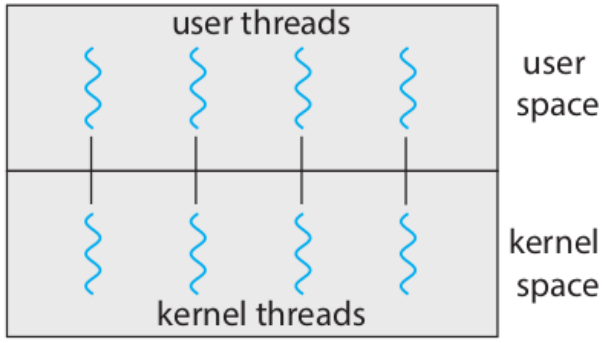
Two-Level ✔



Many-One ✔



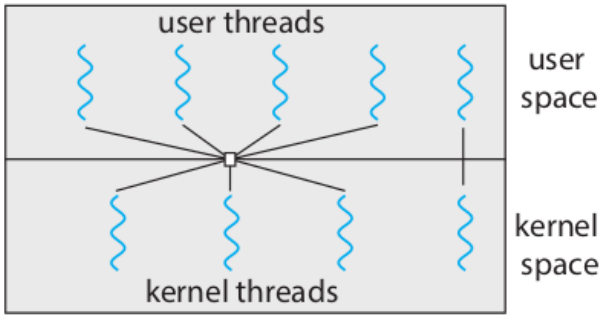
Many-Many ✔



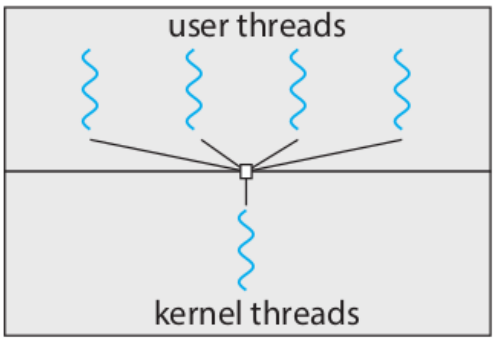
One-One ✔

Your answer is correct.

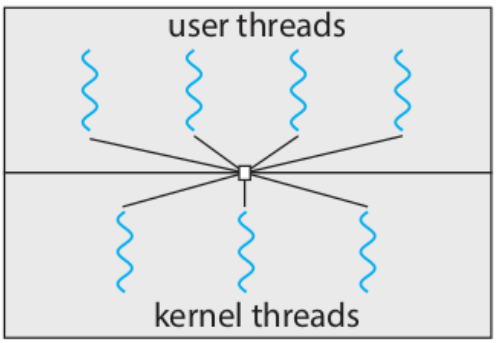
The correct answer is:



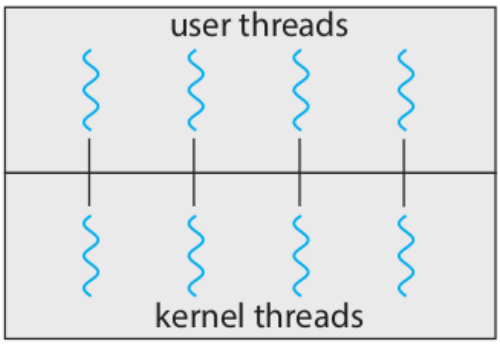
→ Two-Level,



→ Many-One,



→ Many-Many,



→ One-One

Question **51**

Correct

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

Your answer is correct.

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

Question **52**

Correct

Mark 1.00 out of 1.00

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

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

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

Question **53**

Partially correct

Mark 0.25 out of 1.00

Select the correct statements about directory entries in ext2:

- [ ] a. Maximum size of file name is 255
- [x] b. It is possible that there is a hole between two directory entries. ✔
- [ ] c. When the first entry is removed, a compaction is done
- [x] d. The size of a directory is = sum of sizes of all directory entries ✖
- [x] e. The rec_len is always = 8 + name_len + number to round off name_len to multiple of 4 ✖
- [x] f. The rec_len of the last entry is set to cover the directory's actual size. ✔
- [ ] g. The name_len is always a multiple of 4, by adding extra character to the actual name
- [x] h. The name_len is always stored in multiple of 4 bytes, suffixing zeroes if required. ✔

Your answer is partially correct.

You have selected too many options.
The correct answers are: It is possible that there is a hole between two directory entries., The rec_len of the last entry is set to cover the directory's actual size., The name_len is always stored in multiple of 4 bytes, suffixing zeroes if required., Maximum size of file name is 255

Question **54**

Correct

Mark 0.50 out of 0.50

If one thread opens a file with read privileges then, without opening the file

Select one:

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

Your answer is correct.

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

Question **55**

Correct

Mark 0.50 out of 0.50

Match the pairs of which action is taken by whom

processor — detect memory violations ✔

kernel — setup the MMU hardware ✔

compiler — generate code using virtual addresses ✔

The correct answer is: processor → detect memory violations, kernel → setup the MMU hardware, compiler → generate code using virtual addresses

Question **56**

Correct

Mark 0.50 out of 0.50

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

- [ ] a. No data structure of kernel should be pageable
- [x] b. The kernel's own page tables should not be pageable ✔
- [x] c. The kernel must have some dedicated frames for it's own work ✔
- [ ] d. No part of kernel code should be pageable.
- [x] e. The page fault handler should not be pageable ✔
- [x] f. The disk driver and disk interrupt handler should not 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 **57**

Incorrect

Mark 0.00 out of 1.00

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

Memory access time = 140 ns

Average page fault service time = 10 ms

Page fault rate = 0.9

Answer: 15 ✖

The correct answer is: 9000014.00

Question **58**

Correct

Mark 2.00 out of 2.00

For Virtual File System to work, which of the following changes are required to be done to an existing OS code (e.g. xv6)?

- a. The file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup. ✔
- b. A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/" ✔
- c. Each open() needs to copy the function pointers from the inode of the parent directory into the inode of the child (if not already done), unless it's traversing a mount point. (This may be done as part of lookup() which is called by open()) ✔
- d. The operating system in-memory inode needs to be a generic-inode representing "inode" like data structure across multiple file systems. ✔
- e. The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories ✔
- f. The filesystem related system calls (e.g. read, write) need to invoke the file system specific functions (e.g. ext2_read, ext2_write, ntfs_read, ntfs_write) using function pointers. ✔
- g. Each file-system writer needs to provide the set of function pointers for VFS, and these function pointers need to be setup in generic inode of "/" of that file system during mount() ✔
- h. The generic inode needs to have a field representing if this inode is a mount point and also to refer/point to the root of the mounted file system's inode. ✔

The correct answers are: A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/", The filesystem related system calls (e.g. read, write) need to invoke the file system specific functions (e.g. ext2_read, ext2_write, ntfs_read, ntfs_write) using function pointers., The file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup., The operating system in-memory inode needs to be a generic-inode representing "inode" like data structure across multiple file systems., The generic inode needs to have a field representing if this inode is a mount point and also to refer/point to the root of the mounted file system's inode., The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories, Each file-system writer needs to provide the set of function pointers for VFS, and these function pointers need to be setup in generic inode of "/" of that file system during mount(), Each open() needs to copy the function pointers from the inode of the parent directory into the inode of the child (if not already done), unless it's traversing a mount point. (This may be done as part of lookup() which is called by open())

Question **59**

Correct

Mark 0.50 out of 0.50

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);
    execl("/usr/bin/echo", "/usr/bin/echo", "one", 0);
}
```

Answer: hi one ✔

The correct answer is: hi one

Question **60**

Correct

Mark 0.50 out of 0.50

Map the technique with it's feature/problem

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

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

Question **61**

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)

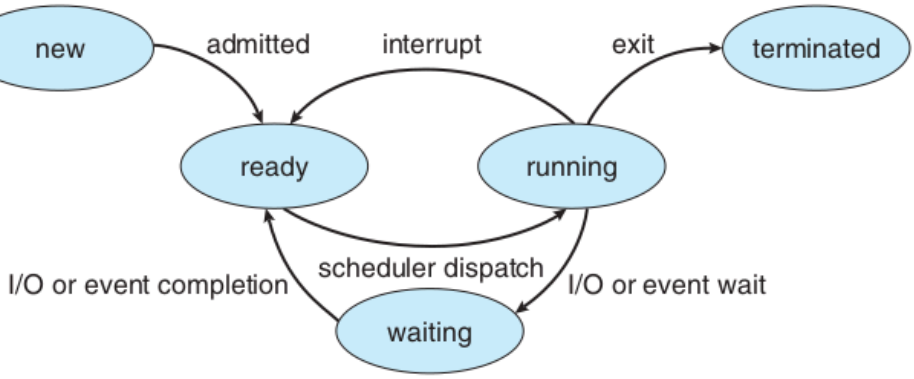


Figure 3.2 Diagram of process state.

True	False		
◉☑	○✖	A RUNNING process becomes READY only on timer interrupt	✔
◉☑	○✖	A process in RUNNING state becomes WAITING if resource is not available immediately	✔
○✖	◉☑	Some forked processes may terminate normally without becoming ZOMBIE.	✔
◉☑	○✖	A READY process becomes RUNNING only when the scheduler selects it	✔
◉✖	○☑	A process in RUNNING state becomes WAITING if the scheuler decides that it should WAIT for it's turn	✖

A RUNNING process becomes READY only on timer interrupt: True
A process in RUNNING state becomes WAITING if resource is not available immediately: True
Some forked processes may terminate normally without becoming ZOMBIE.: False
A READY process becomes RUNNING only when the scheduler selects it: True
A process in RUNNING state becomes WAITING if the scheuler decides that it should WAIT for it's turn: False

Question **62**

Correct

Mark 2.00 out of 2.00

The following processes are being scheduled using a pre-emptive, priority-based, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	8	15	0
P_2	3	20	0
P_3	4	20	20
P_4	4	20	25
P_5	5	5	45
P_6	5	15	55

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the currently highest priority process for it's full duration, unless it gets pre-empted by newly arriving higher priority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is pre-empted by a higher-priority process, the pre-empted process is placed at the front of the queue of same priority processes (so that it's turn continues when higher priority process is over).

The order in which the processes get scheduled is (write your answer without a space, e.g. P1,P2,P3,P4,P5) :

P1,P2,P3,P4,P5,P4,P6,P4,P2 ✔

The turn-around time for the process P2 is: 95 ✔

The turn-around time for the process P6 is: 15 ✔

The process P5 finishes at time unit: 50 ✔

0-15 P1

15-20 P2

20-40 P3

40-45 P4

45-50 P5

50-55 P4

55-70 P6

70-80 P4

80-95 P2

--

P2 turnaround time = 95 - 15 = 80

Question **63**

Correct

Mark 1.00 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 table of page faulted process is updated
12. ✔ Page faulted process is moved to ready-queue

Your answer is correct.

◄ Homework questions: Basics of MM, xv6 booting

Jump to...

Link to UBuntu VM for this course (VDI File, 15+GB) ►