

Question **14**

Not yet answered

Marked out of 3.00

List down all changes required to xv6 code, in order to add the system call chown().

Time left 1:53:37

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.

Paragraph

(a) int chown (char * path , char *ownername);
(d)there is no need of new data structures or changes to the existing data structure
(e)by using this system call ,user to change the owner if the file
(f)_trychown\ at the end of UPROGS

Path: p

◀ Random Quiz - 6 (xv6 file system)

Jump to...

Homework questions: Basics of MM, xv6 booting ►

Started on Wednesday, 19 April 2023, 6:16 PM

State Finished

Completed on Wednesday, 19 April 2023, 8:31 PM

Time taken 2 hours 14 mins

Overdue 14 mins 47 secs

Grade **14.08** out of 30.00 (**46.92%**)

Question **1**

Incorrect

Mark 0.00 out of 1.00

Select all correct statements about file system recovery (without journaling) programs e.g. fsck

Select one or more:

- a. Recovery programs are needed only if the file system has a delayed-write policy. ✓
- b. Recovery is possible due to redundancy in file system data structures
- c. It is possible to lose data as part of recovery ✓
- d. A recovery program, most typically, builds the file system data structure and checks for inconsistencies ✓
- e. Recovery programs recalculate most of the metadata summaries (e.g. free inode count) ✓
- f. They are used to recover deleted files ✗
- g. They may take very long time to execute ✓
- h. Even with a write-through policy, it is possible to need a recovery program.
- i. They can make changes to the on-disk file system

Your answer is incorrect.

The correct answers are: Recovery is possible due to redundancy in file system data structures, A recovery program, most typically, builds the file system data structure and checks for inconsistencies, It is possible to lose data as part of recovery, They may take very long time to execute, They can make changes to the on-disk file system, Recovery programs recalculate most of the metadata summaries (e.g. free inode count), Recovery programs are needed only if the file system has a delayed-write policy., Even with a write-through policy, it is possible to need a recovery program.

Question 2

Partially correct

Mark 0.70 out of 1.00

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

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

During thrashing the CPU is under-utilised as most time is spent in I/O: True

Thrashing occurs when the total size of all process's locality exceeds total memory size.: True

Thrashing can be limited if local replacement is used.: True

Thrashing occurs because some process is doing lot of disk I/O.: False

The working set model is an attempt at approximating the locality of a process.: True

Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.: True

Thrashing can occur even if entire memory is not in use.: False

mmap() solves the problem of thrashing.: False

Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.: False

Question 3

Partially correct

Mark 0.25 out of 1.00

Match each suggested semaphore implementation (discussed in class)

with the problems that it faces

```
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) {  
        spinunlock(&(s->sl));  
        spinlock(&(s->sl));  
    }  
    (s->val)--;  
    spinunlock(&(s->sl));  
}
```

too much spinning, bounded wait not guaranteed ✓

```
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));  
}
```

blocks holding a spinlock ✗

```

struct semaphore {
    int val;
    spinlock lk;
    list l;
};

sem_init(semaphore *s, int initval) {
    s->val = initval;
    s->s1 = 0;
}

block(semaphore *s) {
    listappend(s->l, current);
    spinunlock(&(s->s1));
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->s1));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->s1));
}

signal(semaphore *s) {
    spinlock(*(s->s1));
    (s->val)++;
    x = dequeue(s->s1) and enqueue(readyq, x);
    spinunlock(*(s->s1));
}

```

too much spinning, bounded wait not guaranteed



```

struct semaphore {
    int val;
    spinlock lk;
    list l;
};

sem_init(semaphore *s, int initval) {
    s->val = initval;
    s->s1 = 0;
}

block(semaphore *s) {
    listappend(s->l, current);
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->s1));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->s1));
}

```

deadlock



Your answer is partially correct.

You have correctly selected 1.

The correct answer is:

```

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) {
        spinunlock(&(s->sl));
        spinlock(&(s->sl));
    }
    (s->val)--;
    spinunlock(&(s->sl));
}

```

→ too much spinning, bounded wait not guaranteed,

```

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));
}

```

→ deadlock,

```

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(semaphore *s) {
    spinlock(*(&(s->sl)));
    (s->val)++;
    x = dequeue(s->sl) and enqueue(readyq, x);
    spinunlock(*(&(s->sl)));
}

```

→ not holding lock after unblock,

```
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);
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->sl));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->sl));
}
```

→ blocks holding a spinlock

Question 4

Partially correct

Mark 0.88 out of 1.00

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

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

The arguments to system call originally reside on process stack.: True

String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer: True

The arguments are accessed in the kernel code using esp on the trapframe.: True

Integer arguments are stored in eax, ebx, ecx, etc. registers: False

Integer arguments are copied from user memory to kernel memory using argint(): True

String arguments are first copied to trapframe and then from trapframe to kernel's other variables.: False

The functions like argint(), argstr() make the system call arguments available in the kernel.: True

The arguments to system call are copied to kernel stack in trapasm.S: False

Question 5

Correct

Mark 1.00 out of 1.00

Map the technique with its feature/problem

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

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

Question 6

Partially correct

Mark 0.10 out of 2.00

Compare paging with demand paging and select the correct statements.

Select one or more:

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

Your answer is partially correct.

You have correctly selected 5.

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

Question **7**

Correct

Mark 1.00 out of 1.00

Given that the memory access time is 150 ns, probability of a page fault is 0.9 and page fault handling time is 6 ms,
The effective memory access time in nanoseconds is:

Answer: 5400015 ✓

The correct answer is: 5400015.00

Question **8**

Correct

Mark 1.00 out of 1.00

Assuming a 8- KB page size, what is the page numbers for the address 803160 reference in decimal :
(give answer also in decimal)

Answer: 98 ✓

The correct answer is: 98

Question 9

Partially correct

Mark 1.50 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. 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() ✓
- b. A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/" ✓
- c. The operating system in-memory inode needs to be a generic-inode representing "inode" like data structure across multiple file systems.
- d. 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. ✓
- e. 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. ✓
- f. The file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup.
- g. 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()) ✓
- h. The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories ✓

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 10

Partially correct

Mark 0.33 out of 1.00

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

"..." means some code.

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

Release the lock held by some another process



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

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



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

Disable interrupts to avoid another process's pointer being returned



Your answer is partially correct.

You have correctly selected 1.

The correct answer is:

```
void  
yield(void)  
{  
...  
release(&ptable.lock);  
} → Release the lock held by some another process, 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
```

Question 11

Partially correct

Mark 0.75 out of 1.00

Select all correct statements about journalling (logging) in file systems like ext3

Select one or more:

- a. A different device driver is always needed to access the journal
- b. The purpose of journal is to speed up file system recovery ✓
- c. Most typically a transaction in journal is recorded atomically (full or none) ✓
- d. Journals are often stored circularly
- e. Journals must be maintained on the same device that hosts the file system
- f. the journal contains a summary of all changes made as part of a single transaction ✓
- g. Journal is hosted in the same device that hosts the swap space

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: The purpose of journal is to speed up file system recovery, the journal contains a summary of all changes made as part of a single transaction, Most typically a transaction in journal is recorded atomically (full or none), Journals are often stored circularly

Question **12**

Partially correct

Mark 1.00 out of 2.00

Select all the correct statements about synchronization primitives.

Select one or more:

- a. Thread that is going to block should not be holding any spinlock
- b. Semaphores can be used for synchronization scenarios like ordered execution ✓
- c. Mutexes can be implemented using spinlock ✓
- d. Blocking means one process passing over control to another process
- e. Semaphores are always a good substitute for spinlocks
- f. Blocking means moving the process to a wait queue and spinning
- g. Blocking means moving the process to a wait queue and calling scheduler
- h. All synchronization primitives are implemented essentially with some hardware assistance.
- i. Spinlocks are good for multiprocessor scenarios, for small critical sections
- j. Mutexes can be implemented without any hardware assistance
- k. Spinlocks consume CPU time ✓
- l. Mutexes can be implemented using blocking and wakeup ✓

Your answer is partially correct.

You have correctly selected 4.

The correct answers are: Spinlocks are good for multiprocessor scenarios, for small critical sections, Spinlocks consume CPU time, Semaphores can be used for synchronization scenarios like ordered execution, Mutexes can be implemented using spinlock, Mutexes can be implemented using blocking and wakeup, Thread that is going to block should not be holding any spinlock, Blocking means moving the process to a wait queue and calling scheduler, All synchronization primitives are implemented essentially with some hardware assistance.

Question 13

Partially correct

Mark 0.50 out of 2.00

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

"..." means some code.

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

Ensure that no printing happens on other processors



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

Disable interrupts to avoid another process's pointer being returned



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

Avoid a self-deadlock



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

Disable interrupts to avoid deadlocks



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

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



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

Release the lock held by some another process



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

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



Your answer is partially correct.

You have correctly selected 2.

The correct answer is: void

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

→ Release the lock held by some another process, **struct proc***

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

→ Disable interrupts to avoid another process's pointer being returned, **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  
acquire(struct spinlock *lk)  
{  
...  
_sync_synchronize();
```

```
→ Tell compiler not to reorder memory access beyond this line, void  
acquire(struct spinlock *lk)  
{  
...  
getcallerpcs(&lk, lk->pcs);
```

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

Question 14

Partially correct

Mark 0.86 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	
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume may span across multiple physical partitions
<input type="radio"/> ✓	<input type="radio"/> X	A logical volume may span across multiple physical volumes
<input checked="" type="radio"/>	<input type="radio"/> X	A physical partition should be initialized as a physical volume, before it can be used by volume manager.
<input checked="" type="radio"/>	<input type="radio"/> X	A volume group consists of multiple physical volumes
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume can be extended in size but upto the size of volume group
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager stores additional metadata on the physical disk partitions

A logical volume may span across multiple physical partitions: True

A logical volume may span across multiple physical volumes: True

A physical partition should be initialized as a physical volume, before it can be used by volume manager.: True

A volume group consists of multiple physical volumes: True

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

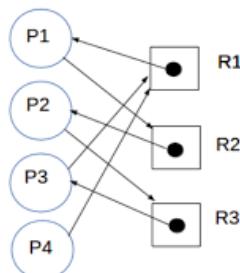
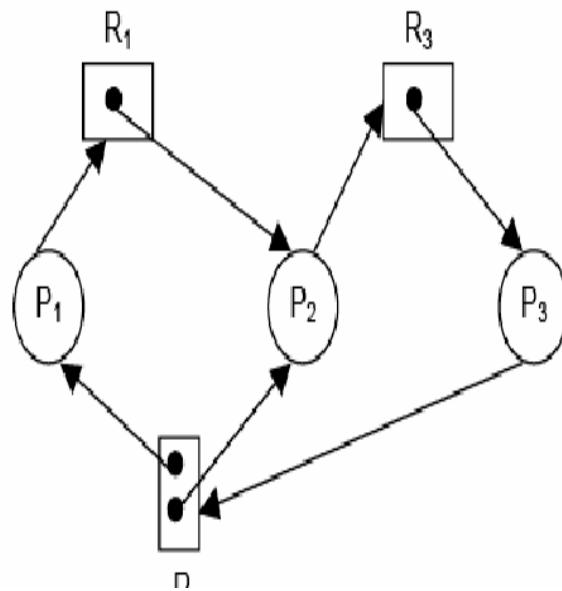
The volume manager stores additional metadata on the physical disk partitions: True

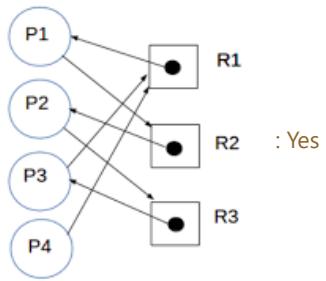
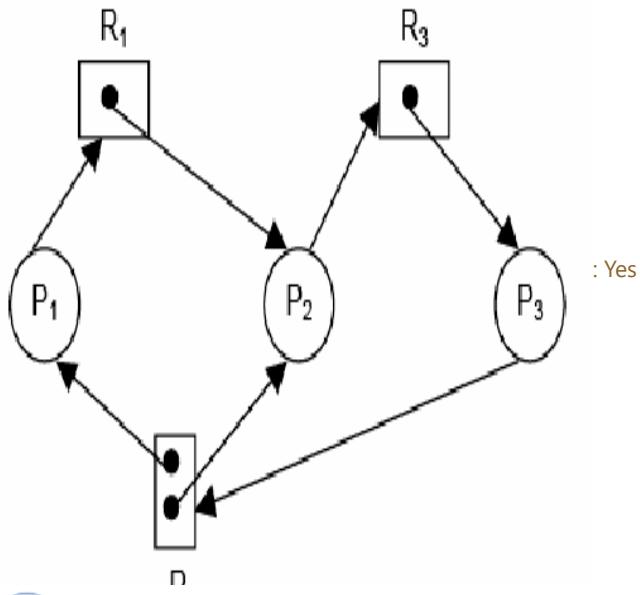
Question 15

Partially correct

Mark 0.50 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**



Question **16**

Partially correct

Mark 0.80 out of 1.00

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

Select one or more:

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 17

Incorrect

Mark 0.00 out of 1.00

Match the code with its functionality

S1 = 0; S2 = 0;

P2:

Statement1;

Signal(S2);

P1:

Wait(S2);

Statementn2;

Signal(S1);

Execution order P1, P2, P3



P3:

Wait(S1);

Statement S3;

S = 0

P1:

Statement1;

Signal(S)

Execution order P3, P2, P1



P2:

Wait(S)

Statment2;

S = 5

Wait(S)

Critical Section

Execution order P2, then P1



Signal(S)

S = 1

Wait(S)

Critical Section

Counting semaphore



Signal(S);

Your answer is incorrect.

The correct answer is: S1 = 0; S2 = 0;

P2:

Statement1;

Signal(S2);

P1:

Wait(S2);

Statemetn2;

Signal(S1);

P3:

Wait(S1);

Statement S3; → Execution order P2, P1, P3, S = 0

P1:

Statement1;

Signal(S)

P2:

Wait(S)

Statement2; → Execution order P1, then P2, S = 5

Wait(S)

Critical Section

Signal(S) → Counting semaphore, S = 1

Wait(S)

Critical Section

Signal(S); → Binary Semaphore for mutual exclusion

Question **18**

Not answered

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.

Question 19

Correct

Mark 1.00 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.

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

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 20

Incorrect

Mark 0.00 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: 11001110

✗

The correct answer is: 11011110

Question **21**

Complete

Mark 0.25 out of 3.00

List down all changes required to xv6 code, in order to add the system call chown().

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.
-
- (a) int chown(char* path, char *owner_name);
 - d) there is no need to change the data structure and no need to add new data structure
 - (e) allow user to change the owner of the file
 - (f) _chown\

Comment:

Question 22

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 23

Partially correct

Mark 0.67 out of 1.00

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

Consider each request as first request.

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

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

[◀ Random Quiz - 6 \(xv6 file system\)](#)

Jump to...

[Homework questions: Basics of MM, xv6 booting ►](#)

Started on Wednesday, 19 April 2023, 8:46 PM

State Finished

Completed on Wednesday, 19 April 2023, 9:26 PM

Time taken 39 mins 31 secs

Grade 15.60 out of 30.00 (51.99%)

Question 1

Incorrect

Mark 0.00 out of 2.00

Select all the correct statements about synchronization primitives.

Select one or more:

- a. Blocking means moving the process to a wait queue and calling scheduler
- b. Blocking means moving the process to a wait queue and spinning ✗
- c. Semaphores are always a good substitute for spinlocks ✗
- d. Spinlocks are good for multiprocessor scenarios, for small critical sections
- e. All synchronization primitives are implemented essentially with some hardware assistance.
- f. Blocking means one process passing over control to another process ✗
- g. Semaphores can be used for synchronization scenarios like ordered execution ✓
- h. Mutexes can be implemented without any hardware assistance
- i. Spinlocks consume CPU time
- j. Mutexes can be implemented using spinlock
- k. Thread that is going to block should not be holding any spinlock
- l. Mutexes can be implemented using blocking and wakeup

Your answer is incorrect.

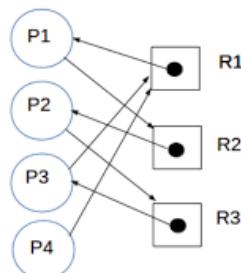
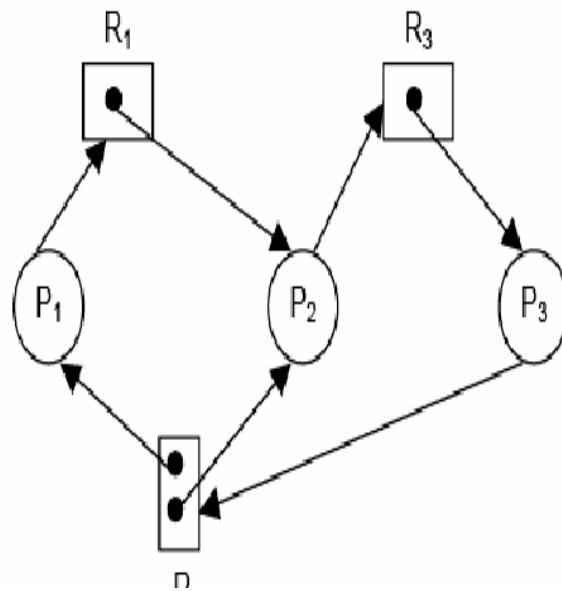
The correct answers are: Spinlocks are good for multiprocessor scenarios, for small critical sections, Spinlocks consume CPU time, Semaphores can be used for synchronization scenarios like ordered execution, Mutexes can be implemented using spinlock, Mutexes can be implemented using blocking and wakeup, Thread that is going to block should not be holding any spinlock, Blocking means moving the process to a wait queue and calling scheduler, All synchronization primitives are implemented essentially with some hardware assistance.

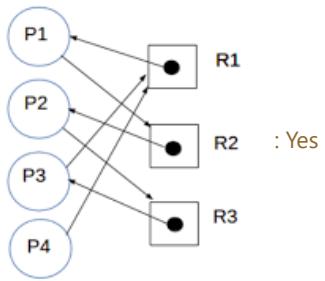
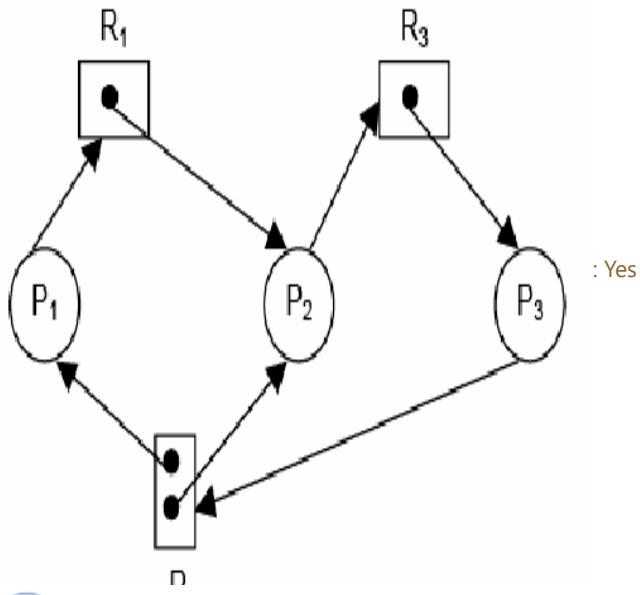
Question 2

Partially correct

Mark 0.50 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**



Question 3

Partially correct

Mark 0.25 out of 1.00

Match each suggested semaphore implementation (discussed in class)

with the problems that it faces

```
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(semaphore *s) {
    spinlock(*(s->sl));
    (s->val)++;
    x = dequeue(s->sl) and enqueue(readyq, x);
    spinunlock(*(s->sl));
}
```

deadlock



```
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);
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->sl));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->sl));
}
```

not holding lock after unblock



```
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) {
        spinunlock(&(s->sl));
        spinlock(&(s->sl));
    }
    (s->val)--;
    spinunlock(&(s->sl));
}
```

too much spinning, bounded wait not guaranteed



```
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));
}
```

blocks holding a spinlock



Your answer is partially correct.

You have correctly selected 1.

The correct answer is:

```

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(semaphore *s) {
    spinlock(*(s->sl));
    (s->val)++;
    x = dequeue(s->sl) and enqueue(readyq, x);
    spinunlock(*(s->sl));
}

```

→ not holding lock after unblock,

```

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);
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->sl));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->sl));
}

```

→ blocks holding a spinlock,

```

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) {
        spinunlock(&(s->sl));
        spinlock(&(s->sl));
    }
    (s->val)--;
    spinunlock(&(s->sl));
}

```

→ too much spinning, bounded wait not guaranteed,

```

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));
}

```

→ deadlock

Question 4

Partially correct

Mark 0.75 out of 1.00

Match the code with its functionality

S = 1
Wait(S)
Critical Section Binary Semaphore for mutual exclusion ✓
Signal(S);

S = 0
P1:
Statement1;
Signal(S) Execution order P1, then P2 ✓
P2:

Wait(S)
Statement2;
S = 5
Wait(S)
Critical Section Execution order P1, P2, P3 ✗
Signal(S)

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

P1:
Wait(S2); Execution order P2, P1, P3 ✓
Statement2;
Signal(S1);

P3:
Wait(S1);
Statement S3;

Your answer is partially correct.

You have correctly selected 3.

The correct answer is: S = 1

Wait(S)
Critical Section
Signal(S); → Binary Semaphore for mutual exclusion, S = 0
P1:
Statement1;
Signal(S)

P2:
Wait(S)
Statement2; → Execution order P1, then P2, S = 5
Wait(S)
Critical Section
Signal(S) → Counting semaphore, S1 = 0; S2 = 0;

P2:

Statement1;

Signal(S2);

P1:

Wait(S2);

Statemetn2;

Signal(S1);

P3:

Wait(S1);

Statement S3; → Execution order P2, P1, P3

Question 5

Partially correct

Mark 1.71 out of 2.00

Compare paging with demand paging and select the correct statements.

Select one or more:

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

Your answer is partially correct.

You have correctly selected 6.

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 6

Partially correct

Mark 0.33 out of 1.00

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

"..." means some code.

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

Disable interrupts to avoid another process's pointer being returned



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

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



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

Release the lock held by some another process



Your answer is partially correct.

You have correctly selected 1.

The correct answer is:

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

Question 7

Correct

Mark 1.00 out of 1.00

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

Consider each request as first request.

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

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

Question 8

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	
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume may span across multiple physical partitions
<input checked="" type="radio"/>	<input type="radio"/> X	A volume group consists of multiple physical volumes
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume may span across multiple physical volumes
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume can be extended in size but upto the size of volume group
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.
<input checked="" type="radio"/>	<input type="radio"/> X	A physical partition should be initialized as a physical volume, before it can be used by volume manager.
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager stores additional metadata on the physical disk partitions

A logical volume may span across multiple physical partitions: True

A volume group consists of multiple physical volumes: True

A logical volume may span across multiple physical volumes: True

A logical volume can be extended in size but upto the size of volume group: True

The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.: True

A physical partition should be initialized as a physical volume, before it can be used by volume manager.: True

The volume manager stores additional metadata on the physical disk partitions: True

Question **9**

Incorrect

Mark 0.00 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: 11001101



The correct answer is: 11011110

Question **10**

Not answered

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.

Question 11

Partially correct

Mark 0.80 out of 1.00

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

Select one or more:

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 12

Partially correct

Mark 0.50 out of 1.00

Select all correct statements about journalling (logging) in file systems like ext3

Select one or more:

- a. the journal contains a summary of all changes made as part of a single transaction ✓
- b. Journals are often stored circularly
- c. Journals must be maintained on the same device that hosts the file system
- d. The purpose of journal is to speed up file system recovery ✓
- e. Most typically a transaction in journal is recorded atomically (full or none)
- f. Journal is hosted in the same device that hosts the swap space
- g. A different device driver is always needed to access the journal

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: The purpose of journal is to speed up file system recovery, the journal contains a summary of all changes made as part of a single transaction, Most typically a transaction in journal is recorded atomically (full or none), Journals are often stored circularly

Question 13

Correct

Mark 1.00 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.

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

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 **14**

Complete

Mark 0.25 out of 3.00

List down all changes required to xv6 code, in order to add the system call chown().

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.
 - (a) int chown (char * path , char *ownername);
 - (d)there is no need of new data structures or changes to the existing data structure
 - (e)by using this system call ,user to change the owner if the file
 - (f)_trychown\ at the end of UPROGS

Comment:

Question 15

Partially correct

Mark 1.00 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 operating system in-memory inode needs to be a generic-inode representing "inode" like data structure across multiple file systems.
- b. 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. ✓
- c. 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() ✓
- d. A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/" ✓
- e. 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()) ✓
- f. The file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup.
- g. The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories
- h. 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 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 16

Partially correct

Mark 0.75 out of 1.00

Map the technique with its feature/problem

dynamic linking	small executable file	✓
static loading	wastage of physical memory	✓
static linking	large executable file	✓
dynamic loading	small executable file	✗

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

Question 17

Correct

Mark 1.00 out of 1.00

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

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

Thrashing can be limited if local replacement is used.: True

Thrashing occurs because some process is doing lot of disk I/O.: False

Thrashing occurs when the total size of all process's locality exceeds total memory size.: True

During thrashing the CPU is under-utilised as most time is spent in I/O: True

Thrashing is particular to demand paging systems, and does not apply to pure paging systems.: True

Thrashing can occur even if entire memory is not in use.: False

Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.: True

Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.: False

The working set model is an attempt at approximating the locality of a process.: True

mmap() solves the problem of thrashing.: False

Question **18**

Partially correct

Mark 0.75 out of 2.00

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();
...
}
```

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;
}

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

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

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



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

Release the lock held by some another process



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



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)
{
...
getcallerpcs(&lk, lk->pcs);
```

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



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

Release the lock held by some another process



Your answer is partially correct.

You have correctly selected 3.

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, **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
panic(char *s)
{
...
panicked = 1; → Ensure that no printing happens on other processors, void
acquire(struct spinlock *lk)
{
    pushcli();
    → Disable interrupts to avoid deadlocks, void
acquire(struct spinlock *lk)
{
...
__sync_synchronize();
```

```
→ Tell compiler not to reorder memory access beyond this line, 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 **19**

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: ✓

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 20

Incorrect

Mark 0.00 out of 1.00

Select all correct statements about file system recovery (without journaling) programs e.g. fsck

Select one or more:

- a. Recovery programs recalculate most of the metadata summaries (e.g. free inode count) ✓
- b. It is possible to lose data as part of recovery ✓
- c. Even with a write-through policy, it is possible to need a recovery program. ✓
- d. They can make changes to the on-disk file system ✓
- e. A recovery program, most typically, builds the file system data structure and checks for inconsistencies ✓
- f. They may take very long time to execute ✓
- g. They are used to recover deleted files ✗
- h. Recovery programs are needed only if the file system has a delayed-write policy.
- i. Recovery is possible due to redundancy in file system data structures

Your answer is incorrect.

The correct answers are: Recovery is possible due to redundancy in file system data structures, A recovery program, most typically, builds the file system data structure and checks for inconsistencies, It is possible to lose data as part of recovery, They may take very long time to execute, They can make changes to the on-disk file system, Recovery programs recalculate most of the metadata summaries (e.g. free inode count), Recovery programs are needed only if the file system has a delayed-write policy., Even with a write-through policy, it is possible to need a recovery program.

Question 21

Correct

Mark 1.00 out of 1.00

Given that the memory access time is 150 ns, probability of a page fault is 0.8 and page fault handling time is 6 ms,
The effective memory access time in nanoseconds is:

Answer: 4800030 ✓

The correct answer is: 4800030.00

Question **22**

Correct

Mark 1.00 out of 1.00

Assuming a 8- KB page size, what is the page numbers for the address 1005699 reference in decimal :

(give answer also in decimal)

Answer: 123



The correct answer is: 123

Question 23

Correct

Mark 1.00 out of 1.00

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

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

String arguments are first copied to trapframe and then from trapframe to kernel's other variables.: False

The arguments to system call are copied to kernel stack in trapasm.S: False

The arguments are accessed in the kernel code using esp on the trapframe.: True

String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer: True

Integer arguments are copied from user memory to kernel memory using argint(): True

The arguments to system call originally reside on process stack.: True

Integer arguments are stored in eax, ebx, ecx, etc. registers: False

The functions like argint(), argstr() make the system call arguments available in the kernel.: True

[◀ Random Quiz - 6 \(xv6 file system\)](#)

Jump to...

[Homework questions: Basics of MM, xv6 booting ►](#)

Started on Thursday, 18 March 2021, 2:46 PM

State Finished

Completed on Thursday, 18 March 2021, 3:50 PM

Time taken 1 hour 4 mins

Grade 10.36 out of 20.00 (52%)

Question 1

Partially correct

Mark 0.57 out of 1.00

Mark True, the actions done as part of code of swtch() in swtch.S, in xv6

True	False	
<input checked="" type="radio"/>	<input type="radio"/> 	Restore new callee saved registers from kernel stack of new context 
<input checked="" type="radio"/>	<input type="radio"/> 	Save old callee saved registers on kernel stack of old context 
<input type="radio"/> 	<input checked="" type="radio"/>	Save old callee saved registers on user stack of old context 
<input type="radio"/> 	<input checked="" type="radio"/>	Switch from old process context to new process context 
<input checked="" type="radio"/>	<input type="radio"/> 	Switch from one stack (old) to another(new) 
<input type="radio"/> 	<input checked="" type="radio"/>	Restore new callee saved registers from user stack of new context 
<input type="radio"/> 	<input checked="" type="radio"/>	Jump to code in new context 

Restore new callee saved registers from kernel stack of new context: True

Save old callee saved registers on kernel stack of old context: True

Save old callee saved registers on user stack of old context: False

Switch from old process context to new process context: False

Switch from one stack (old) to another(new): True

Restore new callee saved registers from user stack of new context: False

Jump to code in new context: False

Question 2

Partially correct

Mark 0.17 out of 0.50

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

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

Your answer is partially correct.

You have correctly selected 2.

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

Question 3

Partially correct

Mark 0.38 out of 1.00

Compare paging with demand paging and select the correct statements.

Select one or more:

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

Your answer is partially correct.

You have correctly selected 5.

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 4

Partially correct

Mark 0.44 out of 0.50

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

Assuming this, mark the statements as True/False

True	False	
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The OS may terminate the process while handling the interrupt of memory violation
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The hardware detects any memory access beyond the limit value and raises an interrupt
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/>	The hardware may terminate the process while handling the interrupt of memory violation
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The OS sets up the relocation and limit registers when the process is scheduled
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The process sets up its own relocation and limit registers when the process is scheduled
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The OS detects any memory access beyond the limit value and raises an interrupt
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The compiler generates machine code assuming appropriately sized segments for code, data and stack.

The OS may terminate the process while handling the interrupt of memory violation: True

The hardware detects any memory access beyond the limit value and raises an interrupt: True

The hardware may terminate the process while handling the interrupt of memory violation: False

The OS sets up the relocation and limit registers when the process is scheduled: True

The compiler generates machine code assuming continuous memory address space for process, and calculating appropriate sizes for code, and data;: True

The process sets up its own relocation and limit registers when the process is scheduled: False

The OS detects any memory access beyond the limit value and raises an interrupt: False

The compiler generates machine code assuming appropriately sized segments for code, data and stack.: False

Question 5

Correct

Mark 0.50 out of 0.50

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

10k, 25k, 12k, 7k, 9k, 13k

Suppose there is a request for chunk of size 9k, then the free chunk selected under each of the following schemes will be

Best fit:

9k



First fit:

10k



Worst fit:

25k

**Question 6**

Partially correct

Mark 0.50 out of 1.00

Select all the correct statements about MMU and its functionality

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.

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

Question 7

Incorrect

Mark 0.00 out of 0.50

Assuming a 8- KB page size, what is the page numbers for the address 874815 reference in decimal :
 (give answer also in decimal)

Answer: ×

The correct answer is: 107

Question 8

Incorrect

Mark 0.00 out of 0.25

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)

Segmentation, then paging	<input type="checkbox"/> Many continuous chunks each of page size	✗
Relocation + Limit	<input type="checkbox"/> Many continuous chunks of same size	✗
Segmentation	<input type="checkbox"/> one continuous chunk	✗
Paging	<input type="checkbox"/> many continuous chunks of variable size	✗

Your answer is incorrect.

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

Question 9

Incorrect

Mark 0.00 out of 0.50

Suppose the memory access time is 180ns and TLB hit ratio is 0.3, then effective memory access time is (in nanoseconds);

Answer: ✗

The correct answer is: 306.00

Question 10

Correct

Mark 0.50 out of 0.50

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.

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

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 11

Partially correct

Mark 0.83 out of 1.50

Arrange the following events in order, in page fault handling:

Disk interrupt wakes up the process

7	✓
---	---

The reference bit is found to be invalid by MMU

1	✓
---	---

OS makes available an empty frame

6	✗
---	---

Restart the instruction that caused the page fault

9	✓
---	---

A hardware interrupt is issued

3	✗
---	---

OS schedules a disk read for the page (from backing store)

5	✓
---	---

Process is kept in wait state

4	✗
---	---

Page tables are updated for the process

8	✓
---	---

Operating system decides that the page was not in memory

2	✗
---	---

Your answer is partially correct.

You have correctly selected 5.

The correct answer is: Disk interrupt wakes up the process → 7, The reference bit is found to be invalid by MMU → 1, OS makes available an empty frame → 4, Restart the instruction that caused the page fault → 9, A hardware interrupt is issued → 2, OS schedules a disk read for the page (from backing store) → 5, Process is kept in wait state → 6, Page tables are updated for the process → 8, Operating system decides that the page was not in memory → 3

Question 12

Incorrect

Mark 0.00 out of 0.50

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:

00001010

Now, there is a request for a chunk of 70 bytes.

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

Answer: 11101010



The correct answer is: 11111010

Question 13

Incorrect

Mark 0.00 out of 0.25

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

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

Your answer is incorrect.

The correct answer is: end, P2V(PHYSTOP)

Question 14

Partially correct

Mark 0.33 out of 0.50

Match the pair

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

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Hashed page table → Linear search on collision 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 15

Partially correct

Mark 0.29 out of 0.50

After virtual memory is implemented

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

True	False	
<input checked="" type="radio"/>	<input type="radio"/> ✗	Code need not be completely in memory
<input checked="" type="radio"/>	<input type="radio"/> ✗	Cumulative size of all programs can be larger than physical memory size
<input type="radio"/> ✗	<input checked="" type="radio"/>	Virtual access to memory is granted
<input checked="" type="radio"/>	<input type="radio"/> ✗	Logical address space could be larger than physical address space
<input type="radio"/> ✗	<input checked="" type="radio"/>	Virtual addresses are available
<input checked="" type="radio"/>	<input checked="" type="radio"/> ✗	Relatively less I/O may be possible during process execution
<input checked="" type="radio"/>	<input type="radio"/> ✗	One Program's size can be larger than physical memory size

Code need not be completely in memory: True

Cumulative size of all programs can be larger than physical memory size: True

Virtual access to memory is granted: False

Logical address space could be larger than physical address space: True

Virtual addresses are available: False

Relatively less I/O may be possible during process execution: True

One Program's size can be larger than physical memory size: True

Question 16

Partially correct

Mark 0.64 out of 1.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB only
Mark statements True or False**True False**

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

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True

The stack allocated in entry.S is used as stack for scheduler's context for first processor: True

The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True

The free page-frame are created out of nearly 222 MB: True

The kernel code and data take up less than 2 MB space: True

The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False

The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False

PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

xv6 uses physical memory upto 224 MB only: True

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True

The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

Question 17

Incorrect

Mark 0.00 out of 1.50

Consider the reference string

6 4 2 0 1 2 6 9 2 0 5

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

Answer: ✖

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

The correct answer is: 9

Question 18

Partially correct

Mark 0.31 out of 0.50

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

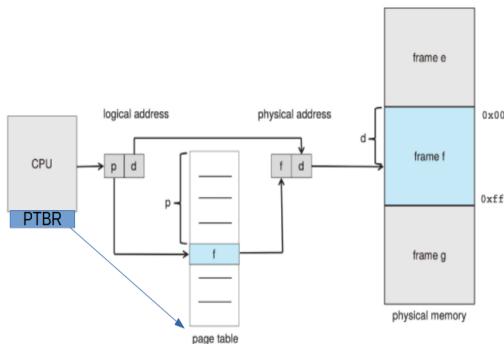


Figure 9.8 Paging hardware.

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

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

The PTBR is present in the CPU as a register: True

The page table is indexed using frame number: False

The page table is indexed using page number: True

The locating of the page table using PTBR also involves paging translation: False

Size of page table is always determined by the size of RAM: False

The page table is itself present in Physical memory: True

Maximum Size of page table is determined by number of bits used for page number: True

The physical address may not be of the same size (in bits) as the logical address: True

Question 19

Correct

Mark 2.00 out of 2.00

Given below is shared memory code with two processes sharing a memory segment.

The first process sends a user input string to second process. The second capitalizes the string. Then the first process prints the capitalized version.

Fill in the blanks to complete the code.

// First process

```
#define SHMSZ 27

int main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s, string[128];
    key = 5679;
    if ((shmid =
        shmget
        ✓ (key, SHMSZ, IPC_CREAT | 0666)) < 0) {
        perror("shmget");
        exit(1);
    }
    if ((shm =
        shmat
        ✓ (shmid, NULL, 0)) == (char *) -1) {
        perror("shmat");
        exit(1);
    }
    s = shm;
    *s = '$';
    scanf("%s", string);
    strcpy(s + 1, string);
    *s =
        @
        ✓ ';' //note the quotes
    while(*s != '
        $
        ')
        sleep(1);
        printf("%s\n", s + 1);
        exit(0);
}
```

//Second process

```
#define SHMSZ 27

int main()
{
    int shmid;
    key_t key;
    char *shm, *s;
    int i;
    char string[128];
    key =
        5679
```

```

✓ ;
if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
    perror("shmget");
    exit(1);
}
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
    perror("shmat");
    exit(1);
}
s =

✓ ;
while(*s != '@')
    sleep(1);
for(i = 0; i < strlen(s + 1); i++)
    s[i + 1] = toupper(s[i + 1]);
*s = '$';
exit(0);
}

```

Question 20

Partially correct

Mark 0.25 out of 0.50

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

return a free page, if available; 0, otherwise

Create page table entries for a given range of virtual and physical addresses; including page directory entries if needed

Array listing the kernel memory mappings, to be used by setupkvm()

Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices

Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary

Setup kernel part of a page table, and switch to that page table

 kinit1()

 mappages()

 kmap[]

 kvmalloc()

 walkpgdir()

 setupkvm()

Your answer is partially correct.

You have correctly selected 3.

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

Question 21

Partially correct

Mark 1.53 out of 2.50

Order events in xv6 timer interrupt code

(Transition from process P1 to P2's code.)

P2 is selected and marked RUNNING

12 ✓

Change of stack from user stack to kernel stack of P1

3 ✓

Timer interrupt occurs

2 ✓

alltraps() will call iret

17 ✗

change to context of P2, P2's kernel stack in use now

13 ✓

P2's trap() will return to alltraps

16 ✗

jump in vector.S

4 ✓

P2 will return from sched() in yield()

14 ✗

yield() is called

8 ✓

trap() is called

7 ✓

Process P2 is executing

18 ✗

P1 is marked as RUNNABLE

9 ✓

P2's yield() will return in trap()

15 ✗

Process P1 is executing

1 ✓

sched() is called,

11 ✗

change to context of the scheduler, scheduler's stack in use now

10 ✗

jump to alltraps

5 ✓

Trapframe is built on kernel stack of P1

6 ✓

Your answer is partially correct.

You have correctly selected 11.

The correct answer is: P2 is selected and marked RUNNING → 12, Change of stack from user stack to kernel stack of P1 → 3, Timer interrupt occurs → 2, alltraps() will call iret → 18, change to context of P2, P2's kernel stack in use now → 13, P2's trap() will return to alltraps → 17, jump in vector.S → 4, P2 will return from sched() in yield() → 15, yield() is called → 8, trap() is called → 7, Process P2 is executing → 14, P1 is marked as RUNNABLE → 9, P2's yield() will return in trap() → 16, Process P1 is executing → 1, sched() is called, → 10, change to context of the scheduler, scheduler's stack in use now → 11, jump to alltraps → 5, Trapframe is built on kernel stack of P1 → 6

Question 22

Incorrect

Mark 0.00 out of 1.00

Given that the memory access time is 200 ns, probability of a page fault is 0.7 and page fault handling time is 8 ms,
The effective memory access time in nanoseconds is:

Answer: ✖

The correct answer is: 5600060.00

Question 23

Correct

Mark 0.25 out of 0.25

Select the state that is not possible after the given state, for a process:

- New: Running ✓
- Ready : Waiting ✓
- Running: None of these ✓
- Waiting: Running ✓

Question 24

Partially correct

Mark 0.63 out of 1.00

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

- a. scheduler() switches to the selected process's context ✓
- b. When either sched() or scheduler() is called, it does not return immediately to caller ✓
- c. After call to swtch() in sched(), the control moves to code in scheduler()
- d. Each call to sched() or scheduler() involves change of one stack inside swtch() ✓
- e. After call to swtch() in scheduler(), the control moves to code in sched()
- f. When either sched() or scheduler() is called, it results in a context switch ✓
- g. sched() switches to the scheduler's context ✓
- h. sched() and scheduler() are co-routines

Your answer is partially correct.

You have correctly selected 5.

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

Question 25

Correct

Mark 0.25 out of 0.25

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

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



Your answer is correct.

The correct answer is: Singly linked NULL terminated list

[◀ \(Assignment\) lseek system call in xv6](#)

Jump to...

Started on Saturday, 20 February 2021, 2:51 PM

State Finished

Completed on Saturday, 20 February 2021, 3:55 PM

Time taken 1 hour 3 mins

Grade 7.30 out of 20.00 (37%)

Question 1

Partially correct

Mark 0.80 out of 1.00

Select all the correct statements about the state of a process.

- a. A process can self-terminate only when it's running ✓
- b. Typically, it's represented as a number in the PCB ✓
- c. A process that is running is not on the ready queue ✓
- d. Processes in the ready queue are in the ready state ✓
- e. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers
- f. Changing from running state to waiting state results in "giving up the CPU" ✓
- g. A process in ready state is ready to receive interrupts
- h. A waiting process starts running after the wait is over ✗
- i. A process changes from running to ready state on a timer interrupt ✓
- j. A process in ready state is ready to be scheduled ✓
- k. A running process may terminate, or go to wait or become ready again ✓
- l. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✓
- m. A process waiting for any condition is woken up by another process only
- n. A process changes from running to ready state on a timer interrupt or any I/O wait

Your answer is partially correct.

You have selected too many options.

The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question 2

Incorrect

Mark 0.00 out of 1.00

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

`jmp *%eax`
in entry.S

0x7c00 to 0x10000



`ljmp $(SEG_KCODE<<3), $start32`
in bootasm.S

0x10000 to 0x7c00



`call bootmain`
in bootasm.S

0x7c00 to 0x10000



`cli`
in bootasm.S

0x7c00 to 0



`readseg((uchar*)elf, 4096, 0);`
in bootmain.c

The 4KB area in kernel image, loaded in memory, named as 'stack'



Your answer is incorrect.

The correct answer is: `jmp *%eax`

`in entry.S` → The 4KB area in kernel image, loaded in memory, named as 'stack', `ljmp $(SEG_KCODE<<3), $start32`

`in bootasm.S` → Immaterail as the stack is not used here, `call bootmain`

`in bootasm.S` → 0x7c00 to 0, `cli`

`in bootasm.S` → Immaterail as the stack is not used here, `readseg((uchar*)elf, 4096, 0);`

`in bootmain.c` → 0x7c00 to 0

Question 3

Correct

Mark 0.25 out of 0.25

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

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

Your answer is correct.

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

Question 4

Partially correct

Mark 0.30 out of 0.50

Consider the following command and its output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

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

```
kernel: $(OBJS) entry.o entryother initcode kernel.ld
$(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
$(OBJDUMP) -S kernel > kernel.asm
$(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- a. Although the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- b. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- c. The kernel.ld file contains instructions to the linker to link the kernel properly ✓
- d. The bootmain() code does not read the kernel completely in memory
- e. readseg() reads first 4k bytes of kernel in memory
- f. Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.
- g. The kernel.asm file is the final kernel file
- h. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.
- i. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✓

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(), readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question 5

Partially correct

Mark 0.50 out of 1.00

```
int f() {  
    int count;  
    for (count = 0; count < 2; count++) {  
        if (fork() == 0)  
            printf("Operating-System\n");  
    }  
    printf("TYCOMP\n");  
}
```

The number of times "Operating-System" is printed, is:

Answer:

The correct answer is: 7.00

Question 6

Partially correct

Mark 0.40 out of 0.50

Select Yes/True if the mentioned element must be a part of PCB

Select No/False otherwise.

Yes**No**

<input checked="" type="radio"/>	<input checked="" type="radio"/>	PID	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Process context	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	List of opened files	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Process state	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Parent's PID	✗
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Pointer to IDT	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Function pointers to all system calls	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Memory management information about that process	✓
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Pointer to the parent process	✗
<input checked="" type="radio"/>	<input checked="" type="radio"/>	EIP at the time of context switch	✓

PID: Yes

Process context: Yes

List of opened files: Yes

Process state: Yes

Parent's PID: No

Pointer to IDT: No

Function pointers to all system calls: No

Memory management information about that process: Yes

Pointer to the parent process: Yes

EIP at the time of context switch: Yes

Question 7

Incorrect

Mark 0.00 out of 1.00

Select all the correct statements about code of bootmain() in xv6

```

void
bootmain(void)
{
    struct elfhdr *elf;
    struct proghdr *ph, *eph;
    void (*entry)(void);
    uchar* pa;

    elf = (struct elfhdr*)0x10000; // scratch space

    // Read 1st page off disk
    readseg((uchar*)elf, 4096, 0);

    // Is this an ELF executable?
    if(elf->magic != ELF_MAGIC)
        return; // let bootasm.S handle error

    // Load each program segment (ignores ph flags).
    ph = (struct proghdr*)((uchar*)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++){
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    }

    // Call the entry point from the ELF header.
    // Does not return!
    entry = (void(*)(void))(elf->entry);
    entry();
}

```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

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

Your answer is incorrect.

The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question 8

Partially correct

Mark 0.13 out of 0.25

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

- a. Check the program for logical errors ✓
- b. Convert high level language code to machine code
- c. Process the # directives in a C program
- d. Invoke the linker to link the function calls with their code, extern globals with their declaration
- e. Check the program for syntactical errors
- f. Suggest alternative pieces of code that can be written

Your answer is partially correct.

You have correctly selected 1.

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

Question 9

Correct

Mark 0.25 out of 0.25

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

Cache	6	✓
Hard Disk	3	✓
RAM	5	✓
Optical Disks	2	✓
Non volatile memory	4	✓
Registers	7	✓
Magnetic Tapes	1	✓

Your answer is correct.

The correct answer is: Cache → 6, Hard Disk → 3, RAM → 5, Optical Disks → 2, Non volatile memory → 4, Registers → 7, Magnetic Tapes → 1

Question 10

Partially correct

Mark 0.21 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- a. local variable declaration
- b. global variables
- c. function calls ✗
- d. #directives ✓
- e. expressions
- f. pointer dereference
- g. typedefs ✓

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: #directives, typedefs, global variables

Question 11

Correct

Mark 0.25 out of 0.25

Match a system call with it's description

pipe	create an unnamed FIFO storage with 2 ends - one for reading and another for writing	✓
dup	create a copy of the specified file descriptor into smallest available file descriptor	✓
dup2	create a copy of the specified file descriptor into another specified file descriptor	✓
exec	execute a binary file overlaying the image of current process	✓
fork	create an identical child process	✓

Your answer is correct.

The correct answer is: pipe → create an unnamed FIFO storage with 2 ends - one for reading and another for writing, dup → create a copy of the specified file descriptor into smallest available file descriptor, dup2 → create a copy of the specified file descriptor into another specified file descriptor, exec → execute a binary file overlaying the image of current process, fork → create an identical child process

Question 12

Correct

Mark 0.25 out of 0.25

Match the register with the segment used with it.

eip	cs	✓
edi	es	✓
esi	ds	✓
ebp	ss	✓
esp	ss	✓

Your answer is correct.

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

Question 13

Correct

Mark 0.25 out of 0.25

What's the trapframe in xv6?

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

Your answer is correct.

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

Question 14

Incorrect

Mark 0.00 out of 0.50

Select all the correct statements about linking and loading.

Select one or more:

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

Your answer is incorrect.

The correct answers are: Continuous memory management schemes can support static linking and static loading. (may be inefficiently), Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently), Dynamic linking essentially results in relocatable code., Loader is part of the operating system, Dynamic linking and loading is not possible without demand paging or demand segmentation.

Question 15

Incorrect

Mark 0.00 out of 0.25

In bootasm.S, on the line

```
ljmp    $(SEG_KCODE<<3), $start32
```

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

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

Your answer is incorrect.

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

Question 16

Partially correct

Mark 0.07 out of 0.50

Order the events that occur on a timer interrupt:

Change to kernel stack

1	✗
---	---

Jump to a code pointed by IDT

2	✗
---	---

Jump to scheduler code

5	✗
---	---

Set the context of the new process

4	✗
---	---

Save the context of the currently running process

3	✓
---	---

Execute the code of the new process

6	✗
---	---

Select another process for execution

7	✗
---	---

Your answer is partially correct.

You have correctly selected 1.

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

Question 17

Incorrect

Mark 0.00 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 programs are correct ✗
- b. Program 2 makes sure that there is one file offset used for '2' and '1' ✗
- c. Only Program 2 is correct ✗
- d. Program 2 does 1>&2 ✗
- e. Program 2 ensures 2>&1 and does not ensure >/tmp/ddd ✗
- f. Program 1 makes sure that there is one file offset used for '2' and '1' ✓
- g. Program 1 is correct for >/tmp/ddd but not for 2>&1 ✗
- h. Program 1 does 1>&2 ✗
- i. Both program 1 and 2 are incorrect ✗
- j. Program 2 is correct for >/tmp/ddd but not for 2>&1 ✗
- k. Only Program 1 is correct ✓
- l. Program 1 ensures 2>&1 and does not ensure >/tmp/ddd ✗

Your answer is incorrect.

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 18

Correct

Mark 0.25 out of 0.25

Select the option which best describes what the CPU does during its powered ON lifetime

- a. Ask the user what is to be done, and execute that task
- b. Ask the OS what is to be done, and execute that task
- c. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask the User or the OS what is to be done next, repeat
- d. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per ✓ the instruction itself, repeat
- e. Fetch instruction specified by OS, Decode and execute it, repeat
- f. Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, Ask OS what is to be done next, repeat

The correct answer is: Fetch instructions specified by location given by PC, Decode and Execute it, during execution increment PC or change PC as per the instruction itself, repeat

Question 19

Partially correct

Mark 0.86 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
int main(int argc, char *argv[]) {
    int fd1, fd2 = 1, fd3 = 1, fd4 = 1;

    fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
    fd2 = open("/tmp/2", O_RDONLY);
    fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
    close(0);
    close(1);
    dup(fd2);
    dup(fd3);
    close(fd3);
    dup2(fd2, fd4);
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);
    return 0;
}
```

1	closed	✗
fd4	/tmp/2	✓
fd2	/tmp/2	✓
fd1	/tmp/1	✓
2	stderr	✓
0	/tmp/2	✓
fd3	closed	✓

Your answer is partially correct.

You have correctly selected 6.

The correct answer is: 1 → /tmp/3, fd4 → /tmp/2, fd2 → /tmp/2, fd1 → /tmp/1, 2 → stderr, 0 → /tmp/2, fd3 → closed

Question 20

Incorrect

Mark 0.00 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[
```

```
    x ] [2];
```

```
    pipe(
```

```
    x );
```

```
    pid1 =
```

```
    x ;
```

```
    if(pid1 != 0) {
```

```
        close(pfd[0]
```

```
    x );
```

```
        close(
```

```
    x );
```

```
        dup(
```

```
    x );
```

```
        execl("/bin/ls", "/bin/ls", "
```

```
    x ", NULL);
```

```
    }
```

```
    pipe(
```

```
    x );
```

```
    x = fork();
```

```
    if(pid2 == 0) {
```

```
        close(
```

```
        x ;
```

```
        close(0);
```

```
        dup(
```

```
        x );
```

```
        close(pfd[1]
```

```
✗ );
close(
  
✗ );
dup(
  
✗ );
execl("/usr/bin/head", "/usr/bin/head", "  
  
", NULL);
} else {
close(pfd
  
✗ );
close(
  
✗ );
dup(
  
✗ );
close(pfd
  
✗ );
execl("/usr/bin/tail", "/usr/bin/tail", "  
  
", NULL);
}  
}
```

Question 21

Partially correct

Mark 0.11 out of 1.00

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

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

Your answer is partially correct.

You have selected too many options.

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

Question 22

Correct

Mark 1.00 out of 1.00

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

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

hi ✓

main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }

hi hi ✓

main() { int i = NULL; fork(); printf("hi\n"); }

hi hi ✓

main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }

hi ✓

Your answer is correct.

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

Question 23

Incorrect

Mark 0.00 out of 0.50

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

Select all the appropriate choices

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

Your answer is incorrect.

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

Question 24

Incorrect

Mark 0.00 out of 0.50

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

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

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

Your answer is incorrect.

The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

Question 25

Incorrect

Mark 0.00 out of 1.00

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

Select one or more:

a. P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheuler

P1 running

P1's system call return

b. P1 running

P1 makes sytem call and blocks

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again



c. P1 running

P1 makes system call

system call returns

P1 running

timer interrupt

Scheduler running

P2 running

d. P1 running

P1 makes sytem call and blocks

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P3 running

Hardware interrupt

Interrupt unblocks P1

Interrupt returns

P3 running

Timer interrupt

Scheduler

P1 running



e.

P1 running

P1 makes sytem call

Scheduler

P2 running

P2 makes sytem call and blocks

Scheduler

P1 running again

f. P1 running

keyboard hardware interrupt

keyboard interrupt handler running

interrupt handler returns

P1 running

P1 makes sytem call

system call returns



P1 running
timer interrupt
scheduler
P2 running

Your answer is incorrect.

The correct answers are: P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again, P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheuler

P1 running

P1's system call return,

P1 running

P1 makes system call

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

Question 26

Correct

Mark 0.25 out of 0.25

Which of the following are the files related to bootloader in xv6?

- a. bootasm.s and entry.S
- b. bootasm.S and bootmain.c ✓
- c. bootasm.S, bootmain.c and bootblock.c
- d. bootmain.c and bootblock.S

Your answer is correct.

The correct answer is: bootasm.S and bootmain.c

Question 27

Correct

Mark 0.25 out of 0.25

Match the following parts of a C program to the layout of the process in memory

Instructions	Text section	✓
Local Variables	Stack Section	✓
Dynamically allocated memory	Heap Section	✓
Global and static data	Data section	✓

Your answer is correct.

The correct answer is:

Instructions → Text section, Local Variables → Stack Section,
Dynamically allocated memory → Heap Section,
Global and static data → Data section

Question 28

Incorrect

Mark 0.00 out of 0.50

What will this program do?

```
int main() {  
    fork();  
    execl("/bin/ls", "/bin/ls", NULL);  
    printf("hello");  
}
```

- a. one process will run ls, another will print hello
- b. run ls once ✗
- c. run ls twice
- d. run ls twice and print hello twice
- e. run ls twice and print hello twice, but output will appear in some random order

Your answer is incorrect.

The correct answer is: run ls twice

Question 29

Correct

Mark 0.25 out of 0.25

What is the OS Kernel?

- a. The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run ✓ correct
- b. The set of tools like compiler, linker, loader, terminal, shell, etc.
- c. Only the system programs like compiler, linker, loader, etc.
- d. Everything that I see on my screen

The correct answer is: The code that controls hardware, abstracts access to hardware resources using system calls, creates an environment for processes to be created and run

Question 30

Correct

Mark 0.50 out of 0.50

Which of the following is/are not saved during context switch?

- a. Program Counter
- b. General Purpose Registers
- c. Bus ✓
- d. Stack Pointer
- e. MMU related registers/information
- f. Cache ✓
- g. TLB ✓

Your answer is correct.

The correct answers are: TLB, Cache, Bus

Question 31

Partially correct

Mark 0.10 out of 0.25

Select the order in which the various stages of a compiler execute.

Linking	3	
Syntactical Analysis	2	
Pre-processing	1	
Intermediate code generation	does not exist	
Loading	4	

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: Linking → 4, Syntactical Analysis → 2, Pre-processing → 1, Intermediate code generation → 3, Loading → does not exist

Question 32

Partially correct

Mark 0.08 out of 0.50

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

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

Your answer is partially correct.

You have correctly selected 1.

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

Question 33

Not answered

Marked out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

- j. On any interrupt/syscall/exception the control first jumps in trapasm.S
- k. The function trap() is called irrespective of hardware interrupt/system-call/exception
- l. xv6 uses the 0x64th entry in IDT for system calls
- m. Before going to alltraps, the kernel stack contains upto 5 entries.

Your answer is incorrect.

The correct answers are: All the 256 entries in the IDT are filled, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in trapasm.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on kernel stack, The function trap() is called irrespective of hardware interrupt/system-call/exception, The CS and EIP are changed only after pushing user code's SS,ESP on stack

[◀ \(Assignment\) Change free list management in xv6](#)

Jump to...



Started on Friday, 17 March 2023, 2:33 PM

State Finished

Completed on Friday, 17 March 2023, 4:54 PM

Time taken 2 hours 21 mins

Grade 6.37 out of 10.00 (63.73%)

Question 1

Incorrect

Mark 0.00 out of 0.50

The first instruction that runs when you do "make qemu" is

cli

from bootasm.S

Why?

- a. "cli" stands for clear screen and the screen should be cleared before OS boots.
- b. "cli" that is Command Line Interface needs to be enabled first
- c. "cli" clears all registers and makes them zero, so that processor is as good as "new"
- d. "cli" clears the pipeline of the CPU so that it is as good as "fresh" CPU
- e. "cli" enables interrupts, it is required because the kernel supports interrupts.
- f. "cli" disables interrupts. It is required because as of now there are no interrupt handlers available X
- g. It disables interrupts. It is required because the interrupt handlers of kernel are not yet installed.
- h. "cli" enables interrupts, it is required because the kernel must handle interrupts.

Your answer is incorrect.

The correct answer is: It disables interrupts. It is required because the interrupt handlers of kernel are not yet installed.

Question 2

Incorrect

Mark 0.00 out of 0.50

The struct buf has a sleeplock, and not a spinlock, because

- a. struct buf is used as a general purpose cache by kernel and cache operations take lot of time, so better to use sleeplock rather than spinlock
- b. sleeplock is preferable because it is used in interrupt context and spinlock can not be used in interrupt context
- c. It could be a spinlock, but xv6 has chosen sleeplock for purpose of demonstrating how to use a sleeplock.
- d. struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is the only option available. X
- e. struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is preferred to spinning/busy-wait for the desired buf.

Your answer is incorrect.

The correct answer is: struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is preferred to spinning/busy-wait for the desired buf.

Question 3

Partially correct

Mark 0.25 out of 0.50

when is each of the following stacks allocated?

kernel stack of process

during fork() in allocproc()



kernel stack for scheduler, on first processor

on the arrival of a hardware interrupt



user stack of process

during exec()



kernel stack for the scheduler, on other processors

in main()->startothers()



Your answer is partially correct.

You have correctly selected 2.

The correct answer is: kernel stack of process → during fork() in allocproc(), kernel stack for scheduler, on first processor → in entry.S, user stack of process → during fork() in copyuvml(), kernel stack for the scheduler, on other processors → in main()->startothers()

Question 4

Partially correct

Mark 0.56 out of 0.75

Mark statements as True/False w.r.t. ptable.lock

True	False	
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	A process can sleep on ptable.lock if it can't acquire it.
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	It is taken by one process but released by another process, running on same processor
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	ptable.lock protects the proc[] array and all struct proc in the array
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The swtch() in scheduler() is called without holding the ptable.lock when control jumps to it from sched()
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	the rule of "never block holding a spinlock" does not apply to ptable.lock in xv6
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	One sequence of function calls which takes and releases the ptable.lock is this: iderw->sleep, acquire(ptable.lock)->sched->swtch()->scheduler()->swtch()->yield(), release(ptable.lock)
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	ptable.lock can be held by different processes on different processors at the same time
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	ptable.lock is acquired but never released

A process can sleep on ptable.lock if it can't acquire it.: False

It is taken by one process but released by another process, running on same processor: True

ptable.lock protects the proc[] array and all struct proc in the array: True

The swtch() in scheduler() is called without holding the ptable.lock when control jumps to it from sched(): False

the rule of "never block holding a spinlock" does not apply to ptable.lock in xv6: True

One sequence of function calls which takes and releases the ptable.lock is this:

iderw->sleep, acquire(ptable.lock)->sched->swtch()->scheduler()->swtch()->yield(), release(ptable.lock): True

ptable.lock can be held by different processes on different processors at the same time: False

ptable.lock is acquired but never released: False

Question 5

Correct

Mark 0.50 out of 0.50

Consider the following command and its output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

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

kernel: $(OBJS) entry.o entryother initcode kernel.ld
    $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
    $(OBJDUMP) -S kernel > kernel.asm
    $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- a. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✓
- b. Althought the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- c. The bootmain() code does not read the kernel completely in memory
- d. Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes. ✓
- e. readseg() reads first 4k bytes of kernel in memory ✓
- f. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.
- g. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- h. The kernel.ld file contains instructions to the linker to link the kernel properly ✓
- i. The kernel.asm file is the final kernel file

Your answer is correct.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(), readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question 6

Partially correct

Mark 0.43 out of 0.75

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

movw %ax, %gs	real mode	✗
jmp *%eax	protected mode with segmentation and 4 MB pages	✓
inb \$0x64,%al	real mode	✓
ljmp \$(SEG_KCODE<<3), \$start32	protected mode with only segmentation	✗
readseg((uchar*)elf, 4096, 0);	protected mode with only segmentation	✓
orl \$CRO_PE, %eax	real mode	✓
movl \$(V2P_WO(entrypgdir)), %eax	protected mode with segmentation and 4 MB pages	✗

The correct answer is: movw %ax, %gs → protected mode with only segmentation, jmp *%eax → protected mode with segmentation and 4 MB pages, inb \$0x64,%al → real mode, ljmp \$(SEG_KCODE<<3), \$start32 → real mode, readseg((uchar*)elf, 4096, 0); → protected mode with only segmentation, orl \$CRO_PE, %eax → real mode, movl \$(V2P_WO(entrypgdir)), %eax → protected mode with only segmentation

Question 7

Correct

Mark 0.50 out of 0.50

Which of the following is DONE by allocproc() ?

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

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

Question 8

Partially correct

Mark 0.25 out of 0.50

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

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

kmem.use_lock is set to 1 after free page list is created, so that kmem.lock is taken before accessing kmem.freelist.: True

The pointers that link the pages together are in the first 4 bytes of the pages themselves: True

if(kmem.use_lock)

acquire(&kmem.lock);

is not done when called from kinit1() because there is no need to take the lock when kinit1() is running because interrupts are disabled and only one processor is running: True

free page list is a singly circular linked list.: False

if(kmem.use_lock)

acquire(&kmem.lock);

this "if" condition is true, when kinit2() runs because multi-processor support has been enabled by now.: False

the kmem.lock is used by kfree() and kalloc() only.: True

Question 9

Partially correct

Mark 0.55 out of 1.00

Given below is code of sleeplock in xv6.

```
// Long-term locks for processes
struct sleeplock {
    uint locked;          // Is the lock held?
    struct spinlock lk;  // spinlock protecting this sleep lock

    // For debugging:
    char *name;           // Name of lock.
    int pid;              // Process holding lock
};

void
acquiresleep(struct sleeplock *lk)
{
    acquire(&lk->lk);
    while (lk->locked) {
        sleep(lk, &lk->lk);
    }
    lk->locked = 1;
    lk->pid = myproc()->pid;
    release(&lk->lk);
}

void
releasesleep(struct sleeplock *lk)
{
    acquire(&lk->lk);
    lk->locked = 0;
    lk->pid = 0;
    wakeup(lk);
    release(&lk->lk);
}
```

Mark the statements as True/False w.r.t. this code.

True	False
<input checked="" type="radio"/> <input type="radio"/>	Wakeup() will wakeup the first process waiting for the lock
<input checked="" type="radio"/> <input type="radio"/>	<pre>acquire(&lk->lk); while (lk->locked) { sleep(lk, &lk->lk); } could also be written as acquire(&lk->lk); if (lk->locked) { sleep(lk, &lk->lk); }</pre>

True	False	
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	A process has acquired the sleeplock when it comes out of sleep(): ✗
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	sleep() is called holding a spinlock. This could be avoided by releasing the lock before calling sleep() and acquiring it again after call to sleep(): ✗
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	sleep() is the function which blocks a process. ✓
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	The process which called acquiresleep() and then got blocked, is woken up by the timer interrupt ✓ it's woken up by another process which called releasesleep() and then wakeup()
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	the 'spinlock lk' protects 'locked' variable, but not the 'name' nor the 'pid' ✗
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	the 'spinlock lk' is needed in a sleeplock, because access to the sleeplock for locking/unlocking itself creates a critical section ✓
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	The spinlock lk->lk is held when the process comes out of sleep(): ✓
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	Sleeplock() will ensure that either the process gets the lock or the process gets blocked. ✓
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	All processes waiting for the sleeplock will have a race for aquiring lk->lk spinlock, because all are woken up ✓ wakeup() wakes up all processes, and they "thunder" to take the spinlock.

Wakeup() will wakeup the first process waiting for the lock: False

```
acquire(&lk->lk);
while (!lk->locked) {
    sleep(lk, &lk->lk);
}
```

could also be written as

```
acquire(&lk->lk);
if (!lk->locked) {
    sleep(lk, &lk->lk);
}: False
```

A process has acquired the sleeplock when it comes out of sleep(): False

sleep() is called holding a spinlock. This could be avoided by releasing the lock before calling sleep() and acquiring it again after call to sleep(): False

The process which called acquiresleep() and then got blocked, is woken up by the timer interrupt: True

The 'spinlock lk' protects 'locked' variable, but not the 'name' nor the 'pid': False

The 'spinlock lk' is needed in a sleeplock, because access to the sleeplock for locking/unlocking itself creates a critical section: True

The spinlock lk->lk is held when the process comes out of sleep(): True

Sleeplock() will ensure that either the process gets the lock or the process gets blocked.: True

All processes waiting for the sleeplock will have a race for aquiring lk->lk spinlock, because all are woken up: True

Question 10

Partially correct

Mark 0.50 out of 1.00

Mark the statements as True/False w.r.t. swtch()

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	p->context used in scheduler()->swtch() was Generally set when the process was interrupted earlier, and came via sched()->swtch()
<input checked="" type="radio"/>	<input type="radio"/>	swtch() is written in assembly language, because it violates calling convention, by changing the stack itself.
<input type="radio"/>	<input checked="" type="radio"/>	swtch() is written in assembly language because it violates the calling convention by pushing parameters on the stack on its own.
<input type="radio"/>	<input checked="" type="radio"/>	swtch() called from scheduler() changes the stack from the process's kernel stack to the scheduler's kernel stack.
<input checked="" type="radio"/>	<input type="radio"/>	swtch() is called only from sched() or scheduler()
<input type="radio"/>	<input checked="" type="radio"/>	switch stores the old context on new stack, and restores new context from old stack.
<input type="radio"/>	<input checked="" type="radio"/>	sched() is the only place when p->context is set
<input type="radio"/>	<input checked="" type="radio"/>	movl %esp, (%eax) means, *(c->scheduler) = contents of esp When swtch() is called from scheduler()
<input checked="" type="radio"/>	<input type="radio"/>	push in swtch() happens on old stack, while pop happens from new stack
<input checked="" type="radio"/>	<input type="radio"/>	swtch() changes the context from "old" to "new"

p->context used in scheduler()->swtch() was **Generally** set when the process was interrupted earlier, and came via sched()->swtch(): True
 swtch() is written in assembly language, because it violates calling convention, by changing the stack itself.: True

swtch() is written in assembly language because it violates the calling convention by pushing parameters on the stack on its own.: False
 swtch() called from scheduler() changes the stack from the process's kernel stack to the scheduler's kernel stack.: False

swtch() is called only from sched() or scheduler(): True

switch stores the old context on new stack, and restores new context from old stack.: False

sched() is the only place when p->context is set: False

movl %esp, (%eax)

means, *(c->scheduler) = contents of esp

When swtch() is called from scheduler(): False

push in swtch() happens on old stack, while pop happens from new stack: True

swtch() changes the context from "old" to "new": True

Question **11**

Correct

Mark 0.25 out of 0.25

Select the odd one out

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

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

Question **12**

Correct

Mark 0.25 out of 0.25

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

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

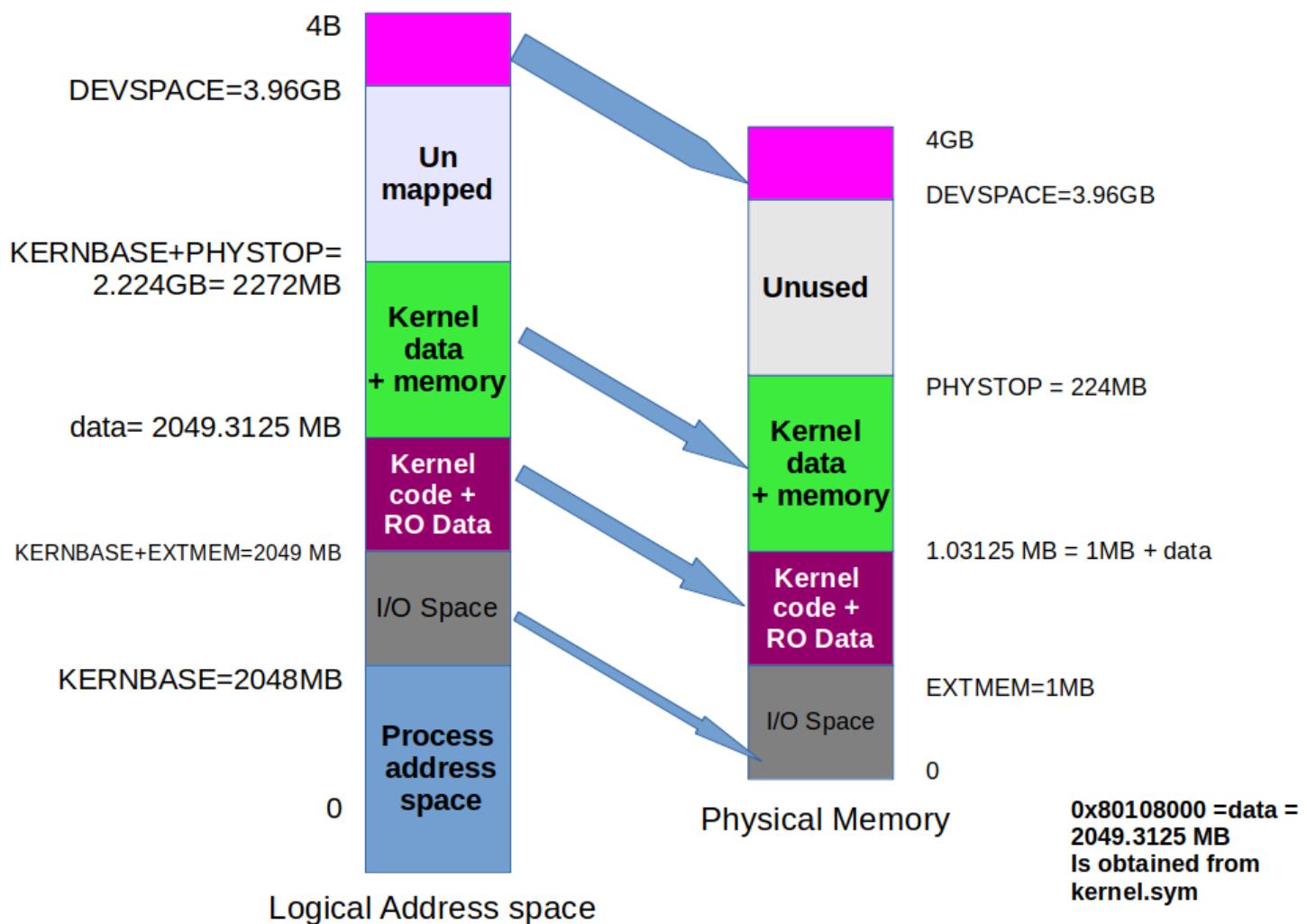
The correct answer is: 801154a8

Question 13

Partially correct

Mark 0.36 out of 0.50

With respect to this diagram, mark statements as True/False.



True False

<input checked="" type="radio"/>	<input type="radio"/>	This diagram only shows the absolutely defined virtual->physical mappings, not the mappings defined at run time by kernel.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/>	"Kernel data + memory" on right side, here refers to the region from which pages are allocated to the kernel and process both.	<input checked="" type="checkbox"/> "Kernel data + memory" on LEFT side, here refers to the virtual addresses of kernel used at run time.
<input checked="" type="radio"/>	<input type="radio"/>	The kernel virtual addresses start from KERNLINK = KERNBASE + EXTMEM	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/>	The process's pages are mapped into physical memory from 1.03125 MB to PHYSTOP.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/>	The kernel file, after compilation, has maximum virtual address up to "data" as shown in the diagram, which is equal to "end" variable	<input checked="" type="checkbox"/>

True **False**

True	False	
<input checked="" type="radio"/>	<input checked="" type="radio"/>	When bootloader loads the kernel, then physical memory from EXTMEM upto EXTMEM + data is occupied.
<input checked="" type="radio"/>	<input checked="" type="radio"/>	PHYSTOP can be changed , but that needs kernel recompilation and re-execution.

This diagram only shows the absolutely defined virtual->physical mappings, not the mappings defined at run time by kernel.: True
"Kernel data + memory" on right side, here refers to the region from which pages are allocated to the kernel and process both.: True

The kernel virtual addresses start from KERNLINK = KERNBASE + EXTMEM: True

The process's pages are mapped into physical memory from 1.03125 MB to PHYSTOP.: True

The kernel file, after compilation, has maximum virtual address up to "data" as shown in the diagram, which is equal to "end" variable: True

When bootloader loads the kernel, then physical memory from EXTMEM upto EXTMEM + data is occupied.: True

PHYSTOP can be changed , but that needs kernel recompilation and re-execution.: True

Question 14

Correct

Mark 0.25 out of 0.25

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

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

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

Question **15**

Correct

Mark 0.50 out of 0.50

We often use terms like "swtch() changes stack from process's kernel stack to scheduler's stack", or "the values are pushed on stack", or "the stack is initialized to the new page", etc. while discussing xv6 on x86.

Which of the following most accurately describes the meaning of "stack" in such sentences?

- a. The stack variable used in the program being discussed
- b. The region of memory where the kernel remembers all the function calls made
- c. The region of memory allocated by kernel for storing the parameters of functions
- d. The ss:esp pair ✓
- e. The "stack" variable declared in "stack.S" in xv6
- f. the region of memory which is currently used as stack by processor
- g. The stack segment

Your answer is correct.

The correct answer is: The ss:esp pair

Question 16

Partially correct

Mark 0.31 out of 0.50

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

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

The value of 'chan' is changed only in sleep(): True

when chan is NULL, the 'state' in proc must be RUNNABLE.: False

chan stores the address of the variable, representing a condition, for which the process is waiting.: True

'chan' is used only by the sleep() and wakeup1() functions.: True

chan is the head pointer to a linked list of processes, waiting for a particular event to occur: False

When chan is not NULL, the 'state' in struct proc must be SLEEPING: True

in xv6, the address of an appropriate variable is used as a "condition" for a waiting process.: True

Changing the state of a process automatically changes the value of 'chan': False

Question 17

Correct

Mark 0.25 out of 0.25

Match function with its meaning

iderw	Issue a disk read/write for a buffer, block the issuing process	✓
idewait	Wait for disc controller to be ready	✓
idestart	tell disc controller to start I/O for the first buffer on idequeue	✓
ideinit	Initialize the disc controller	✓
ideintr	disk interrupt handler, transfer data from controller to buffer, wake up processes waiting for this buffer, start I/O for next buffer	✓

Your answer is correct.

The correct answer is: iderw → Issue a disk read/write for a buffer, block the issuing process, idewait → Wait for disc controller to be ready, idestart → tell disc controller to start I/O for the first buffer on idequeue, ideinit → Initialize the disc controller, ideintr → disk interrupt handler, transfer data from controller to buffer, wake up processes waiting for this buffer, start I/O for next buffer

Question 18

Correct

Mark 0.25 out of 0.25

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

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

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

Question **19**

Correct

Mark 0.25 out of 0.25

Which of the following call sequence is impossible in xv6?

- a. Process 1: write() -> sys_write()-> file_write() -- timer interrupt -> trap() -> yield() -> sched() -> switch() (jumps to)-> scheduler() -> swtch() (jumps to)-> Process 2 (return call sequence) sched() -> yield() -> trap-> user-code
- b. Process 1: write() -> sys_write()-> file_write() -> writei() -> bread() -> bget() -> iderw() -> sleep() -> sched() -> switch() (jumps to)-> scheduler() ->switch()(jumps to)-> Process 2 (return call sequence) sched() -> yield() -> trap-> user-code
- c. Process 1: timer interrupt -> trap() -> yield() -> sched() -> switch() -> scheduler()-> Process 2 runs -> write -> sys_write() -> trap()-> ... ✓

Your answer is correct.

The correct answer is: Process 1: timer interrupt -> trap() -> yield() -> sched() -> switch() -> scheduler()-> Process 2 runs -> write -> sys_write() -> trap()-> ...

Question **20**

Partially correct

Mark 0.17 out of 0.50

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

Started on Friday, 24 February 2023, 2:44 PM

State Finished

Completed on Friday, 24 February 2023, 4:39 PM

Time taken 1 hour 55 mins

Grade **8.78** out of 10.00 (**87.77%**)

Question **1**

Correct

Mark 0.50 out of 0.50

What is meant by formatting a disk/partition?

- a. erasing all data on the disk/partition
- b. storing all the necessary programs on the disk/partition
- c. creating layout of empty directory tree/graph data structure ✓
- d. writing zeroes on all sectors

The correct answer is: creating layout of empty directory tree/graph data structure

Question 2

Partially correct

Mark 0.44 out of 0.50

How does the compiler calculate addresses for the different parts of a C program, when paging is used?

Text	starting with 0	✓
Static variables	Immediately after the text, along with globals	✓
Local variables	An offset with respect to stack pointer (esp)	✓
#include files	No memory allocated, they are handled by linker	✗
#define	No memory allocated, they are handled by pre-processor	✓
typedef	No memory allocated, as they are not variables, but only conceptual definition of a type	✓
Global variables	Immediately after the text	✓
malloced memory	Heap (handled by the malloc-free library, using OS's system calls)	✓

Your answer is partially correct.

You have correctly selected 7.

The correct answer is: Text → starting with 0, Static variables → Immediately after the text, along with globals, Local variables → An offset with respect to stack pointer (esp), #include files → No memory allocated for the file, but if it contains variables, then variables may be allocated memory, #define → No memory allocated, they are handled by pre-processor, typedef → No memory allocated, as they are not variables, but only conceptual definition of a type, Global variables → Immediately after the text, malloced memory → Heap (handled by the malloc-free library, using OS's system calls)

Question 3

Correct

Mark 0.50 out of 0.50

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

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 4

Partially correct

Mark 0.43 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	
<input type="radio"/> ✗	<input checked="" type="radio"/>	The term "core" refers to the core code of the kernel.
<input type="radio"/> ✗	<input checked="" type="radio"/>	On Linux, the message is printed only because the memory management scheme is segmentation
<input checked="" type="radio"/>	<input type="radio"/> ✗	The process has definitely performed illegal memory access.
<input checked="" type="radio"/>	<input type="radio"/> ✗	On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.
<input type="radio"/> ✗	<input checked="" type="radio"/>	The illegal memory access was detected by the kernel and the process was punished by kernel.
<input checked="" type="radio"/>	<input type="radio"/> ✗	The core file can be analysed later using a debugger, to determine what went wrong.
<input checked="" type="radio"/>	<input type="radio"/> ✗	The image of the process is stored in a file called "core", if the ulimit allows so.

The term "core" refers to the core code of the kernel.: False

On Linux, the message is printed only because the memory management scheme is segmentation: False

The process has definitely performed illegal memory access.: True

On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.: True

The illegal memory access was detected by the kernel and the process was punished by kernel.: False

The core file can be analysed later using a debugger, to determine what went wrong.: True

The image of the process is stored in a file called "core", if the ulimit allows so.: True

Question 5

Correct

Mark 0.50 out of 0.50

Which of the following instructions should be privileged?

Select one or more:

- a. Access memory management unit of the processor ✓
- b. Turn off interrupts. ✓
- c. Read the clock.
- d. Set value of timer. ✓
- e. Switch from user to kernel mode. ✓ This instruction (like INT) is itself privileged - and that is why it not only changes the mode, but also ensures a jump to an ISR (kernel code)
- f. Access a general purpose register
- g. Set value of a memory location
- h. Modify entries in device-status table ✓
- i. Access I/O device. ✓

Your answer is correct.

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

Question 6

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	✓
Continuous allocation	need for compaction	✓
Linked allocation	Too many seeks	✓

Your answer is correct.

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

Question 7

Partially correct

Mark 0.43 out of 0.50

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[0][1] ✗ );
        close(0);
        dup( pfd[1][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 8

Partially correct

Mark 0.45 out of 0.50

Match the elements of C program to their place in memory

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

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

Question 9

Correct

Mark 0.50 out of 0.50

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

Write the answer as a number.

The correct answer is: 6

Question 10

Partially correct

Mark 0.42 out of 0.50

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

True	False	
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	It's possible that a particular hardware has multiple device drivers available for it.
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	Device driver is part of OS code
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	Device driver is an intermediary between the hardware controller and OS
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	Device driver is part of hardware
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✗	Writing a device driver mandatorily demands reading the technical documentation about the hardware.

Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.: False

It's possible that a particular hardware has multiple device drivers available for it.: True

Device driver is part of OS code: True

Device driver is an intermediary between the hardware controller and OS: True

Device driver is part of hardware: False

Writing a device driver mandatorily demands reading the technical documentation about the hardware.: True

Question 11

Correct

Mark 0.50 out of 0.50

Map each signal with its meaning

SIGSEGV	Invalid Memory Reference	✓
SIGALRM	Timer Signal from alarm()	✓
SIGPIPE	Broken Pipe	✓
SIGUSR1	User Defined Signal	✓
SIGCHLD	Child Stopped or Terminated	✓

The correct answer is: SIGSEGV → Invalid Memory Reference, SIGALRM → Timer Signal from alarm(), SIGPIPE → Broken Pipe, SIGUSR1 → User Defined Signal, SIGCHLD → Child Stopped or Terminated

Question 12

Partially correct

Mark 0.25 out of 0.50

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"

Block bitmap(s) for all the blocks of the file

 Yes ✓

Data blocks of the file

 Yes ✗

One or multiple data blocks of the parent directory

 Yes ✗

Superblock

 Yes ✓

Possibly one block bitmap corresponding to the parent directory

 Yes ✓

One or more data bitmap blocks for the parent directory

 Yes ✗

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 3.

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

Question 13

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)

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

Your answer is correct.

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

Question 14

Partially correct

Mark 0.30 out of 0.50

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

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

Named pipe exists as a file.: True

Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.: True

Both types of pipes are an extension of the idea of "message passing": True

A named pipe has a name decided by the kernel.: False

Un-named pipes are inherited by a child process from parent.: True

Named pipes can exist beyond the life-time of processes using them.: 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 pipes can be used for communication between only "related" processes.: False

The pipe() system call can be used to create either a named or un-named pipe.: False

Question 15

Partially correct

Mark 0.29 out of 0.50

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

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

The kernel refers to the page table for converting each virtual address to physical address.: False

When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.: True

The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.: False

The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.: False

The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.: True

When a process is executing, each virtual address is converted into physical address by the kernel directly.: False

The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True

Question 16

Correct

Mark 0.50 out of 0.50

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

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

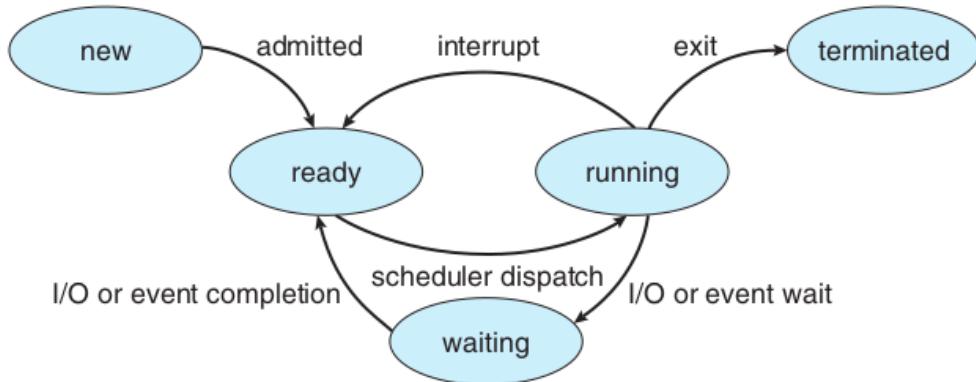


Figure 3.2 Diagram of process state.

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	Every forked process has to go through ZOMBIE state, at least for a small duration.
<input type="radio"/>	<input checked="" type="radio"/>	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first
<input checked="" type="radio"/>	<input type="radio"/>	Only a process in READY state is considered by scheduler
<input checked="" type="radio"/>	<input type="radio"/>	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet
<input type="radio"/>	<input checked="" type="radio"/>	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.

Every forked process has to go through ZOMBIE state, at least for a small duration.: True

A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False

Only a process in READY state is considered by scheduler: True

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

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

Question 17

Partially correct

Mark 0.45 out of 0.50

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

Select No otherwise.

Yes	No	
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Memory management information about that process
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Pointer to IDT
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	PID of Init
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	PID
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Function pointers to all system calls
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	EIP at the time of context switch
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Process state
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	List of opened files
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Process context
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Pointer to the parent process

Memory management information about that process: Yes

Pointer to IDT: No

PID of Init: No

PID: Yes

Function pointers to all system calls: No

EIP at the time of context switch: Yes

Process state: Yes

List of opened files: Yes

Process context: Yes

Pointer to the parent process: Yes

Question **18**

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

Answer: hi one one



The correct answer is: hi one one

Question **19**

Correct

Mark 0.50 out of 0.50

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

Select one:

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

Your answer is correct.

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

Question **20**

Partially correct

Mark 0.33 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- a. pointer dereference
- b. #directives✓
- c. typedefs✓
- d. global variables
- e. function calls
- f. expressions
- g. local variable declaration

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: #directives, typedefs, global variables

[◀ Quiz-1 Preparation questions](#)

Jump to...

[Quiz - 2 \(17 March 2023\) ►](#)

Started on	Saturday, 12 February 2022, 10:01:48 AM
State	Finished
Completed on	Saturday, 12 February 2022, 11:56:09 AM
Time taken	1 hour 54 mins
Grade	5.82 out of 10.00 (58%)

Question 1

Complete

Mark 0.75 out of 1.00

Select the correct statements about interrupt handling in xv6 code

- a. The trapframe pointer in struct proc, points to a location on user stack
- b. On any interrupt/syscall/exception the control first jumps in vectors.S
- c. On any interrupt/syscall/exception the control first jumps in trapasm.S
- d. Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt
- e. xv6 uses the 0x64th entry in IDT for system calls
- f. The CS and EIP are changed only after pushing user code's SS,ESP on stack
- g. The trapframe pointer in struct proc, points to a location on kernel stack
- h. All the 256 entries in the IDT are filled

- i. xv6 uses the 64th entry in IDT for system calls
- j. The CS and EIP are changed only immediately on a hardware interrupt
- k. The function trap() is called only in case of hardware interrupt
- l. Before going to alltraps, the kernel stack contains upto 5 entries.
- m. The function trap() is called irrespective of hardware interrupt/system-call/exception

The correct answers are: All the 256 entries in the IDT are filled, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on kernel stack, The function trap() is called irrespective of hardware interrupt/system-call/exception, The CS and EIP are changed only after pushing user code's SS,ESP on stack

Question 2

Complete

Mark 0.00 out of 1.00

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

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

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

Select one or more:

a. P1 running

P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running

b. P1 running

P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running

c. P1 running

P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

d. P1 running

keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
scheduler
P2 running

e.

P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

f. P1 running

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

The correct answers are: P1 running

P1 makes system call and blocks
 Scheduler
 P2 running
 P2 makes system call and blocks
 Scheduler
 P1 running again, P1 running
 P1 makes system call
 timer interrupt
 Scheduler
 P2 running
 timer interrupt
 Scheuler
 P1 running
 P1's system call return,
 P1 running
 P1 makes system call
 Scheduler
 P2 running
 P2 makes system call and blocks
 Scheduler
 P1 running again

Question 3

Complete

Mark 0.00 out of 0.50

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

Select all the appropriate choices

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

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

Question 4

Complete

Mark 0.13 out of 0.50

Select all the correct statements about zombie processes

Select one or more:

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

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

Question 5

Complete

Mark 0.50 out of 0.50

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

Process P1 is running

6	◆
---	---

timer interrupt occurs

2	◆
---	---

context of P1 is loaded from P1's PCB

4	◆
---	---

context of P0 is saved in P0's PCB

3	◆
---	---

Process P0 is running

1	◆
---	---

Control is passed to P1

5	◆
---	---

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

Question 6

Complete

Mark 0.50 out of 0.50

Suppose a program does a `scanf()` call.

Essentially the `scanf` does a `read()` system call.

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

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

Select one:

- a. `read()` will return and process will be taken to a wait queue
- 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. OS code for `read()` will call scheduler
- e. `read()` returns and process calls scheduler()

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

Question 7

Complete

Mark 0.50 out of 0.50

Consider the following programs

exec1.c

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

exec2.c

```
#include <unistd.h>
#include <stdio.h>
int main() {
    exec("/bin/ls", "/bin/ls", NULL);
    printf("hello\n");
}
```

Compiled as

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

And run as

```
$ ./exec1
```

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

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

Select one:

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

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

Question 8

Complete

Mark 0.25 out of 0.50

The bootmain() function has this code

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

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

In these statements 0x1000 is referred to as ADDRESS

True False

<input type="radio"/>	<input checked="" type="radio"/>	This line effectively loads the ELF header and the program headers at ADDRESS
-----------------------	----------------------------------	---

<input type="radio"/>	<input checked="" type="radio"/>	If the value of ADDRESS is changed, then the program will not work
-----------------------	----------------------------------	--

<input type="radio"/>	<input checked="" type="radio"/>	If the value of ADDRESS is changed to a higher number (upto a limit), the program could still work
-----------------------	----------------------------------	--

<input type="radio"/>	<input checked="" type="radio"/>	If the value of ADDRESS is changed to a lower number (upto a limit), the program could still work
-----------------------	----------------------------------	---

<input type="radio"/>	<input checked="" type="radio"/>	The value ADDRESS is changed to a 0 the program could still work
-----------------------	----------------------------------	--

<input checked="" type="radio"/>	<input type="radio"/>	This line loads the kernel code at ADDRESS
----------------------------------	-----------------------	--

This line effectively loads the ELF header and the program headers at ADDRESS: False

If the value of ADDRESS is changed, then the program will not work: False

If the value of ADDRESS is changed to a higher number (upto a limit), the program could still work: True

If the value of ADDRESS is changed to a lower number (upto a limit), the program could still work: True

The value ADDRESS is changed to a 0 the program could still work: False

This line loads the kernel code at ADDRESS: False

Question 9

Complete

Mark 0.80 out of 1.00

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

True**False**

The work of selecting and scheduling a process is done only in `scheduler()` and not in `sched()`

The variable `c->scheduler` on first processor uses the stack allocated entry.S

`sched()` and `scheduler()` are co-routines

the control returns to `switchkvm();` after `swtch(&(c->scheduler), p->context);` in `scheduler()`

When a process is scheduled for execution, it resumes execution in `sched()` after the call to `swtch()`

`swtch` is a function that saves old context, loads new context, and returns to last EIP in the new context

`sched()` calls `scheduler()` and `scheduler()` calls `sched()`

`swtch` is a function that does not return to the caller

The function `scheduler()` executes using the kernel-only stack

the control returns to `mycpu()->intena = intena; ()`; after `swtch(&p->context, mycpu()->scheduler);` in `sched()`

The work of selecting and scheduling a process is done only in `scheduler()` and not in `sched()`: True

The variable `c->scheduler` on first processor uses the stack allocated entry.S: True

`sched()` and `scheduler()` are co-routines: True

the control returns to `switchkvm();` after `swtch(&(c->scheduler), p->context);` in `scheduler()`: False

When a process is scheduled for execution, it resumes execution in `sched()` after the call to `swtch()`

: True

`swtch` is a function that saves old context, loads new context, and returns to last EIP in the new context: True

`sched()` calls `scheduler()` and `scheduler()` calls `sched()`: False

`swtch` is a function that does not return to the caller: True

The function `scheduler()` executes using the kernel-only stack: True

the control returns to `mycpu()->intena = intena; ()`; after `swtch(&p->context, mycpu()->scheduler);` in `sched()`:

False

Question 10

Complete

Mark 0.36 out of 0.50

Order the events that occur on a timer interrupt:

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

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

Question 11

Complete

Mark 0.20 out of 0.50

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

`jmp *%eax`
in entry.S

The 4KB area in kernel image, loaded in memory, named as 'stack' ◆

`cli`
in bootasm.S

Immaterial as the stack is not used here ◆

`readseg((uchar*)elf, 4096, 0);`
in bootmain.c

0x10000 to 0x7c00 ◆

`call bootmain`
in bootasm.S

0x7c00 to 0x10000 ◆

`ljmp $(SEG_KCODE<<3), $start32`
in bootasm.S

0x7c00 to 0 ◆

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

Question 12

Complete

Mark 0.50 out of 0.50

What's the trapframe in xv6?

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

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

Question 13

Complete

Mark 0.50 out of 0.50

In bootasm.S, on the line

```
ljmp      $(SEG_KCODE<<3), $start32
```

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

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

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

Question 14

Complete

Mark 0.43 out of 0.50

Select all the correct statements about code of bootmain() in xv6

```

void
bootmain(void)
{
    struct elfhdr *elf;
    struct proghdr *ph, *eph;
    void (*entry) (void);
    uchar* pa;

    elf = (struct elfhdr*)0x10000; // scratch space

    // Read 1st page off disk
    readseg((uchar*)elf, 4096, 0);

    // Is this an ELF executable?
    if(elf->magic != ELF_MAGIC)
        return; // let bootasm.S handle error

    // Load each program segment (ignores ph flags).
    ph = (struct proghdr*)((uchar*)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++) {
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    }

    // Call the entry point from the ELF header.
    // Does not return!
    entry = (void(*)(void))(elf->entry);
    entry();
}

```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

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

The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question 15

Complete

Mark 0.40 out of 0.50

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

Select No otherwise.

Yes**No**

<input type="radio"/>	<input checked="" type="radio"/> PID of Init	
<input type="radio"/>	<input checked="" type="radio"/> Pointer to IDT	
<input checked="" type="radio"/>	<input type="radio"/> EIP at the time of context switch	
<input type="radio"/>	<input checked="" type="radio"/> Pointer to the parent process	
<input checked="" type="radio"/>	<input type="radio"/> PID	
<input checked="" type="radio"/>	<input type="radio"/> Process state	
<input checked="" type="radio"/>	<input type="radio"/> Function pointers to all system calls	
<input checked="" type="radio"/>	<input type="radio"/> Process context	
<input type="radio"/>	<input type="radio"/> List of opened files	
<input checked="" type="radio"/>	<input type="radio"/> Memory management information about that process	

PID of Init: No

Pointer to IDT: No

EIP at the time of context switch: Yes

Pointer to the parent process: Yes

PID: Yes

Process state: Yes

Function pointers to all system calls: No

Process context: Yes

List of opened files: Yes

Memory management information about that process: Yes

Question 16

Not answered

Marked out of 1.00

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

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

Maximum two points to be written.

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

Jump to...



(Code) IPC - Shm, Messages ►

Started on Tuesday, 22 March 2022, 2:06:46 PM

State Finished

Completed on Tuesday, 22 March 2022, 5:29:11 PM

Time taken 3 hours 22 mins

Grade 23.12 out of 40.00 (58%)

Question 1

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

This sequence of events is: possible ✗

Because

Process P2 has to return from interrupt context before it's user code executes ✓

Question 2

Correct

Mark 1.00 out of 1.00

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

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



Your answer is correct.

The correct answer is: Singly linked NULL terminated list

Question 3

Incorrect

Mark 0.00 out of 1.00

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

Consider each request as first request.

220 KB, best fit	100 KB	▼	✗
50 KB, worst fit	100 KB	▼	✗
100 KB, worst fit	100 KB	▼	✗
150 KB, best fit	100 KB	▼	✗
200 KB, first fit	100 KB	▼	✗
150 KB, first fit	100 KB	▼	✗

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

Question 4

Incorrect

Mark 0.00 out of 1.00

Select the most common causes of use of IPC by processes

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

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

Question 5

Partially correct

Mark 0.33 out of 1.00

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

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

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

Question 6

Correct

Mark 1.00 out of 1.00

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

10011010

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

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

Answer: ✓

The correct answer is: 11111010

Question 7

Partially correct

Mark 0.67 out of 1.00

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

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

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

Question 8

Partially correct

Mark 0.67 out of 1.00

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

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 9

Correct

Mark 2.00 out of 2.00

For the reference string

3 4 3 5 2

using FIFO replacement policy for pages,

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

Select the correct statement.

Select one:

- a. Exhibit Balady's anomaly between 3 and 4 frames
- b. Do not exhibit Balady's anomaly
- c. Exhibit Balady's anomaly between 2 and 3 frames



Your answer is correct.

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

Question 10

Partially correct

Mark 0.60 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

<input checked="" type="radio"/>	<input checked="" type="radio"/>	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).	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Local replacement results in more predictable per-process completion time because number of page faults can be better predicted.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input checked="" type="radio"/>	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.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Global replacement may give highly variable per process completion time because number of page faults become unpredictable.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input checked="" type="radio"/>	Local replacement can lead to under-utilisation of memory, because a process may not use all the pages allocated to it all the time.	<input checked="" type="checkbox"/>

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

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

Question 11

Partially correct

Mark 0.60 out of 1.00

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

True**False**

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

mmap() solves the problem of thrashing.: False

Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.: True

Thrashing can be limited if local replacement is used.: True

During thrashing the CPU is under-utilised as most time is spent in I/O: True

Thrashing is particular to demand paging systems, and does not apply to pure paging systems.: True

Thrashing occurs when the total size of all processes's locality exceeds total memory size.: True

Thrashing can occur even if entire memory is not in use.: False

Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.: False

The working set model is an attempt at approximating the locality of a process.: True

Thrashing occurs because some process is doing lot of disk I/O.: False

Question 12

Incorrect

Mark 0.00 out of 1.00

Select all the correct statements about signals

Select one or more:

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

Your answer is incorrect.

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

Question 13

Incorrect

Mark 0.00 out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

Your answer is incorrect.

The correct answers are: All the 256 entries in the IDT are filled, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on kernel stack, The function trap() is the called irrespective of hardware interrupt/system-call/exception, The CS and EIP are changed only after pushing user code's SS,ESP on stack

Question 14

Correct

Mark 1.00 out of 1.00

For the reference string

3 4 3 5 2

using LRU replacement policy for pages,

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

Select the most correct statement.

Select one:

- a. Exhibit Balady's anomaly between 2 and 3 frames
- b. Exhibit Balady's anomaly between 3 and 4 frames
- c. LRU will never exhibit Balady's anomaly
- d. This example does not exhibit Balady's anomaly



Your answer is correct.

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

Question 15

Partially correct

Mark 0.10 out of 1.00

Select all the correct statements about process states.

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

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

Your answer is partially correct.

You have correctly selected 2.

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

Question 16

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

The correct answer is: end, P2V(PHYSTOP)

Question 17

Incorrect

Mark 0.00 out of 1.00

If one thread opens a file with read privileges then

Select one:

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

Your answer is incorrect.

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

Question 18

Correct

Mark 1.00 out of 1.00

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

Setup kernel part of a page table, and switch to that page table

kvmalloc()



mappages()



setupkvm()



walkpgdir()



kmap[]



kalloc()



Setup kernel part of a page table, mapping kernel code, data, read-only data, I/O space, devices

Return address of page table entry in a given page directory, for a given virtual address; creates page table if necessary

Array listing the kernel memory mappings, to be used by setupkvm()

return a free page, if available; 0, otherwise

Your answer is correct.

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

Question 19

Partially correct

Mark 0.63 out of 1.00

Consider a demand-paging system with the following time-measured utilizations:

CPU utilization : 20%

Paging disk: 97.7%

Other I/O devices: 5%

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

a. Install a faster CPU : Yes

b. Install a bigger paging disk. : Yes

c. Increase the degree of multiprogramming. : Yes

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 20

Not answered

Marked 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:



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

Question 21

Correct

Mark 1.00 out of 1.00

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

Write answer as a decimal number.

A conventional, single-level page table:

1048576



An inverted page table:

131072



Question 22

Partially correct

Mark 0.38 out of 1.00

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

True**False**

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

String arguments are first copied to trapframe and then from trapframe to kernel's other variables.: False

The arguments to system call originally reside on process stack.: True

String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer: True

The arguments to system call are copied to kernel stack in trapasm.S: False

The arguments are accessed in the kernel code using esp on the trapframe.: True

Integer arguments are copied from user memory to kernel memory using argint(): True

The functions like argint(), argstr() make the system call arguments available in the kernel.: True

Integer arguments are stored in eax, ebx, ecx, etc. registers: False

Question 23

Partially correct

Mark 0.60 out of 1.00

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

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.

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

Question 24

Correct

Mark 1.00 out of 1.00

After virtual memory is implemented

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

True**False**

<input checked="" type="radio"/>	<input type="radio"/> X	Cumulative size of all programs can be larger than physical memory size	<input checked="" type="checkbox"/>	
<input type="radio"/> X	<input checked="" type="radio"/>	Virtual addresses become available to executing process	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/> X	Code need not be completely in memory	<input checked="" type="checkbox"/>	
<input type="radio"/> X	<input checked="" type="radio"/>	Virtual access to memory is granted to all processes	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/> X	Relatively less I/O may be possible during process execution	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/> X	Logical address space could be larger than physical address space	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/> X	One Program's size can be larger than physical memory size	<input checked="" type="checkbox"/>	

Cumulative size of all programs can be larger than physical memory size: True

Virtual addresses become available to executing process: False

Code need not be completely in memory: True

Virtual access to memory is granted to all processes: False

Relatively less I/O may be possible during process execution: True

Logical address space could be larger than physical address space: True

One Program's size can be larger than physical memory size: True

Question 25

Partially correct

Mark 0.50 out of 1.00

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

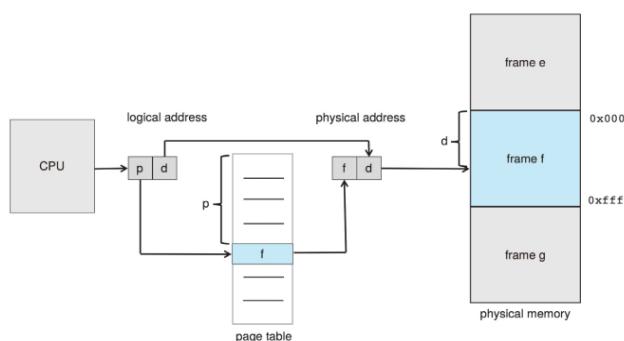
- a. POSIX shared memory is newer than System V shared memory
- b. POSIX allows giving name to shared memory, System V does not
- c. System V is more prevalent than POSIX even today
- d. POSIX shared memory is "thread safe", System V is not

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

Question 26

Partially correct

Mark 0.50 out of 1.00

**Figure 9.8** Paging hardware.

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

True False

<input checked="" type="radio"/>	<input type="radio"/>	The logical address issued by CPU is the same one generated by compiler	✓	
<input checked="" type="radio"/>	<input type="radio"/>	The page table is in physical memory and must be continuous	✓	
<input checked="" type="radio"/>	<input type="radio"/>	The combining of f and d is done by MMU	✗	
<input type="radio"/>	<input checked="" type="radio"/>	Using the offset d in the physical page-frame is done by MMU	✓	
<input checked="" type="radio"/>	<input type="radio"/>	The split of logical address into p and d is done by MMU	✗	
<input type="radio"/>	<input checked="" type="radio"/>	There are total 3 memory references in this diagram	✗	

The logical address issued by CPU is the same one generated by compiler: True

The page table is in physical memory and must be continuous: True

The combining of f and d is done by MMU: True

Using the offset d in the physical page-frame is done by MMU: False

The split of logical address into p and d is done by MMU: True

There are total 3 memory references in this diagram: False

Question 27

Partially correct

Mark 0.60 out of 1.00

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

True**False**

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

The pipe() system call can be used to create either a named or un-named pipe.: False

Named pipes can be used for communication between only "related" processes.: False

Un-named pipes 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

Both types of pipes are an extension of the idea of "message passing": True

Named pipe exists as a file: True

Both types of pipes provide FIFO communication.: True

Un-named pipes are inherited by a child process from parent.: True

Named pipes can exist beyond the life-time of processes using them.: True

The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.: False

Question 28

Partially correct

Mark 0.20 out of 2.00

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

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

Your answer is partially correct.

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

Grade details: 2 / 20 = 10%

Here are the scores for each item in this response:

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

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

The correct order for these items is as follows:

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

Question 29

Correct

Mark 2.00 out of 2.00

Consider the reference string

6 4 2 0 1 2 6 9 2 0 5

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

Answer: 10



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

The correct answer is: 10

Question 30

Correct

Mark 2.00 out of 2.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB only Mark statements True or False

True False

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

xv6 uses physical memory upto 224 MB only: True

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True

The stack allocated in entry.S is used as stack for scheduler's context for first processor: True

The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

The free page-frame are created out of nearly 222 MB: True

The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True

The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False

PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True

The kernel code and data take up less than 2 MB space: True

Question 31

Partially correct

Mark 0.60 out of 1.00

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

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

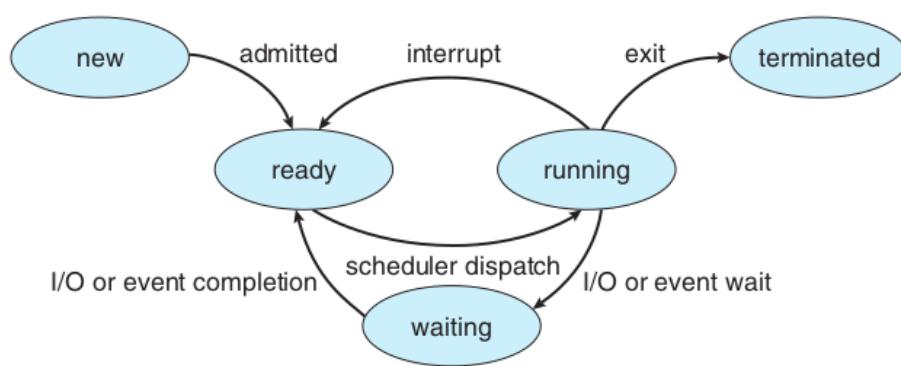


Figure 3.2 Diagram of process state.

True False

<input checked="" type="radio"/>	<input type="radio"/>	Only a process in READY state is considered by scheduler	<input checked="" type="checkbox"/>	
<input type="radio"/>	<input checked="" type="radio"/>	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first	<input type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/>	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/>	<input type="radio"/>	Every forked process has to go through ZOMBIE state, at least for a small duration.	<input checked="" type="checkbox"/>	
<input type="radio"/>	<input checked="" type="radio"/>	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.	<input type="checkbox"/>	

Only a process in READY state is considered by scheduler: True

A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

Every forked process has to go through ZOMBIE state, at least for a small duration.: True

A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.: False

Question 32

Partially correct

Mark 0.89 out of 2.00

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

Copy the code pages of a process	copyuvvm()	✗
Setup and load the user page table for initcode process	setupkvm()	✗
Load contents from ELF into existing pages	loaduvvm()	✓
Create a copy of the page table of a process	clearpteul()	✗
Switch to kernel page table	switchuvvm()	✗
Mark the page as in-accessible	clearpteul()	✓
Switch to user page table	switchuvvm()	✓
setup the kernel part in the page table	setupkvm()	✓
Load contents from ELF into pages after allocating the pages first	loaduvvm()	✗

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

Question 33

Correct

Mark 1.00 out of 1.00

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

Select one or more:

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

Your answer is correct.

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

Question 34

Partially correct

Mark 0.67 out of 1.00

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

True**False**

<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/> <input type="checkbox"/>	MMap_FIXED guarantees that the mapping is always done at the specified address	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	mmap() results in changes to page table of a process.	<input checked="" type="checkbox"/>	
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	mmap() can be implemented on both demand paged and non-demand paged systems.	<input checked="" type="checkbox"/>	
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	mmap() results in changes to buffer-cache of the kernel.	<input checked="" type="checkbox"/>	
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/> <input type="checkbox"/>	on failure mmap() returns NULL	<input checked="" type="checkbox"/>	
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	MAP_SHARED leads to a mapping that is copy-on-write	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	MAP_PRIVATE leads to a mapping that is copy-on-write	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	mmap() is a system call	<input checked="" type="checkbox"/>	
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	on failure mmap() returns (void *)-1	<input checked="" type="checkbox"/>	

MAP_FIXED guarantees that the mapping is always done at the specified address.: False

mmap() results in changes to page table of a process.: True

mmap() can be implemented on both demand paged and non-demand paged systems.: True

mmap() results in changes to buffer-cache of the kernel.: False

on failure mmap() returns NULL: False

MAP_SHARED leads to a mapping that is copy-on-write: False

MAP_PRIVATE leads to a mapping that is copy-on-write: True

mmap() is a system call: True

on failure mmap() returns (void *)-1: True

Question 35

Partially correct

Mark 0.10 out of 1.00

Select all the correct statements about linking and loading.

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: Continuous memory management schemes can support static linking and static loading. (may be inefficiently), Continuous memory management schemes can support static linking and dynamic loading. (may be inefficiently), Dynamic linking essentially results in relocatable code., Loader is part of the operating system, Dynamic linking and loading is not possible without demand paging or demand segmentation.

[◀ \(Optional Assignment\) Iseek system call in xv6](#)

Jump to...



[Feedback on Quiz-2 ►](#)

Started on Monday, 16 January 2023, 9:00 PM

State Finished

Completed on Monday, 16 January 2023, 10:07 PM

Time taken 1 hour 6 mins

Grade 13.25 out of 15.00 (88.33%)

Question **1**

Correct

Mark 1.00 out of 1.00

When you turn your computer ON, you are often shown an option like "Press F9 for boot options". What does this mean?

- a. The choice of booting slowly or fast
- b. The choice of which OS to boot from
- c. The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded ✓
- d. The choice of the boot loader (e.g. GRUB or Windows-Loader)

The correct answer is: The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded

Question **2**

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.

- a. Modern Bootloaders often allow configuring the way an OS boots ✓
- b. Bootloader must be one sector in length
- c. The bootloader loads the BIOS
- d. Bootloaders allow selection of OS to boot from ✓
- 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 3

Correct

Mark 2.00 out of 2.00

What will this program do?

```
int main() {  
    fork();  
    execl("/bin/ls", "/bin/ls", NULL);  
    printf("hello");  
}
```

- a. one process will run ls, another will print hello
- b. run ls once
- c. run ls twice and print hello twice
- d. run ls twice and print hello twice, but output will appear in some random order
- e. run ls twice✓

Your answer is correct.

The correct answer is: run ls twice

Question 4

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

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

Question 5

Partially correct

Mark 0.50 out of 1.00

Select the correct statements about hard and soft links

Select one or more:

- a. Soft link shares the inode of actual file
- b. Deleting a hard link deletes the file, only if link count was 1 ✓
- c. Soft links increase the link count of the actual file inode
- d. Soft links can span across partitions while hard links can't ✓
- e. Deleting a soft link deletes only the actual file
- f. Deleting a hard link always deletes the file ✗
- g. Hard links can span across partitions while soft links can't
- h. Deleting a soft link deletes the link, not the actual file ✓
- i. Hard links increase the link count of the actual file inode
- j. Deleting a soft link deletes both the link and the actual file
- k. Hard links share the inode ✓
- l. Hard links enforce separation of filename from its metadata in on-disk data structures.

Your answer is partially correct.

You have correctly selected 4.

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

Question **6**

Partially correct

Mark 1.75 out of 3.00

Select correct statements about mounting

Select one or more:

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

Your answer is partially correct.

You have correctly selected 5.

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 **7**

Correct

Mark 1.00 out of 1.00

Compare multiprogramming with multitasking

- a. A multitasking system is not necessarily multiprogramming
- b. A multiprogramming system is not necessarily multitasking ✓

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

Question **8**

Correct

Mark 1.00 out of 1.00

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

- a. yes
- b. no ✓ wrong

The correct answer is: no

Question 9

Correct

Mark 1.00 out of 1.00

Consider the following programs

[exec1.c](#)

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

[exec2.c](#)

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

Compiled as

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

And run as

[\\$./exec1](#)

Explain the output of the above command ([./exec1](#))

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

Select one:

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

Your answer is correct.

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

Question 10

Correct

Mark 1.00 out of 1.00

A process blocks itself means

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

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

Question 11

Correct

Mark 1.00 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

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

Your answer is correct.

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

Question **12**

Correct

Mark 1.00 out of 1.00

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

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

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

[◀ Random Quiz - 1 \(Pre-Requisite Quiz\)](#)

Jump to...

[Random Quiz - 3 \(processes, memory management, event driven kernel\), compilation-linking-loading, ipc-pipes ►](#)

Started on Thursday, 2 February 2023, 9:01 PM

State Finished

Completed on Thursday, 2 February 2023, 10:55 PM

Time taken 1 hour 53 mins

Grade 16.46 out of 20.00 (82.28%)

Question 1

Complete

Mark 1.00 out of 1.00

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

- a. Check the program for logical errors
- b. Convert high level language code to machine code
- c. Invoke the linker to link the function calls with their code, extern globals with their declaration
- d. Suggest alternative pieces of code that can be written
- e. Check the program for syntactical errors
- f. Process the # directives in a C program

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

Question 2

Complete

Mark 1.00 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
int main(int argc, char *argv[]) {  
    int fd1, fd2 = 1, fd3 = 1, fd4 = 1;  
  
    fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    fd2 = open("/tmp/2", O_RDONLY);  
    fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    close(0);  
    close(1);  
    dup(fd2);  
    dup(fd3);  
    close(fd3);  
    dup2(fd2, fd4);  
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);  
    return 0;  
}
```

fd4	/tmp/2
fd2	/tmp/2
1	/tmp/3
fd1	/tmp/1
fd3	closed
0	/tmp/2
2	stderr

The correct answer is: fd4 → /tmp/2, fd2 → /tmp/2, 1 → /tmp/3, fd1 → /tmp/1, fd3 → closed, 0 → /tmp/2, 2 → stderr

Question 3

Complete

Mark 0.75 out of 1.00

Select the sequence of events that are NOT possible, assuming an interruptible kernel code

Select one or more:

- a. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
scheduler
P2 running
- b. P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- c. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running
- d. P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again
- e. P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

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

The correct answers are: P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again,

P1 running

P1 makes system call

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

Question 4

Complete

Mark 0.00 out of 1.00

Select all the correct statements about named pipes and ordinary(unnamed) pipe

Select one or more:

- a. named pipes can be used between multiple processes but ordinary pipes can not be used
- b. named pipe can be used between any processes
- c. both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes
- d. named pipes are more efficient than ordinary pipes
- e. ordinary pipe can only be used between related processes
- f. a named pipe exists as a file on the file system
- g. named pipe exists even if the processes using it do exit()

The correct answers are: ordinary pipe can only be used between related processes, named pipe can be used between any processes, a named pipe exists as a file on the file system, named pipe exists even if the processes using it do exit(), both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes

Question 5

Complete

Mark 4.50 out of 5.00

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", " -l ", NULL);
    }
    pipe( pfd[1] );
    pid2 = fork();
    if(pid2 == 0) {
        close( pfd[0][1] );
        close(0);
        dup( pfd[1][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 6

Complete

Mark 1.00 out of 1.00

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)

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

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

Question 7

Complete

Mark 1.40 out of 2.00

Match the elements of C program to their place in memory

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

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

Question 8

Complete

Mark 0.63 out of 1.00

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

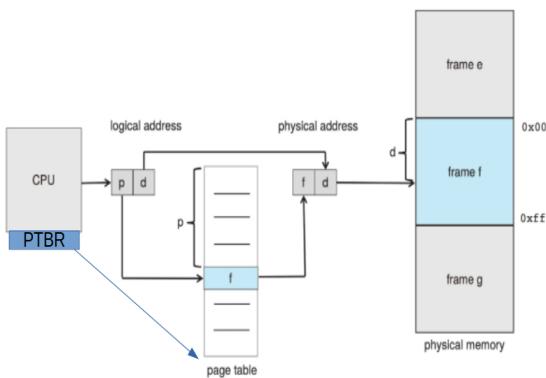


Figure 9.8 Paging hardware.

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

True False

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

The physical address may not be of the same size (in bits) as the logical address: True

The locating of the page table using PTBR also involves paging translation: False

Size of page table is always determined by the size of RAM: False

The page table is indexed using frame number: False

The PTBR is present in the CPU as a register: True

The page table is indexed using page number: True

Maximum Size of page table is determined by number of bits used for page number: True

The page table is itself present in Physical memory: True

Question 9

Complete

Mark 0.67 out of 1.00

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

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

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

Select one or more:

- a. P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P3 running

Hardware interrupt

Interrupt unblocks P1

Interrupt returns

P3 running

Timer interrupt

Scheduler

P1 running

- b. P1 running

P1 makes system call

system call returns

P1 running

timer interrupt

Scheduler running

P2 running

- c. P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

- d. P1 running

keyboard hardware interrupt

keyboard interrupt handler running

interrupt handler returns

P1 running

P1 makes system call

system call returns

P1 running

timer interrupt

scheduler

P2 running

- e. P1 running

P1 makes system call

timer interrupt

Scheduler

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

f.

P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

The correct answers are: P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again, P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheuler

P1 running

P1's system call return,

P1 running

P1 makes system call

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

Question 10

Complete

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 programs are correct
- b. Program 2 is correct for > /tmp/ddd but not for 2>&1
- c. Program 1 is correct for > /tmp/ddd but not for 2>&1
- d. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd
- e. Program 2 makes sure that there is one file offset used for '2' and '1'
- f. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd
- g. Only Program 2 is correct
- h. Program 1 does 1>&2
- i. Program 1 makes sure that there is one file offset used for '2' and '1'
- j. Program 2 does 1>&2
- k. Both program 1 and 2 are incorrect
- l. Only Program 1 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 **11**

Complete

Mark 0.71 out of 1.00

Order the events that occur on a timer interrupt:

Jump to a code pointed by IDT

2

Save the context of the currently running process

3

Execute the code of the new process

6

Jump to scheduler code

4

Change to kernel stack of currently running process

1

Select another process for execution

5

Set the context of the new process

7

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

Question **12**

Complete

Mark 0.80 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

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

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

Question 13

Complete

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

New: RunningReady : WaitingRunning: : None of theseWaiting: Running**Question 14**

Complete

Mark 1.00 out of 1.00

Select the order in which the various stages of a compiler execute.

Pre-processing 1Intermediate code generation 3Linking 4Syntactical Analysis 2Loading does not exist

The correct answer is: Pre-processing → 1, Intermediate code generation → 3, Linking → 4, Syntactical Analysis → 2, Loading → does not exist

Question 15

Complete

Mark 1.00 out of 1.00

A process blocks itself means

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

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

Jump to...

[Random Quiz 4 : Scheduling, signals, segmentation, paging, compilation, process-state ►](#)

Started on Thursday, 9 March 2023, 6:20 PM

State Finished

Completed on Thursday, 9 March 2023, 7:30 PM

Time taken 1 hour 10 mins

Overdue 14 mins

Grade **5.55** out of 10.00 (**55.53%**)

Question 1

Partially correct

Mark 0.18 out of 1.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB only
Mark statements True or False

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

xv6 uses physical memory upto 224 MB only: True

The stack allocated in entry.S is used as stack for scheduler's context for first processor: True

The free page-frame are created out of nearly 222 MB: True

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True

PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True

The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

The kernel code and data take up less than 2 MB space: True

The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False

The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True

Question **2**

Correct

Mark 1.00 out of 1.00

What's the trapframe in xv6?

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

Your answer is correct.

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

Question 3

Partially correct

Mark 0.75 out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

Your answer is partially correct.

You have correctly selected 6.

The correct answers are: All the 256 entries in the IDT are filled in xv6 code, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on process's kernel stack, The function trap() is the called even if any of the hardware interrupt/system-call/exception occurs, The CS and EIP are changed only after pushing user code's SS,ESP on stack

Question 4

Partially correct

Mark 0.55 out of 1.00

Consider the following command and its output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

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

kernel: $(OBJS) entry.o entryother initcode kernel.ld
    $(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
    $(OBJDUMP) -S kernel > kernel.asm
    $(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- a. Althought the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- b. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not X read as it is user programs.
- c. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✓
- d. The kernel.asm file is the final kernel file
- e. Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.
- f. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- g. The kernel.ld file contains instructions to the linker to link the kernel properly ✓
- h. readseg() reads first 4k bytes of kernel in memory ✓
- i. The bootmain() code does not read the kernel completely in memory

Your answer is partially correct.

You have correctly selected 4.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(), readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Althought the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question 5

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

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

Question 6

Correct

Mark 1.00 out of 1.00

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

Select all the appropriate choices

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

Your answer is correct.

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

Question 7

Incorrect

Mark 0.00 out of 1.00

xv6.img: bootblock kernel

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

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

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

Your answer is incorrect.

The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

Question 8

Partially correct

Mark 0.07 out of 1.00

Select all the correct statements about code of bootmain() in xv6

```
void
bootmain(void)
{
    struct elfhdr *elf;
    struct proghdr *ph, *eph;
    void (*entry) (void);
    uchar* pa;

    elf = (struct elfhdr*)0x10000; // scratch space

    // Read 1st page off disk
    readseg((uchar*)elf, 4096, 0);

    // Is this an ELF executable?
    if(elf->magic != ELF_MAGIC)
        return; // let bootasm.S handle error

    // Load each program segment (ignores ph flags).
    ph = (struct proghdr*)((uchar*)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++) {
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    }

    // Call the entry point from the ELF header.
    // Does not return!
    entry = (void(*)(void)) (elf->entry);
    entry();
}
```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

- a. The kernel file has only two program headers ✓
- b. The kernel file gets loaded at the Physical address 0x10000 in memory. ✓
- c. The elf->entry is set by the linker in the kernel file and it's 0x80000000 ✗
- d. The readseg finally invokes the disk I/O code using assembly instructions
- e. The elf->entry is set by the linker in the kernel file and it's 8010000c
- f. The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded
- g. The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it. ✓
- h. The elf->entry is set by the linker in the kernel file and it's 0x80000000 ✗
- i. The kernel file gets loaded at the Physical address 0x10000 +0x80000000 in memory.

- j. The condition if(ph->memsz > ph->filesz) is never true.
- k. The stosb() is used here, to fill in some space in memory with zeroes ✓

Your answer is partially correct.

You have correctly selected 4.

The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question 9

Correct

Mark 1.00 out of 1.00

In bootasm.S, on the line

```
1jmp    $(SEG_KCODE<<3), $start32
```

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

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

Your answer is correct.

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

Question 10

Incorrect

Mark 0.00 out of 1.00

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

`ljmp $(SEG_KCODE<<3), $start32
in bootasm.S`

0x7c00 to 0



`readseg((uchar*)elf, 4096, 0);
in bootmain.c`

Immaterial as the stack is not used here



`jmp *%eax
in entry.S`

0x7c00 to 0



`cli
in bootasm.S`

Choose...

`call bootmain
in bootasm.S`

Immaterial as the stack is not used here



Your answer is incorrect.

The correct answer is: `ljmp $(SEG_KCODE<<3), $start32`

`in bootasm.S → Immaterial as the stack is not used here, readseg((uchar*)elf, 4096, 0);`

`in bootmain.c → 0x7c00 to 0, jmp *%eax`

`in entry.S → The 4KB area in kernel image, loaded in memory, named as 'stack', cli`

`in bootasm.S → Immaterial as the stack is not used here, call bootmain`

`in bootasm.S → 0x7c00 to 0`

[◀ Random Quiz 4 : Scheduling, signals, segmentation, paging, compilation, process-state](#)

Jump to...

[Random Quiz - 6 \(xv6 file system\) ►](#)

Started on Friday, 31 March 2023, 6:18 PM

State Finished

Completed on Friday, 31 March 2023, 7:00 PM

Time taken 41 mins 52 secs

Grade 4.92 out of 15.00 (32.83%)

Question 1

Partially correct

Mark 0.67 out of 1.00

Select all the actions taken by iget()

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

Your answer is partially correct.

You have correctly selected 2.

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 **2**

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.

- a. ext4 is a log structured file system X it's a journaled file system, not log structured
- b. file system recovery recovers all the lost data X
- c. log structured file systems considerably improve the recovery time ✓
- d. xv6 has a log structured file system
- e. ext2 is by default a log structured file system X

Your answer is incorrect.

The correct answers are: xv6 has a log structured file system, log structured file systems considerably improve the recovery time

Question 3

Partially correct

Mark 0.86 out of 2.00

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

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

device files are not supported: False

The code supports IDE, and not SATA/SCSI: True

log is kept on the same device as the file system: True

disk driver handles only one buffer at a time: True

the superblock does not contain number of free blocks: True

only direct blocks are supported: False

only 2 disks are handled by default: True

Question 4

Partially correct

Mark 1.00 out of 2.00

Marks the statements as True/False w.r.t. "struct buf"

True	False	
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	B_VALID means the buffer is empty and can be reused
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	Lock on a buffer is acquired in bget, and released in brelse
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The reference count (refcnt) in struct buf is = number of processes accessing the buffer
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	A buffer can have both B_VALID and B_DIRTY flags set
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The "next" pointer chain gives the buffers in LRU order
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	A buffer can be both on the MRU/LRU list and also on ideoqueue list.
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The buffers are maintained in LRU order, in the function brelse
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	B_DIRTY flag means the buffer contains modified data

B_VALID means the buffer is empty and can be reused: False

Lock on a buffer is acquired in bget, and released in brelse: True

The reference count (refcnt) in struct buf is = number of processes accessing the buffer: True

A buffer can have both B_VALID and B_DIRTY flags set: False

The "next" pointer chain gives the buffers in LRU order: False

A buffer can be both on the MRU/LRU list and also on ideoqueue list.: True

The buffers are maintained in LRU order, in the function brelse: True

B_DIRTY flag means the buffer contains modified data: True

Question **5**

Partially correct

Mark 0.40 out of 1.00

Arrange the following in their typical order of use in xv6.

1. iget
2. ilock
3. iunlock
4. iput
5. use inode

Your answer is partially correct.

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

Grade details: 2 / 5 = 40%

Here are the scores for each item in this response:

1. 1 / 1 = 100%
2. 0 / 1 = 0%
3. 1 / 1 = 100%
4. 0 / 1 = 0%
5. 0 / 1 = 0%

The correct order for these items is as follows:

1. iget
2. ilock
3. use inode
4. iunlock
5. iput

Question 6

Partially correct

Mark 0.50 out of 1.00

Compare XV6 and EXT2 file systems.

Select True/False for each point.

True	False	
<input checked="" type="radio"/> ✗	<input type="radio"/> ✓	In both ext2 and xv6, the superblock gives location of first inode block ✗
<input type="checkbox"/> ✓	<input checked="" type="radio"/> ✗	xv6 contains journal, ext2 does not ✗
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✓	Ext2 contains superblock but xv6 does not. ✓
<input checked="" type="radio"/> ✗	<input checked="" type="radio"/> ✓	xv6 contains inode bitmap, but ext2 does not ✓
<input checked="" type="radio"/> ✗	<input type="radio"/> ✓	Both xv6 and ext2 contain magic number ✗
<input type="checkbox"/> ✓	<input checked="" type="radio"/> ✗	Ext2 contains group descriptors but xv6 does not ✓

In both ext2 and xv6, the superblock gives location of first inode block: False

xv6 contains journal, ext2 does not: True

Ext2 contains superblock but xv6 does not.: False

xv6 contains inode bitmap, but ext2 does not: False

Both xv6 and ext2 contain magic number: False

Ext2 contains group descriptors but xv6 does not: True

Question 7

Incorrect

Mark 0.00 out of 1.00

Maximum size of a file on xv6 in **bytes** is

(just write a numeric answer)

Answer: 512

✗

The correct answer is: 71680

Question 8

Partially correct

Mark 0.17 out of 1.00

Select all the actions taken by ilock()

- a. Mark the in-memory inode as valid, if needed
- b. Read the inode from disk, if needed ✓
- c. Take the sleeplock on the inode, always
- d. Get the inode from the inode-cache
- e. Copy the on-disk inode into in-memory inode, if neeed ✓
- f. Take the sleeplock on the inode, optionally ✗
- g. Lock all the buffers of the file in memory

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: Read the inode from disk, if needed, Copy the on-disk inode into in-memory inode, if neeed, Take the sleeplock on the inode, always, Mark the in-memory inode as valid, if needed

Question 9

Incorrect

Mark 0.00 out of 1.00

The lines

```
if(ip->type != T_DIR){  
    iunlockput(ip);  
    return 0;  
}
```

in namex() function

mean

- a. The last path component (which is a file, and not a directory) has been resolved, so release the lock (using iunlockput) and return
- b. One of the sub-components on the given path name, was a directory, but it was not supposed to be a directory, hence an error ✗
- c. There was a syntax error in the pathname specified
- d. One of the sub-components on the given path name, was not a directory, hence it's an error
- e. One of the sub-components on the given path name, did not exist, hence it's an error ✗
- f. ilock is held on the inode, and hence it's an error if it is a directory ✗
- g. No directory entry was found for the file to be opened, hence an error

Your answer is incorrect.

The correct answer is: One of the sub-components on the given path name, was not a directory, hence it's an error

Question 10

Partially correct

Mark 0.50 out of 1.00

Suppose an application on xv6 does the following:

```
int main() {
    char arr[128];
    int fd = open("README", O_RDONLY);
    read(fd, arr, 100);
}
```

Assume that the code works.

Which of the following things are true about xv6 kernel code, w.r.t. the above C program.

True	False	
<input checked="" type="radio"/> ✘	<input type="radio"/> ✓	The "memmove(dst, bp->data + off%BSIZE, m);" in readi() will copy the data from the disk to the kernel buffers
<input type="radio"/> ✓	<input checked="" type="radio"/> ✘	The process will be made to sleep only once
<input checked="" type="radio"/> ✘	<input type="radio"/> ✓	The data is transferred from disk to kernel buffers first, and then address of arr is mapped to the kernel buffers
<input type="radio"/> ✓	<input checked="" type="radio"/> ✘	The ONLY function that gets called on return devsw[ip->major].read(ip, dst, n); is consoleread
<input checked="" type="radio"/> ✘	<input type="radio"/> ✓	The loop in readi() will always read a different block using bread()
<input type="radio"/> ✓	<input checked="" type="radio"/> ✘	value of fd will be 3

The "memmove(dst, bp->data + off%BSIZE, m);" in readi() will copy the data from the disk to the kernel buffers: False

The process will be made to sleep only once: True

The data is transferred from disk to kernel buffers first, and then address of arr is mapped to the kernel buffers: False

The ONLY function that gets called on return devsw[ip->major].read(ip, dst, n); is consoleread: True

The loop in readi() will always read a different block using bread(): False

value of fd will be 3: True

Question 11

Partially correct

Mark 0.83 out of 1.00

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

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

Your answer is partially correct.

You have correctly selected 10.

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

Question 12

Incorrect

Mark 0.00 out of 1.00

Match function with its functionality

nameiparent	Write a new entry in a given directory	✗
dirlink	Link a directory with another directory	✗
namex	Search a given name in a given directory	✗
dirlookup	Lookup (search) for a given directory	✗

Your answer is incorrect.

The correct answer is: nameiparent → return in-memory inode for parent directory of a given pathname, dirlink → Write a new entry in a given directory, namex → return in-memory inode for a given pathname, dirlookup → Search a given name in a given directory

Question 13

Incorrect

Mark 0.00 out of 1.00

An inode is read from disk as a part of this function

- a. ilock
- b. sys_read ✗
- c. iget
- d. readi
- e. iread

Your answer is incorrect.

The correct answer is: ilock

[◀ Random Quiz - 5: xv6 make, bootloader, interrupt handling, memory management](#)

Jump to...

[\(Random Quiz - 7 \) Pre-Endsem Quiz ►](#)

Started on Thursday, 16 February 2023, 9:00 PM

State Finished

Completed on Thursday, 16 February 2023, 10:43 PM

Time taken 1 hour 43 mins

Grade 13.00 out of 15.00 (86.68%)

Question 1

Correct

Mark 1.00 out of 1.00

Which of the following statements is false ?

Select one:

- a. A process scheduling algorithm is preemptive if the CPU can be forcibly removed from a process.
- b. Time sharing systems generally use preemptive CPU scheduling.
- c. Real time systems generally use non preemptive CPU scheduling. ✓
- d. Response time is more predictable in preemptive systems than in non preemptive systems.

Your answer is correct.

The correct answer is: Real time systems generally use non preemptive CPU scheduling.

Question 2

Correct

Mark 1.00 out of 1.00

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

timer interrupt occurs

2	✓
---	---

Process P1 is running

6	✓
---	---

context of P0 is saved in P0's PCB

3	✓
---	---

Control is passed to P1

5	✓
---	---

context of P1 is loaded from P1's PCB

4	✓
---	---

Process P0 is running

1	✓
---	---

Your answer is correct.

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

Question 3

Correct

Mark 1.00 out of 1.00

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)

Paging

one continuous chunk	✓
----------------------	---

Relocation + Limit

one continuous chunk	✓
----------------------	---

Segmentation

many continuous chunks of variable size	✓
---	---

Segmentation, then paging

many continuous chunks of variable size	✓
---	---

Your answer is correct.

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

Question 4

Correct

Mark 1.00 out of 1.00

Match the names of PCB structures with kernel

- | | | |
|-------|--------------------|---|
| linux | struct task_struct | ✓ |
| xv6 | struct proc | ✓ |

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

Question 5

Correct

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

- | | | |
|----------|---------------|---|
| New: | Running | ✓ |
| Ready : | Waiting | ✓ |
| Running: | None of these | ✓ |
| Waiting: | Running | ✓ |

Question 6

Partially correct

Mark 0.47 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 7

Partially correct

Mark 0.86 out of 1.00

Mark True/False

Statements about scheduling and scheduling algorithms

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	xv6 code does not care about Processor Affinity
<input type="radio"/>	<input checked="" type="radio"/>	Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.
<input checked="" type="radio"/>	<input type="radio"/>	Response time will be quite poor on non-interruptible kernels
<input checked="" type="radio"/>	<input type="radio"/>	Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.
<input checked="" type="radio"/>	<input type="radio"/>	Processor Affinity refers to memory accesses of a process being stored on cache of that processor
<input checked="" type="radio"/>	<input type="radio"/>	A scheduling algorithm is non-preemptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.
<input type="radio"/>	<input checked="" type="radio"/>	On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread It's the negation of this. Time NOT spent in idle thread.

xv6 code does not care about Processor Affinity: True

Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.: True

Response time will be quite poor on non-interruptible kernels: True

Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.: True

Processor Affinity refers to memory accesses of a process being stored on cache of that processor: True

A scheduling algorithm is non-preemptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.: True

On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread: False

Question 8

Correct

Mark 1.00 out of 1.00

Mark whether the concept is related to scheduling or not.

Yes	No	
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	file-table
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	context-switch
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	timer interrupt
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	ready-queue
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	runnable process

file-table: No

context-switch: Yes

timer interrupt: Yes

ready-queue: Yes

runnable process: Yes

Question 9

Partially correct

Mark 0.50 out of 1.00

Select all the correct statements about signals

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.

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

Question 10

Correct

Mark 1.00 out of 1.00

Map each signal with its meaning

SIGPIPE	Broken Pipe	✓
SIGCHLD	Child Stopped or Terminated	✓
SIGUSR1	User Defined Signal	✓
SIGALRM	Timer Signal from alarm()	✓
SIGSEGV	Invalid Memory Reference	✓

The correct answer is: SIGPIPE → Broken Pipe, SIGCHLD → Child Stopped or Terminated, SIGUSR1 → User Defined Signal, SIGALRM → Timer Signal from alarm(), SIGSEGV → Invalid Memory Reference

Question 11

Partially correct

Mark 0.80 out of 1.00

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

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

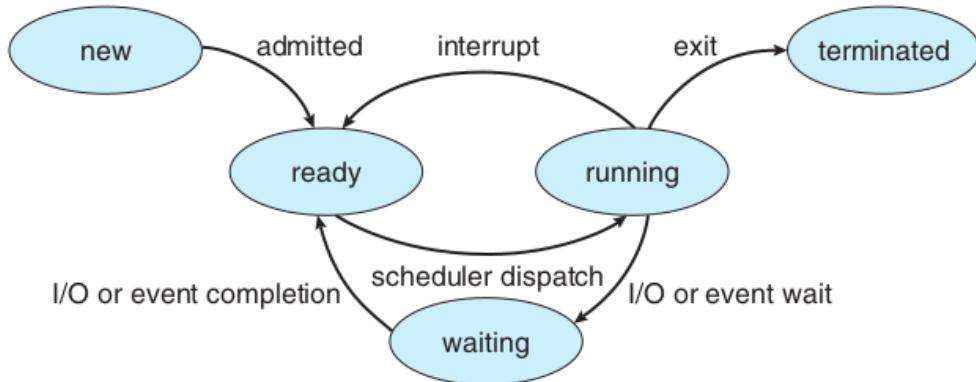


Figure 3.2 Diagram of process state.

True	False	
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Every forked process has to go through ZOMBIE state, at least for a small duration.
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Only a process in READY state is considered by scheduler
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/>	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/>	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.

Every forked process has to go through ZOMBIE state, at least for a small duration.: True

Only a process in READY state is considered by scheduler: True

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: 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

Question **12**

Correct

Mark 1.00 out of 1.00

Which of the following parts of a C program do not have any corresponding machine code ?

- a. pointer dereference
- b. function calls
- c. global variables ✓
- d. local variable declaration
- e. #directives ✓
- f. typedefs ✓
- g. expressions

Your answer is correct.

The correct answers are: #directives, typedefs, global variables

Question 13

Partially correct

Mark 1.38 out of 2.00

Select all the correct statements about the state of a process.

- a. A process in ready state is ready to be scheduled ✓
- b. A waiting process starts running after the wait is over
- c. A process changes from running to ready state on a timer interrupt or any I/O wait
- d. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✓
- e. Processes in the ready queue are in the ready state ✓
- f. A process changes from running to ready state on a timer interrupt
- g. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers
- h. A process waiting for any condition is woken up by another process only ✗
- i. A running process may terminate, or go to wait or become ready again ✓
- j. A process can self-terminate only when it's running ✓
- k. Typically, it's represented as a number in the PCB ✓
- l. A process that is running is not on the ready queue ✓
- m. Changing from running state to waiting state results in "giving up the CPU" ✓
- n. A process in ready state is ready to receive interrupts

Your answer is partially correct.

You have correctly selected 8.

The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question **14**

Correct

Mark 1.00 out of 1.00

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

- a. Invoke the linker to link the function calls with their code, extern globals with their declaration
- b. Check the program for logical errors ✓
- c. Convert high level language code to machine code
- d. Check the program for syntactical errors
- e. Suggest alternative pieces of code that can be written ✓
- f. Process the # directives in a C program

Your answer is correct.

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

◀ Random Quiz - 3 (processes, memory management, event driven kernel), compilation-linking-loading, ipc-pipes

Jump to...

Random Quiz - 5: xv6 make, bootloader, interrupt handling, memory management ►

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

Started on Monday, 24 January 2022, 7:02:26 PM

State Finished

Completed on Monday, 24 January 2022, 8:57:38 PM

Time taken 1 hour 55 mins

Grade **10.40** out of 20.00 (52%)

Question 1

Complete

Mark 1.00 out of 1.00

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

What's the value printed here at LINE A?

Answer:

The correct answer is: 5

Question 2

Complete

Mark 1.00 out of 1.00

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

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

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

Question 3

Complete

Mark 2.00 out of 2.00

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

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

hi

hi hi

hi

hi hi

`main() { fork(); execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }`

`main() { execl("/usr/bin/echo", "/usr/bin/echo", "hi\n", NULL); }`

`main() { int i = NULL; fork(); printf("hi\n"); }`

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

Question 4

Complete

Mark 1.33 out of 2.00

Which of the following instructions should be privileged?

Select one or more:

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

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

Question 5

Complete

Mark 0.00 out of 0.50

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

- a. yes
- b. no

The correct answer is: yes

Question 6

Complete

Mark 1.00 out of 1.00

Match the register with the segment used with it.

esp	ss	▼
ebp	ss	▼
eip	cs	▼
esi	ds	▼
edi	es	▼

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

Question 7

Complete

Mark 0.00 out of 2.00

```
xv6.img: bootblock kernel
```

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

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

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

The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

Question 8

Complete

Mark 0.50 out of 0.50

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

- a. yes
- b. no

The correct answer is: no

Question 9

Complete

Mark 1.00 out of 2.00

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

- a. Process the # directives in a C program
- b. Convert high level language code to machine code
- c. Invoke the linker to link the function calls with their code, extern globals with their declaration
- d. Suggest alternative pieces of code that can be written
- e. Check the program for syntactical errors
- f. Check the program for logical errors

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

Question 10

Complete

Mark 0.00 out of 0.50

Compare multiprogramming with multitasking

- a. A multiprogramming system is not necessarily multitasking
- b. A multitasking system is not necessarily multiprogramming

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

Question 11

Complete

Mark 0.40 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

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

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

Question 12

Complete

Mark 1.00 out of 1.00

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

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

Magnetic tapes

Optical disk

Hard-disk drives

Nonvolatile memory

Main memory

Cache

Registers

Question 13

Complete

Mark 0.83 out of 2.00

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

Select one or more:

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

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

Question 14

Complete

Mark 0.33 out of 0.50

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

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

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

Question 15

Complete

Mark 0.00 out of 2.00

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

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

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

Question 16

Complete

Mark 0.00 out of 1.00

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

Select one:

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

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

[◀ \(Task\) Compulsory xv6 task](#)

Jump to...



[\(Optional Assignment\) Shell Programming\(Conformance tests\) ▶](#)

Started on Wednesday, 9 February 2022, 7:01:59 PM

State Finished

Completed on Wednesday, 9 February 2022, 8:01:50 PM

Time taken 59 mins 51 secs

Grade 10.00 out of 11.00 (91%)

Question 1

Complete

Mark 1.00 out of 1.00

The ljmp instruction in general does

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

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

Question 2

Complete

Mark 1.00 out of 1.00

The kernel is loaded at Physical Address

- a. 0x0010000
- b. 0x80000000
- c. 0x00100000
- d. 0x80100000

The correct answer is: 0x00100000

Question 3

Complete

Mark 1.00 out of 1.00

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

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

The correct answer is: 3

Question 4

Complete

Mark 1.00 out of 1.00

The variable \$stack in entry.S is

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

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

Question 5

Complete

Mark 1.00 out of 1.00

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

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

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

Question 6

Complete

Mark 1.00 out of 1.00

The kernel ELF file contains how many Program headers?

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

The correct answer is: 3

Question 7

Complete

Mark 1.00 out of 1.00

ELF Magic number is

- a. 0xELF
- b. 0xFFFFFFFF
- c. 0
- d. 0x0x464CELF
- e. 0xELFELFELF
- f. 0x464C457FL
- g. 0x464C457FU

The correct answer is: 0x464C457FU

Question 8

Complete

Mark 1.00 out of 1.00

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

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

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

Question 9

Complete

Mark 1.00 out of 1.00

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

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

The correct answer is: Because it needs to setup paging

Question 10

Not answered

Marked out of 0.50

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

Answer:

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

Question 11

Not answered

Marked out of 0.50

Match the pairs of which action is taken by whom

Answer:

The correct answer is: kernel

Question 12

Complete

Mark 1.00 out of 1.00

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

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

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

[◀ Homework questions: Basics of MM, xv6 booting](#)

Jump to...

[\(Code\)](#) [Files](#), [redirection](#), [dup](#), [\(IPC\)pipe](#) ►

Started on Saturday, 26 February 2022, 5:20:40 PM

State Finished

Completed on Saturday, 26 February 2022, 6:35:00 PM

Time taken 1 hour 14 mins

Grade 9.05 out of 15.00 (60%)

Question 1

Complete

Mark 0.75 out of 1.00

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

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

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 2

Complete

Mark 0.67 out of 1.00

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

Select one or more:

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

The correct answers are: paging, segmentation, demand paging

Question 3

Complete

Mark 0.29 out of 1.00

Compare paging with demand paging and select the correct statements.

Select one or more:

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

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 4

Complete

Mark 0.00 out of 1.00

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

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

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 5

Complete

Mark 0.14 out of 1.00

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

Page table of P1, Page 5	23	▼
Page table of P2, Page 1	5	▼
Page table of P2, Page 0	6	▼
Page table of P1, Page 3	9	▼
Page table of P1, Page 4	9	▼
Page table of P2, Page 3	7	▼
Page table of P2, Page 4	9	▼

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

Question 6

Complete

Mark 1.00 out of 1.00

Page sizes are a power of 2 because

Select one:

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

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

Question 7

Complete

Mark 0.60 out of 1.00

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

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

```

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

```

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

```

7f275ba1e000-7f275ba46000 r--p 00066000 103:05 924277	/usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0	
7f275ba46000-7f275ba47000 r--p 0008d000 103:05 924277	/usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0	
7f275ba47000-7f275ba48000 rw-p 0008e000 103:05 924277	/usr/lib/x86_64-linux-
gnu/libpcre2-8.so.0.9.0	
7f275ba48000-7f275ba4d000 r--p 00000000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275ba4d000-7f275bbe5000 r-xp 00025000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275bbe5000-7f275bc2f000 r--p 0019d000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275bc2f000-7f275bc30000 ---p 001e7000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275bc30000-7f275bc33000 r--p 001e7000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275bc33000-7f275bc36000 rw-p 001ea000 103:05 917893	/usr/lib/x86_64-linux-
gnu/libc-2.31.so	
7f275bc36000-7f275bc3a000 rw-p 00000000 00:00 0	
7f275bc3a000-7f275bc41000 r--p 00000000 103:05 917906	/usr/lib/x86_64-linux-
gnu/libpthread-2.31.so	
7f275bc41000-7f275bc52000 r-xp 00007000 103:05 917906	/usr/lib/x86_64-linux-
gnu/libpthread-2.31.so	
7f275bc52000-7f275bc57000 r--p 00018000 103:05 917906	/usr/lib/x86_64-linux-
gnu/libpthread-2.31.so	
7f275bc57000-7f275bc58000 r--p 0001c000 103:05 917906	/usr/lib/x86_64-linux-
gnu/libpthread-2.31.so	
7f275bc58000-7f275bc59000 rw-p 0001d000 103:05 917906	/usr/lib/x86_64-linux-
gnu/libpthread-2.31.so	
7f275bc59000-7f275bc5d000 rw-p 00000000 00:00 0	
7f275bc5d000-7f275bcce000 r--p 00000000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275bcce000-7f275bf29000 r-xp 00071000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275bf29000-7f275c142000 r--p 002cc000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275c142000-7f275c143000 ---p 004e5000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275c143000-7f275c149000 r--p 004e5000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275c149000-7f275c190000 rw-p 004eb000 103:05 917016	/usr/lib/x86_64-linux-
gnu/libpython3.8.so.1.0	
7f275c190000-7f275c1b3000 rw-p 00000000 00:00 0	
7f275c1b3000-7f275c1b4000 r--p 00000000 103:05 917894	/usr/lib/x86_64-linux-
gnu/libdl-2.31.so	
7f275c1b4000-7f275c1b6000 r-xp 00001000 103:05 917894	/usr/lib/x86_64-linux-
gnu/libdl-2.31.so	
7f275c1b6000-7f275c1b7000 r--p 00003000 103:05 917894	/usr/lib/x86_64-linux-
gnu/libdl-2.31.so	
7f275c1b7000-7f275c1b8000 r--p 00003000 103:05 917894	/usr/lib/x86_64-linux-
gnu/libdl-2.31.so	
7f275c1b8000-7f275c1b9000 rw-p 00004000 103:05 917894	/usr/lib/x86_64-linux-
gnu/libdl-2.31.so	
7f275c1b9000-7f275c1bb000 rw-p 00000000 00:00 0	
7f275c1bb000-7f275c1c0000 r-xp 00000000 103:05 923815	/usr/lib/x86_64-linux-
gnu/libgpm.so.2	
7f275c1c0000-7f275c3bf000 ---p 00005000 103:05 923815	/usr/lib/x86_64-linux-
gnu/libgpm.so.2	
7f275c3bf000-7f275c3c0000 r--p 00004000 103:05 923815	/usr/lib/x86_64-linux-
gnu/libgpm.so.2	
7f275c3c0000-7f275c3c1000 rw-p 00005000 103:05 923815	/usr/lib/x86_64-linux-
gnu/libgpm.so.2	

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

```
7f275c5dc000-7f275c5dd000 rw-p 0002d000 103:05 917889          /usr/lib/x86_64-linux-gnu/ld-
2.31.so
7f275c5dd000-7f275c5de000 rw-p 00000000 00:00 0
7ffd22d2f000-7ffd22d50000 rw-p 00000000 00:00 0          [stack]
7ffd22db0000-7ffd22db4000 r--p 00000000 00:00 0          [vvar]
7ffd22db4000-7ffd22db6000 r-xp 00000000 00:00 0          [vdso]
fffffffff600000-ffffffffffff601000 --xp 00000000 00:00 0          [vsySCALL]
```

True False

This is a virtual memory map (not physical memory map)

The 5th entry 55a4352c5000-55a4352e2000 **may** correspond to "data" of the vim.basic

vim.basic uses the math library

The size of the stack is one page

The size of the heap is one page

This is a virtual memory map (not physical memory map): True

The 5th entry 55a4352c5000-55a4352e2000 **may** correspond to "data" of the vim.basic: True

vim.basic uses the math library: True

The size of the stack is one page: False

The size of the heap is one page: False

Question 8

Complete

Mark 0.50 out of 0.50

Map the technique with its feature/problem

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

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

Question 9

Complete

Mark 0.36 out of 0.50

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

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

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

Question 10

Complete

Mark 0.75 out of 1.00

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

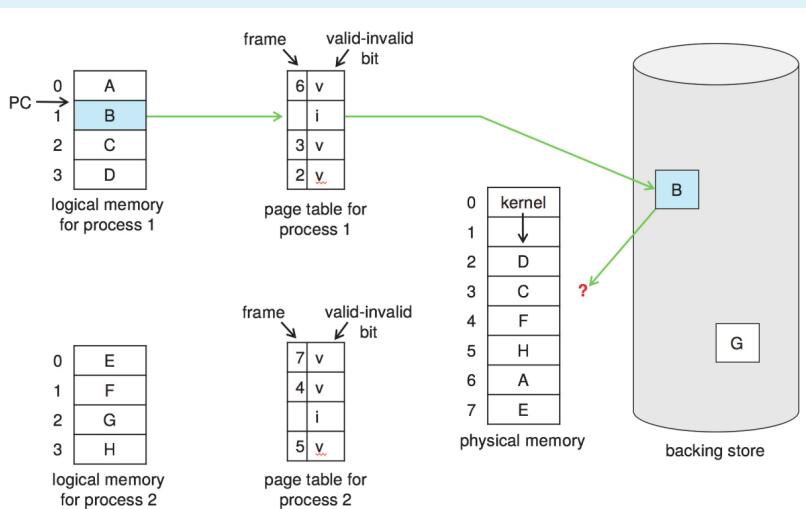


Figure 10.9 Need for page replacement.

True False

- | | | |
|----------------------------------|----------------------------------|--|
| <input type="radio"/> | <input checked="" type="radio"/> | Global replacement means chose any of the frame from 2 to 7 |
| <input checked="" type="radio"/> | <input type="radio"/> | Page 1 of process 1 needs a replacement |
| <input checked="" type="radio"/> | <input type="radio"/> | Handling this scenario demands two disk I/Os |
| <input checked="" type="radio"/> | <input type="radio"/> | Local replacement means chose any of the frames 2, 3, 6 |
| <input type="radio"/> | <input checked="" type="radio"/> | The kernel's pages can not used for replacement if kernel is not pageable. |
| <input checked="" type="radio"/> | <input type="radio"/> | Global replacement means chose any of the frame from 0 to 7 |
| <input checked="" type="radio"/> | <input type="radio"/> | Kernel occupies two page frames |
| <input type="radio"/> | <input checked="" type="radio"/> | Local replacement means chose any of the frame from 2 to 7 |

Global replacement means chose any of the frame from 2 to 7: True

Page 1 of process 1 needs a replacement: True

Handling this scenario demands two disk I/Os: True

Local replacement means chose any of the frames 2, 3, 6: True

The kernel's pages can not used for replacement if kernel is not pageable.: True

Global replacement means chose any of the frame from 0 to 7: False

Kernel occupies two page frames: True

Local replacement means chose any of the frame from 2 to 7: False

Question 11

Complete

Mark 0.75 out of 1.00

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

```
626729 482768 /usr/lib/firefox/firefox -contentproc -parentBuildID 20220202182137 -prefsLen 9256 -prefMapSize 264738 -appDir /usr/lib/firefox/browser 6094 true rdd
2167 687 /usr/sbin/apache2 -k start
1265185 222 /usr/bin/gnome-shell
102648 111 /usr/sbin/mysqld
9813 0 bash
15497 370 /usr/bin/gedit --gapplication-service
```

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

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

Question 12

Complete

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 = 105 ns

Average page fault service time = 13 ms

Page fault rate = 0.6

Answer:

The correct answer is: 7800042.00

Question 13

Complete

Mark 0.50 out of 1.00

For the reference string

3 4 3 5 2

the number of page faults (including initial ones) using

FIFO replacement and 2 page frames is :

4

FIFO replacement and 3 page frames is :

3

Question 14

Complete

Mark 1.00 out of 1.00

Assuming a 8- KB page size, what is the page numbers for the address 874815 reference in decimal :

(give answer also in decimal)

Answer: 107

The correct answer is: 107

Question 15

Complete

Mark 0.75 out of 1.00

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

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

The correct order for these items is as follows:

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

Question 16

Complete

Mark 1.00 out of 1.00

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

- worst fit 500 KB 635 KB ▲
- best fit 500 KB 600 KB ▲
- worst fit 115 KB 750 KB ▲
- best fit 115 KB 125 KB ▲
- first fit 115 KB 300 KB ▲
- first fit 500 KB 600 KB ▲

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

[◀ \(Code\) mmap related programs](#)

Jump to...

[Points from Mid-term feedback ►](#)

Started on Monday, 7 March 2022, 7:00:46 PM

State Finished

Completed on Monday, 7 March 2022, 8:20:00 PM

Time taken 1 hour 19 mins

Grade 11.42 out of 15.00 (76%)

Question 1

Complete

Mark 1.00 out of 1.00

Map the virtual address to physical address in xv6

KERNBASE	0	▼
KERNLINK	0x100000	▼
80108000	0x108000	▼
0xFE000000	0xFE000000	▼

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

Question 2

Complete

Mark 1.00 out of 1.00

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

- a. no
- b. yes

The correct answer is: yes

Question 3

Complete

Mark 1.00 out of 1.00

What does seginit() do?

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

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 4

Complete

Mark 1.50 out of 1.50

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

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

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

Question 5

Complete

Mark 1.00 out of 1.00

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

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

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

Question 6

Complete

Mark 0.67 out of 1.00

Which of the following is done by mappages()?

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

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

Question 7

Complete

Mark 0.08 out of 1.00

Select all the correct statements about initcode

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

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

Question 8

Complete

Mark 1.00 out of 1.00

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

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

The correct answer is: 801154a8

Question 9

Complete

Mark 1.00 out of 1.00

What does userinit() do ?

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

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

Question 10

Complete

Mark 1.50 out of 1.50

Which of the following is DONE by allocproc() ?

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

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

Question 11

Complete

Mark 0.67 out of 2.00

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

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

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 12

Complete

Mark 0.00 out of 1.00

The approximate number of page frames created by kinit1 is

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

The correct answer is: 3000

Question 13

Complete

Mark 1.00 out of 1.00

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

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

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

[◀ Questions for test on kalloc/kfree/kvmalloc, etc.](#)

Jump to...



(Optional Assignment) Slab allocator in xv6 ►

Started on Wednesday, 19 April 2023, 6:30 PM

State Finished

Completed on Wednesday, 19 April 2023, 8:52 PM

Time taken 2 hours 21 mins

Overdue 21 mins 46 secs

Grade 23.26 out of 30.00 (77.54%)

Question 1

Partially correct

Mark 0.88 out of 1.00

Select all correct statements about file system recovery (without journaling) programs e.g. fsck

Select one or more:

- a. A recovery program, most typically, builds the file system data structure and checks for inconsistencies
- b. Recovery programs are needed only if the file system has a delayed-write policy. ✓
- c. They may take very long time to execute ✓
- d. It is possible to lose data as part of recovery ✓
- e. Even with a write-through policy, it is possible to need a recovery program. ✓
- f. They can make changes to the on-disk file system ✓
- g. Recovery is possible due to redundancy in file system data structures ✓
- h. They are used to recover deleted files
- i. Recovery programs recalculate most of the metadata summaries (e.g. free inode count) ✓

Your answer is partially correct.

You have correctly selected 7.

The correct answers are: Recovery is possible due to redundancy in file system data structures, A recovery program, most typically, builds the file system data structure and checks for inconsistencies, It is possible to lose data as part of recovery, They may take very long time to execute, They can make changes to the on-disk file system, Recovery programs recalculate most of the metadata summaries (e.g. free inode count), Recovery programs are needed only if the file system has a delayed-write policy., Even with a write-through policy, it is possible to need a recovery program.

Question 2

Partially correct

Mark 1.75 out of 2.00

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

"..." means some code.

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

Tell compiler not to reorder memory access beyond this line



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

Release the lock held by some another process



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

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



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

Disable interrupts to avoid deadlocks



```
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);  
}
```

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



```

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;
}

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

```

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



Disable interrupts to avoid another process's pointer being returned



Your answer is partially correct.

You have correctly selected 7.

The correct answer is: `void
acquire(struct spinlock *lk)
{
...
__sync_synchronize();`

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

```

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

```

→ Release the lock held by some another process, `void`

```

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

```

→ Traverse ebp chain to get sequence of instructions followed in functions calls, `void`

```

acquire(struct spinlock *lk)
{
    pushcli();
    → Disable interrupts to avoid deadlocks, 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, 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), struct proc*
myproc(void) {
    ...
    pushcli();
    c = mycpu();
    p = c->proc;
    popcli();
    ...
}

```

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

Question 3

Partially correct

Mark 0.80 out of 1.00

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

Select one or more:

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 4

Correct

Mark 2.00 out of 2.00

Select all the correct statements about synchronization primitives.

Select one or more:

- a. Blocking means one process passing over control to another process
- b. Semaphores are always a good substitute for spinlocks
- c. Spinlocks consume CPU time ✓
- d. Mutexes can be implemented using blocking and wakeup ✓
- e. All synchronization primitives are implemented essentially with some hardware assistance. ✓
- f. Mutexes can be implemented without any hardware assistance
- g. Spinlocks are good for multiprocessor scenarios, for small critical sections ✓
- h. Semaphores can be used for synchronization scenarios like ordered execution ✓
- i. Mutexes can be implemented using spinlock ✓
- j. Blocking means moving the process to a wait queue and calling scheduler ✓
- k. Thread that is going to block should not be holding any spinlock ✓
- l. Blocking means moving the process to a wait queue and spinning

Your answer is correct.

The correct answers are: Spinlocks are good for multiprocessor scenarios, for small critical sections, Spinlocks consume CPU time, Semaphores can be used for synchronization scenarios like ordered execution, Mutexes can be implemented using spinlock, Mutexes can be implemented using blocking and wakeup, Thread that is going to block should not be holding any spinlock, Blocking means moving the process to a wait queue and calling scheduler, All synchronization primitives are implemented essentially with some hardware assistance.

Question 5

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
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

✓

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

Ensure that no printing happens on other processors

✓

Your answer is correct.

The correct answer is:

```
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, void
panic(char *s)
{
...
panicked = 1; → Ensure that no printing happens on other processors
```

Question 6

Correct

Mark 1.00 out of 1.00

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

Consider each request as first request.

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

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

Question 7

Correct

Mark 1.00 out of 1.00

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

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

The functions like argint(), argstr() make the system call arguments available in the kernel.: True

The arguments are accessed in the kernel code using esp on the trapframe.: True

String arguments are first copied to trapframe and then from trapframe to kernel's other variables.: False

Integer arguments are copied from user memory to kernel memory using argint(): True

Integer arguments are stored in eax, ebx, ecx, etc. registers: False

The arguments to system call originally reside on process stack.: True

String arguments are NOT copied in kernel memory, but just pointed to by a kernel memory pointer: True

The arguments to system call are copied to kernel stack in trapasm.S: False

Question 8

Correct

Mark 1.00 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.

- a. file system recovery recovers all the lost data
- b. file system recovery may end up losing data✓
- c. log may be kept on same block device or another block device✓
- d. even if file systems followed immediate writes (i.e. non-delayed writes), it could still require recovery✓
- 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 9

Complete

Mark 1.50 out of 3.00

List down all changes required to xv6 code, in order to add the system call chown().

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.

This implementation assumes there is multiple user support for xv6

- a) void chown(char* pathname, int owner, int group){
 if(priviledge(currentOwner) > privileged(owner)){
 //change owner in inode of file/folder pointed by pathname
 }
}
- b) sys_calls to check currentOwner() and filePriviledges(char* filepath)
- c) all file related sys_calls like open and read should check for currently logged in user and priviledges for the file
- d)
- e)
struct inode {
 uint dev; // Device number
 uint inum; // Inode number
 int ref; // Reference count
 struct sleeplock lock; // protects everything below here
 int valid; // inode has been read from disk?
 short type; // copy of disk inode
 short major;
 short minor;
 short nlink;
 uint size;
 uint addrs[NDIRECT+1];
 int owner; -----> owner
 int group; -----> group
};
- f) no changes in makefile for adding system a sys_call
- g)

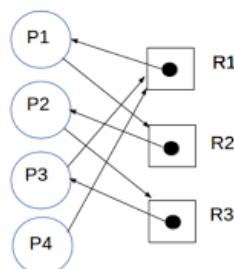
Comment:

Question 10

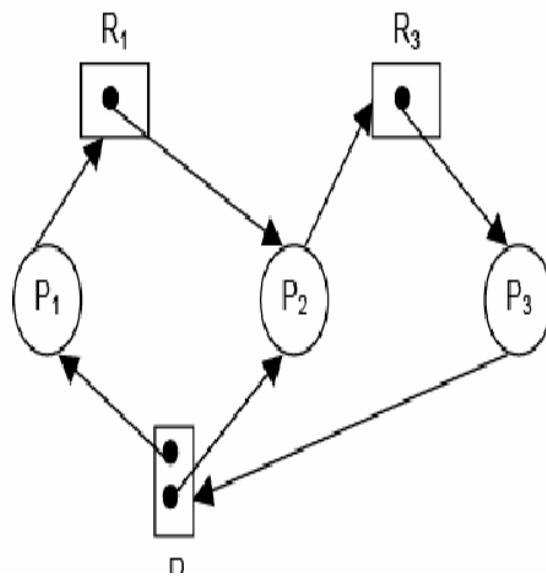
Correct

Mark 1.00 out of 1.00

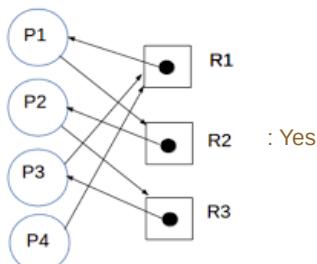
For each of the resource allocation diagram shown,
infer whether the graph contains at least one deadlock or not.

Yes**No** X

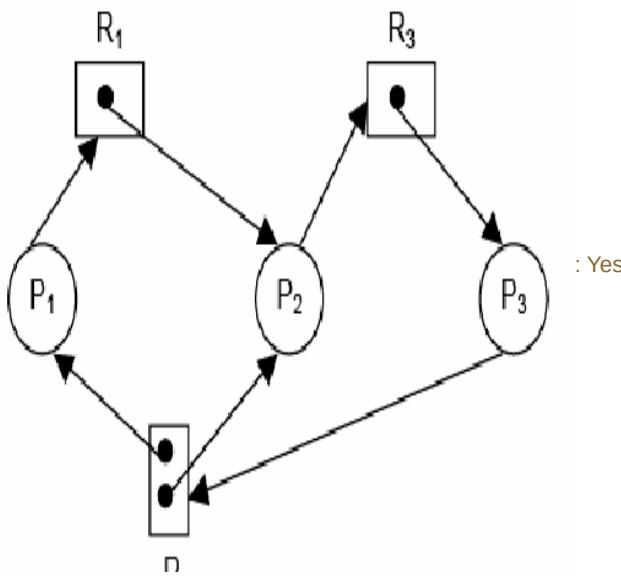
✓

 X

✓



: Yes



: Yes

Question 11

Incorrect

Mark 0.00 out of 1.00

Given that the memory access time is 150 ns, probability of a page fault is 0.8 and page fault handling time is 6 ms,
The effective memory access time in nanoseconds is:

Answer: ×

The correct answer is: 4800030.00

Question 12

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:



The correct answer is: 11011110

Question 13

Partially correct

Mark 0.86 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	
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager stores additional metadata on the physical disk partitions
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume can be extended in size but upto the size of volume group
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume may span across multiple physical partitions
<input checked="" type="radio"/>	<input type="radio"/> X	A physical partition should be initialized as a physical volume, before it can be used by volume manager.
<input checked="" type="radio"/>	<input type="radio"/> X	A logical volume may span across multiple physical volumes
<input checked="" type="radio"/>	<input type="radio"/> X	The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.
<input checked="" type="radio"/>	<input type="radio"/> X	A volume group consists of multiple physical volumes

The volume manager stores additional metadata on the physical disk partitions: 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 partitions: True

A physical partition should be initialized as a physical volume, before it can be used by volume manager.: True

A logical volume may span across multiple physical volumes: True

The volume manager can create further internal sub-divisions of a physical partition for efficiency or features.: True

A volume group consists of multiple physical volumes: True

Question 14

Incorrect

Mark 0.00 out of 1.00

Assuming a 8- KB page size, what is the page numbers for the address 26583 reference in decimal :

(give answer also in decimal)

Answer: 13

X

The correct answer is: 3

Question 15

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. 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() ✓
- b. 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. ✓
- 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 file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup. ✓
- e. 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. ✓
- f. A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/" ✓
- g. The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories ✓
- h. The operating system in-memory inode needs to be a generic-inode representing "inode" like data structure across multiple file systems. ✓

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 16

Partially correct

Mark 1.43 out of 2.00

Compare paging with demand paging and select the correct statements.

Select one or more:

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

Your answer is partially correct.

You have correctly selected 5.

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 17

Partially correct

Mark 0.80 out of 1.00

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

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

Processes keep changing their locality of reference, and least number of page faults occur when they are changing the locality.: False
Thrashing occurs when the total size of all process's locality exceeds total memory size.: True

Thrashing can be limited if local replacement is used.: True

During thrashing the CPU is under-utilised as most time is spent in I/O: True

Thrashing can occur even if entire memory is not in use.: False

The working set model is an attempt at approximating the locality of a process.: True

Thrashing is particular to demand paging systems, and does not apply to pure paging systems.: True

Thrashing occurs because some process is doing lot of disk I/O.: False

mmap() solves the problem of thrashing.: False

Processes keep changing their locality of reference, and a high rate of page faults occur when they are changing the locality.: True

Question 18

Correct

Mark 1.00 out of 1.00

Match the code with its functionality

S = 5

Wait(S)

Critical Section

Counting semaphore



Signal(S)

S1 = 0; S2 = 0;

P2:

Statement1;

Signal(S2);

P1:

Wait(S2);

Statemetn2;

Signal(S1);

Execution order P2, P1, P3



P3:

Wait(S1);

Statement S3;

S = 0

P1:

Statement1;

Signal(S)

Execution order P1, then P2



P2:

Wait(S)

Statment2;

S = 1

Wait(S)

Critical Section

Binary Semaphore for mutual exclusion



Signal(S);

Your answer is correct.

The correct answer is: S = 5

Wait(S)

Critical Section

Signal(S) → Counting semaphore, S1 = 0; S2 = 0;

P2:

Statement1;

Signal(S2);

P1:

Wait(S2);

Statemetn2;

Signal(S1);

P3:

Wait(S1);

Statement S3; → Execution order P2, P1, P3, S = 0

P1:

Statement1;

Signal(S)

P2:

Wait(S)

Statement2; → Execution order P1, then P2, **S = 1**

Wait(S)

Critical Section

Signal(S); → Binary Semaphore for mutual exclusion

Question **19**

Correct

Mark 1.00 out of 1.00

Map the technique with its feature/problem

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

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

Question **20**

Partially correct

Mark 0.25 out of 1.00

Select all correct statements about journaling (logging) in file systems like ext3

Select one or more:

- a. A different device driver is always needed to access the journal
- b. Journals are often stored circularly
- c. Most typically a transaction in journal is recorded atomically (full or none) ✓
- d. Journals must be maintained on the same device that hosts the file system
- e. The purpose of journal is to speed up file system recovery
- f. Journal is hosted in the same device that hosts the swap space
- g. the journal contains a summary of all changes made as part of a single transaction

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: The purpose of journal is to speed up file system recovery, the journal contains a summary of all changes made as part of a single transaction, Most typically a transaction in journal is recorded atomically (full or none), Journals are often stored circularly

Question 21

Complete

Mark 1.00 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.

a) buddyAllocate(int requiredSize, string allocatedBitmap, int i, int j){
 string sizeRequired = find the least bigger power of 2 than requiredSize in a form of bitmap
 if(sizeRequired == allocatedBitmap[i:j]) {
 return location in bitmap which match with substring allocatedBitmap[i:j];
 buddyAllocate(requiredSize, 0, sizeof(allocatedBitmap)/2));
 buddyAllocate(requiredSize, sizeof(allocatedBitmap)/2+1, sizeof(allocatedBitmap));

b)
c) filealloc() // remove lines ---->

// for(f = ftable.file; f < ftable.file + NFILE; f++){
// if(f->ref == 0){
// f->ref = 1;
// release(&ftable.lock);
// return f;
// }
// }
and add -----> f = getCache();

d) struct memoryBitmap{
 spinlock sl;
 uint allocatedBitmap;
 int sizeMappedToEachBitmap; // like 32 bytes in one of previous questions

- e)
- f) no changes
- g) nope

Comment:

checked

Question 22

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:



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 23

Correct

Mark 1.00 out of 1.00

Match each suggested semaphore implementation (discussed in class)

with the problems that it faces

```
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);  
    schedule();  
}  
wait(semaphore *s) {  
    spinlock(&(s->sl));  
    while(s->val <=0) {  
        block(s);  
    }  
    (s->val)--;  
    spinunlock(&(s->sl));  
}
```

blocks holding a spinlock



```
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) {  
        spinunlock(&(s->sl));  
        spinlock(&(s->sl));  
    }  
    (s->val)--;  
    spinunlock(&(s->sl));  
}
```

too much spinning, bounded wait not guaranteed



```
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));  
}
```

deadlock



```

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(semaphore *s) {
    spinlock(*(s->sl));
    (s->val)++;
    x = dequeue(s->sl) and enqueue(readyq, x);
    spinunlock(*(s->sl));
}

```

not holding lock after unblock ✓

Your answer is correct.

The correct answer is:

```

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);
    schedule();
}

wait(semaphore *s) {
    spinlock(&(s->sl));
    while(s->val <=0) {
        block(s);
    }
    (s->val)--;
    spinunlock(&(s->sl));
}

```

→ blocks holding a spinlock,

```

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) {
        spinunlock(&(s->sl));
        spinlock(&(s->sl));
    }
    (s->val)--;
    spinunlock(&(s->sl));
}

```

→ too much spinning, bounded wait not guaranteed,

```

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));
}

```

→ deadlock,

```

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(semaphore *s) {
    spinlock(*(&s->sl));
    (s->val)++;
    x = dequeue(s->sl) and enqueue(readyq, x);
    spinunlock(*(&s->sl));
}

```

→ not holding lock after unblock

Started on Friday, 31 March 2023, 6:18 PM

State Finished

Completed on Friday, 31 March 2023, 7:00 PM

Time taken 41 mins 48 secs

Grade 7.73 out of 15.00 (51.54%)

Question 1

Partially correct

Mark 0.75 out of 1.00

Select all the actions taken by iget()

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

Your answer is partially correct.

You have selected too many options.

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 2

Partially correct

Mark 0.60 out of 1.00

Arrange the following in their typical order of use in xv6.

1. use inode
2. iget
3. ilock
4. iunlock
5. iput

Your answer is partially correct.

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

Grade details: 3 / 5 = 60%

Here are the scores for each item in this response:

1. 0 / 1 = 0%
2. 1 / 1 = 100%
3. 0 / 1 = 0%
4. 1 / 1 = 100%
5. 1 / 1 = 100%

The correct order for these items is as follows:

1. iget
2. ilock
3. use inode
4. iunlock
5. iput

Question 3

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.

- a. file system recovery recovers all the lost data
- b. xv6 has a log structured file system✓
- c. ext4 is a log structured file system ✗ it's a journaled file system, not log structured
- d. ext2 is by default a log structured file system
- e. log structured file systems considerably improve the recovery time✓

Your answer is incorrect.

The correct answers are: xv6 has a log structured file system, log structured file systems considerably improve the recovery time

Question 4

Correct

Mark 1.00 out of 1.00

Select all the actions taken by ilock()

- a. Get the inode from the inode-cache
- b. Take the sleeplock on the inode, always✓
- c. Lock all the buffers of the file in memory
- d. Take the sleeplock on the inode, optionally
- e. Copy the on-disk inode into in-memory inode, if needed✓
- f. Mark the in-memory inode as valid, if needed✓
- g. Read the inode from disk, if needed✓

Your answer is correct.

The correct answers are: Read the inode from disk, if needed, Copy the on-disk inode into in-memory inode, if needed, Take the sleeplock on the inode, always, Mark the in-memory inode as valid, if needed

Question 5

Incorrect

Mark 0.00 out of 1.00

Maximum size of a file on xv6 in **bytes** is

(just write a numeric answer)

Answer: 16920576



The correct answer is: 71680

Question 6

Partially correct

Mark 1.71 out of 2.00

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

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

The code supports IDE, and not SATA/SCSI: True

only direct blocks are supported: False

the superblock does not contain number of free blocks: True

only 2 disks are handled by default: True

disk driver handles only one buffer at a time: True

log is kept on the same device as the file system: True

device files are not supported: False

Question 7

Partially correct

Mark 0.50 out of 1.00

Compare XV6 and EXT2 file systems.

Select True/False for each point.

True	False	
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/> <input type="checkbox"/>	xv6 contains journal, ext2 does not
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Ext2 contains superblock but xv6 does not.
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	In both ext2 and xv6, the superblock gives location of first inode block
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	xv6 contains inode bitmap, but ext2 does not
<input checked="" type="radio"/> <input type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Both xv6 and ext2 contain magic number
<input type="radio"/> <input checked="" type="checkbox"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Ext2 contains group descriptors but xv6 does not

xv6 contains journal, ext2 does not: True

Ext2 contains superblock but xv6 does not.: False

In both ext2 and xv6, the superblock gives location of first inode block: False

xv6 contains inode bitmap, but ext2 does not: False

Both xv6 and ext2 contain magic number: False

Ext2 contains group descriptors but xv6 does not: True

Question 8

Correct

Mark 1.00 out of 1.00

An inode is read from disk as a part of this function

- a. iread
- b. readi
- c. iget
- d. sys_read
- e. ilock✓

Your answer is correct.

The correct answer is: ilock

Question 9

Correct

Mark 2.00 out of 2.00

Marks the statements as True/False w.r.t. "struct buf"

True	False	
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The reference count (refcnt) in struct buf is = number of processes accessing the buffer
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Lock on a buffer is acquired in bget, and released in brelse
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	The "next" pointer chain gives the buffers in LRU order
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	B_DIRTY flag means the buffer contains modified data
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	A buffer can be both on the MRU/LRU list and also on idequeue list.
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	A buffer can have both B_VALID and B_DIRTY flags set
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	B_VALID means the buffer is empty and can be reused
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	The buffers are maintained in LRU order, in the function brelse

The reference count (refcnt) in struct buf is = number of processes accessing the buffer: True

Lock on a buffer is acquired in bget, and released in brelse: True

The "next" pointer chain gives the buffers in LRU order: False

B_DIRTY flag means the buffer contains modified data: True

A buffer can be both on the MRU/LRU list and also on idequeue list.: True

A buffer can have both B_VALID and B_DIRTY flags set: False

B_VALID means the buffer is empty and can be reused: False

The buffers are maintained in LRU order, in the function brelse: True

Question 10

Partially correct

Mark 0.17 out of 1.00

Suppose an application on xv6 does the following:

```
int main() {  
    char arr[128];  
    int fd = open("README", O_RDONLY);  
    read(fd, arr, 100);  
}
```

Assume that the code works.

Which of the following things are true about xv6 kernel code, w.r.t. the above C program.

True False

<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The loop in readi() will always read a different block using bread()	✗
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	value of fd will be 3	✓
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The "memmove(dst, bp->data + off%BSIZE, m);" in readi() will copy the data from the disk to the kernel buffers	✗
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The process will be made to sleep only once	✗
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The ONLY function that gets called on return devsw[ip->major].read(ip, dst, n); is consoleread	✗
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The data is transferred from disk to kernel buffers first, and then address of arr is mapped to the kernel buffers	✗ No. data is copied into arr.

The loop in readi() will always read a different block using bread(): False
value of fd will be 3: True

The "memmove(dst, bp->data + off%BSIZE, m);" in readi() will copy the data from the disk to the kernel buffers: False

The process will be made to sleep only once: True

The ONLY function that gets called on return devsw[ip->major].read(ip, dst, n); is consoleread: True

The data is transferred from disk to kernel buffers first, and then address of arr is mapped to the kernel buffers: False

Question 11

Not answered

Marked out of 1.00

The lines

```
if(ip->type != T_DIR){  
    iunlockput(ip);  
    return 0;  
}
```

in namex() function

mean

- a. The last path component (which is a file, and not a directory) has been resolved, so release the lock (using iunlockput) and return
- b. No directory entry was found for the file to be opened, hence an error
- c. One of the sub-components on the given path name, was not a directory, hence it's an error
- d. One of the sub-components on the given path name, was a directory, but it was not supposed to be a directory, hence an error
- e. There was a syntax error in the pathname specified
- f. ilock is held on the inode, and hence it's an error if it is a directory
- g. One of the sub-components on the given path name, did not exist, hence it's an error

Your answer is incorrect.

The correct answer is: One of the sub-components on the given path name, was not a directory, hence it's an error

Question 12

Not answered

Marked out of 1.00

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

sys_chdir()	Choose...
skipelem	Choose...
ialloc	Choose...
namei	Choose...
bmap	Choose...
filestat()	Choose...
dirlookup	Choose...
balloc	Choose...
ideintr	Choose...
bread	Choose...
stati	Choose...
commit	Choose...

Your answer is incorrect.

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

Question 13

Not answered

Marked out of 1.00

Match function with it's functionality

dirlookup	Choose...
namex	Choose...
nameiparent	Choose...
dirlink	Choose...

Your answer is incorrect.

The correct answer is: dirlookup → Search a given name in a given directory, namex → return in-memory inode for a given pathname, nameiparent → return in-memory inode for parent directory of a given pathname, dirlink → Write a new entry in a given directory

[◀ Random Quiz - 5: xv6 make, bootloader, interrupt handling, memory management](#)

Jump to...

(Random Quiz - 7) Pre-Endsem Quiz ►

Started on Thursday, 9 March 2023, 6:20 PM

State Finished

Completed on Thursday, 9 March 2023, 7:23 PM

Time taken 1 hour 3 mins

Overdue 7 mins 42 secs

Grade 5.46 out of 10.00 (54.56%)

Question 1

Partially correct

Mark 0.15 out of 1.00

Consider the following command and its output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

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

```
kernel: $(OBJS) entry.o entryother initcode kernel.ld
$(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
$(OBJDUMP) -S kernel > kernel.asm
$(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* //; /^$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- a. Although the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- b. The kernel.asm file is the final kernel file ✗
- c. readseg() reads first 4k bytes of kernel in memory
- d. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- e. The bootmain() code does not read the kernel completely in memory
- f. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain().
- g. Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes. ✓
- h. The kernel.ld file contains instructions to the linker to link the kernel properly
- i. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.

Your answer is partially correct.

You have correctly selected 2.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(), readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question 2

Partially correct

Mark 0.20 out of 1.00

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

ljmp \$(SEG_KCODE<<3), \$start32
in bootasm.S

0x10000 to 0x7c00



jmp *%eax
in entry.S

Immaterial as the stack is not used here



cli
in bootasm.S

Immaterial as the stack is not used here



readseg((uchar*)elf, 4096, 0);
in bootmain.c

The 4KB area in kernel image, loaded in memory, named as 'stack'



call bootmain
in bootasm.S

0x10000 to 0x7c00



Your answer is partially correct.

You have correctly selected 1.

The correct answer is: ljmp \$(SEG_KCODE<<3), \$start32

in bootasm.S → Immortal as the stack is not used here, jmp *%eax

in entry.S → The 4KB area in kernel image, loaded in memory, named as 'stack', cli

in bootasm.S → Immortal as the stack is not used here, readseg((uchar*)elf, 4096, 0);

in bootmain.c → 0x7c00 to 0, call bootmain

in bootasm.S → 0x7c00 to 0

Question 3

Incorrect

Mark 0.00 out of 1.00

In bootasm.S, on the line

ljmp \$(SEG_KCODE<<3), \$start32

The SEG_KCODE << 3, that is shifting of 1 by 3 bits is done because

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

Your answer is incorrect.

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

Question 4

Correct

Mark 1.00 out of 1.00

What's the trapframe in xv6?

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

Your answer is correct.

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

Question 5

Partially correct

Mark 0.57 out of 1.00

Select all the correct statements about code of bootmain() in xv6

```
void
bootmain(void)
{
    struct elfhdr *elf;
    struct proghdr *ph, *eph;
    void (*entry)(void);
    uchar* pa;

    elf = (struct elfhdr*)0x10000; // scratch space

    // Read 1st page off disk
    readseg((uchar*)elf, 4096, 0);

    // Is this an ELF executable?
    if(elf->magic != ELF_MAGIC)
        return; // let bootasm.S handle error

    // Load each program segment (ignores ph flags).
    ph = (struct proghdr*)((uchar*)elf + elf->phoff);
    eph = ph + elf->phnum;
    for(; ph < eph; ph++){
        pa = (uchar*)ph->paddr;
        readseg(pa, ph->filesz, ph->off);
        if(ph->memsz > ph->filesz)
            stosb(pa + ph->filesz, 0, ph->memsz - ph->filesz);
    }

    // Call the entry point from the ELF header.
    // Does not return!
    entry = (void(*)(void))(elf->entry);
    entry();
}
```

Also, inspect the relevant parts of the xv6 code. binary files, etc and run commands as you deem fit to answer this question.

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

Your answer is partially correct.

You have correctly selected 4.

The correct answers are: The kernel file gets loaded at the Physical address 0x10000 in memory., The kernel file in memory is not necessarily a continuously filled in chunk, it may have holes in it., The elf->entry is set by the linker in the kernel file and it's 8010000c, The readseg finally invokes the disk I/O code using assembly instructions, The stosb() is used here, to fill in some space in memory with zeroes, The kernel ELF file contains actual physical address where particular sections of 'kernel' file should be loaded, The kernel file has only two program headers

Question 6

Partially correct

Mark 0.50 out of 1.00

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

Select all the appropriate choices

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

Your answer is partially correct.

You have correctly selected 1.

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

Question 7

Partially correct

Mark 0.50 out of 1.00

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

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

Your answer is partially correct.

You have correctly selected 3.

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

Question 8

Partially correct

Mark 0.91 out of 1.00

W.r.t. Memory management in xv6,

xv6 uses physical memory upto 224 MB only
Mark statements True or False

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

The switchkvm() call in scheduler() is invoked after control comes to it from swtch() scheduler(), thus demanding execution in new process's context: False

The switchkvm() call in scheduler() changes CR3 to use page directory of new process: False

PHYSTOP can be increased to some extent, simply by editing memlayout.h: True

The switchkvm() call in scheduler() changes CR3 to use page directory kpgdir: True

The process's address space gets mapped on frames, obtained from ~2MB:224MB range: True

The kernel's page table given by kpgdir variable is used as stack for scheduler's context: False

The kernel code and data take up less than 2 MB space: True

The free page-frame are created out of nearly 222 MB: True

The stack allocated in entry.S is used as stack for scheduler's context for first processor: True

The switchkvm() call in scheduler() is invoked after control comes to it from sched(), thus demanding execution in kernel's context: True

xv6 uses physical memory upto 224 MB only: True

Question 9

Partially correct

Mark 0.88 out of 1.00

Select the correct statements about interrupt handling in xv6 code

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

Your answer is partially correct.

You have correctly selected 7.

The correct answers are: All the 256 entries in the IDT are filled in xv6 code, Each entry in IDT essentially gives the values of CS and EIP to be used in handling that interrupt, xv6 uses the 64th entry in IDT for system calls, On any interrupt/syscall/exception the control first jumps in vectors.S, Before going to alltraps, the kernel stack contains upto 5 entries., The trapframe pointer in struct proc, points to a location on process's kernel stack, The function trap() is called even if any of the hardware interrupt/system-call/exception occurs, The CS and EIP are changed only after pushing user code's SS,ESP on stack

Question 10

Partially correct

Mark 0.75 out of 1.00

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

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

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

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: xv6.img is the virtual processor used by the qemu emulator, The xv6.img is of the size 10,000 blocks of 512 bytes each and occupies upto 10,000 blocks on the disk., The size of the kernel file is nearly 5 MB, The size of xv6.img is exactly = (size of bootblock) + (size of kernel)

[◀ Random Quiz 4 : Scheduling, signals, segmentation, paging, compilation, process-state](#)

Jump to...

[Random Quiz - 6 \(xv6 file system\) ▶](#)

Started on Thursday, 16 February 2023, 9:00 PM

State Finished

Completed on Thursday, 16 February 2023, 9:54 PM

Time taken 53 mins 39 secs

Grade 12.88 out of 15.00 (85.86%)

Question 1

Correct

Mark 1.00 out of 1.00

Mark whether the concept is related to scheduling or not.

Yes	No	
<input checked="" type="radio"/>	<input type="radio"/> 	ready-queue
<input checked="" type="radio"/>	<input type="radio"/> 	context-switch
<input checked="" type="radio"/>	<input type="radio"/> 	timer interrupt
<input checked="" type="radio"/>	<input type="radio"/> 	runnable process
<input type="radio"/> 	<input checked="" type="radio"/>	file-table

ready-queue: Yes

context-switch: Yes

timer interrupt: Yes

runnable process: Yes

file-table: No

Question 2

Partially correct

Mark 0.67 out of 1.00

Which of the following parts of a C program do not have any corresponding machine code ?

- a. #directives✓
- b. pointer dereference
- c. typedefs✓
- d. local variable declaration
- e. global variables
- f. function calls
- g. expressions

Your answer is partially correct.

You have correctly selected 2.

The correct answers: #directives, typedefs, global variables

Question 3

Partially correct

Mark 0.50 out of 1.00

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

Control is passed to P1

5	✓
---	---

timer interrupt occurs

4	✗
---	---

context of P1 is loaded from P1's PCB

3	✗
---	---

Process P0 is running

1	✓
---	---

Process P1 is running

6	✓
---	---

context of P0 is saved in P0's PCB

2	✗
---	---

Your answer is partially correct.

You have correctly selected 3.

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

Question 4

Partially correct

Mark 0.80 out of 1.00

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

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

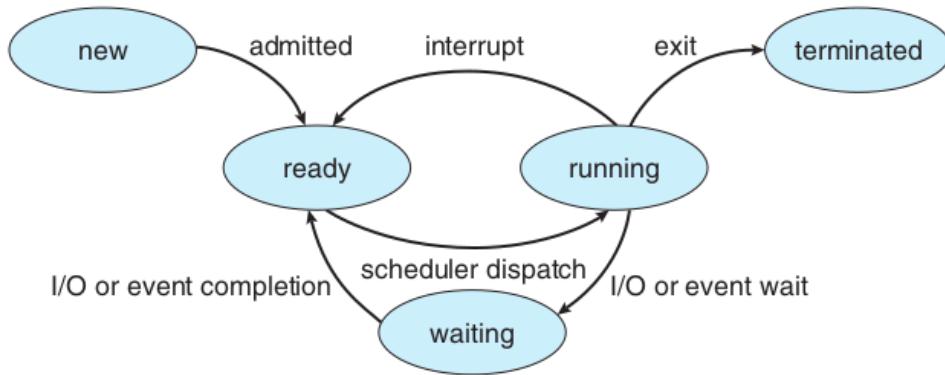


Figure 3.2 Diagram of process state.

True	False		
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	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.	✓
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	Every forked process has to go through ZOMBIE state, at least for a small duration.	✓
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet	✓
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first	✗
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	Only a process in READY state is considered by scheduler	✓

A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.: False

Every forked process has to go through ZOMBIE state, at least for a small duration.: True

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False

Only a process in READY state is considered by scheduler: True

Question 5

Correct

Mark 1.00 out of 1.00

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

- a. Invoke the linker to link the function calls with their code, extern globals with their declaration
- b. Process the # directives in a C program
- c. Check the program for syntactical errors
- d. Convert high level language code to machine code
- e. Check the program for logical errors ✓
- f. Suggest alternative pieces of code that can be written ✓

Your answer is correct.

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

Question 6

Correct

Mark 1.00 out of 1.00

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)

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

Your answer is correct.

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

Question 7

Partially correct

Mark 1.56 out of 2.00

Select all the correct statements about the state of a process.

- a. A process waiting for any condition is woken up by another process only
- b. A waiting process starts running after the wait is over
- c. A process that is running is not on the ready queue ✓
- d. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✓
- e. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers
- f. A running process may terminate, or go to wait or become ready again ✓
- g. Typically, it's represented as a number in the PCB
- h. A process can self-terminate only when it's running ✓
- i. Processes in the ready queue are in the ready state ✓
- j. A process changes from running to ready state on a timer interrupt or any I/O wait
- k. A process in ready state is ready to be scheduled ✓
- l. A process in ready state is ready to receive interrupts
- m. Changing from running state to waiting state results in "giving up the CPU"
- n. A process changes from running to ready state on a timer interrupt ✓

Your answer is partially correct.

You have correctly selected 7.

The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question 8

Correct

Mark 1.00 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

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

Your answer is correct.

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

Question 9

Partially correct

Mark 0.50 out of 1.00

Select all the correct statements about signals

Select one or more:

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

Your answer is partially correct.

You have selected too many options.

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

Question 10

Correct

Mark 1.00 out of 1.00

Map each signal with it's meaning

SIGCHLD	Child Stopped or Terminated	✓
SIGPIPE	Broken Pipe	✓
SIGALRM	Timer Signal from alarm()	✓
SIGUSR1	User Defined Signal	✓
SIGSEGV	Invalid Memory Reference	✓

The correct answer is: SIGCHLD → Child Stopped or Terminated, SIGPIPE → Broken Pipe, SIGALRM → Timer Signal from alarm(), SIGUSR1 → User Defined Signal, SIGSEGV → Invalid Memory Reference

Question 11

Correct

Mark 1.00 out of 1.00

Match the names of PCB structures with kernel

xv6	struct proc	✓
linux	struct task_struct	✓

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

Question 12

Partially correct

Mark 0.86 out of 1.00

Mark True/False

Statements about scheduling and scheduling algorithms

True False

<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.	<input checked="" type="checkbox"/>
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread	<input checked="" type="checkbox"/> It's the negation of this. Time NOT spent in idle thread.
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	A scheduling algorithm is non-preemptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	xv6 code does not care about Processor Affinity	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Response time will be quite poor on non-interruptible kernels	<input checked="" type="checkbox"/>
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Processor Affinity refers to memory accesses of a process being stored on cache of that processor	<input checked="" type="checkbox"/>

Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.: True

On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread: False

A scheduling algorithm is non-preemptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.: True

Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.: True

xv6 code does not care about Processor Affinity: True

Response time will be quite poor on non-interruptible kernels: True

Processor Affinity refers to memory accesses of a process being stored on cache of that processor: True

Question 13

Correct

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

- New: Running ✓
- Ready : Waiting ✓
- Running: : None of these ✓
- Waiting: Running ✓

Question 14

Correct

Mark 1.00 out of 1.00

Which of the following statements is false ?

Select one:

- a. Real time systems generally use non preemptive CPU scheduling. ✓
- b. A process scheduling algorithm is preemptive if the CPU can be forcibly removed from a process.
- c. Time sharing systems generally use preemptive CPU scheduling.
- d. Response time is more predictable in preemptive systems than in non preemptive systems.

Your answer is correct.

The correct answer is: Real time systems generally use non preemptive CPU scheduling.

[◀ Random Quiz - 3 \(processes, memory management, event driven kernel\), compilation-linking-loading, ipc-pipes](#)

Jump to...

[Random Quiz - 5: xv6 make, bootloader, interrupt handling, memory management ►](#)

Started on Thursday, 2 February 2023, 9:00 PM

State Finished

Completed on Thursday, 2 February 2023, 11:00 PM

Time taken 1 hour 59 mins

Grade **14.19** out of 20.00 (**70.93%**)

Question 1

Complete

Mark 0.50 out of 1.00

Select the sequence of events that are NOT possible, assuming an interruptible kernel code

Select one or more:

- a. P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again
- b. P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- c. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheduler
P1 running
P1's system call return
- d. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
scheduler
P2 running
- e. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running
- f. P1 running
P1 makes system call and blocks
Scheduler

P2 running
P2 makes system call and blocks
Scheduler
P1 running again

The correct answers are: P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again,
P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

Question 2

Complete

Mark 0.00 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. Program 1 makes sure that there is one file offset used for '2' and '1'
- b. Program 2 makes sure that there is one file offset used for '2' and '1'
- c. Both program 1 and 2 are incorrect
- d. Program 1 is correct for > /tmp/ddd but not for 2>&1
- e. Program 2 is correct for > /tmp/ddd but not for 2>&1
- f. Program 1 ensures 2>&1 and does not ensure > /tmp/ddd
- g. Program 2 ensures 2>&1 and does not ensure > /tmp/ddd
- h. Program 1 does 1>&2
- i. Only Program 1 is correct
- j. Both programs are correct
- k. Only Program 2 is correct
- l. Program 2 does 1>&2

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 3

Complete

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

New: Running

Ready : Waiting

Running: None of these

Waiting: Running

Question 4

Complete

Mark 4.75 out of 5.00

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
```

```
];
```

```
    pipe(
```

```
        pfd[0]
```

```
);
```

```
    pid1 =
```

```
        fork()
```

```
;
```

```
    if(pid1 != 0) {
```

```
        close(pfd[0]
```

```
        [0]
```

```
);
```

```
        close(
```

```
        1
```

```
);
```

```
        dup(
```

```
            pfd[0][1]
```

```
);
```

```
        execl("/bin/ls", "/bin/ls", "
```

```
        -l
```

```
        , NULL);
```

```
    }
```

```
    pipe(
```

```
        pfd[1]
```

```
);
```

```
    pid2
```

```
= fork();
```

```
    if(pid2 == 0) {
```

```
        close(
```

```
            pfd[0][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 5

Complete

Mark 1.00 out of 1.00

Select the order in which the various stages of a compiler execute.

Loading	does not exist
Linking	4
Pre-processing	1
Intermediate code generation	3
Syntactical Analysis	2

The correct answer is: Loading → does not exist, Linking → 4, Pre-processing → 1, Intermediate code generation → 3, Syntactical Analysis → 2

Question 6

Complete

Mark 0.00 out of 1.00

Select all the correct statements about named pipes and ordinary(unnamed) pipe

Select one or more:

- a. both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes
- b. named pipes are more efficient than ordinary pipes
- c. named pipe exists even if the processes using it do exit()
- d. ordinary pipe can only be used between related processes
- e. named pipes can be used between multiple processes but ordinary pipes can not be used
- f. a named pipe exists as a file on the file system
- g. named pipe can be used between any processes

The correct answers are: ordinary pipe can only be used between related processes, named pipe can be used between any processes, a named pipe exists as a file on the file system, named pipe exists even if the processes using it do exit(), both named and unnamed pipes require some kind of agreed protocol to be effectively used among multiple processes

Question 7

Complete

Mark 0.33 out of 1.00

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

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

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

Select one or more:

- a. P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
Scheduler running
P2 running
- b. P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P3 running
Hardware interrupt
Interrupt unblocks P1
Interrupt returns
P3 running
Timer interrupt
Scheduler
P1 running
- c. P1 running
P1 makes system call
timer interrupt
Scheduler
P2 running
timer interrupt
Scheduler
P1 running
P1's system call return
- d. P1 running
P1 makes system call and blocks
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again
- e. P1 running
keyboard hardware interrupt
keyboard interrupt handler running
interrupt handler returns
P1 running
P1 makes system call
system call returns
P1 running
timer interrupt
scheduler
P2 running

f.

P1 running
P1 makes system call
Scheduler
P2 running
P2 makes system call and blocks
Scheduler
P1 running again

The correct answers are: P1 running

P1 makes system call and blocks

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again, P1 running

P1 makes system call

timer interrupt

Scheduler

P2 running

timer interrupt

Scheuler

P1 running

P1's system call return,

P1 running

P1 makes system call

Scheduler

P2 running

P2 makes system call and blocks

Scheduler

P1 running again

Question 8

Complete

Mark 1.00 out of 1.00

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

- a. Convert high level language code to machine code
- b. Process the # directives in a C program
- c. Check the program for logical errors
- d. Check the program for syntactical errors
- e. Suggest alternative pieces of code that can be written
- f. Invoke the linker to link the function calls with their code, extern globals with their declaration

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

Question 9

Complete

Mark 1.40 out of 2.00

Match the elements of C program to their place in memory

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

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

Question 10

Complete

Mark 0.67 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

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

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

Question 11

Complete

Mark 0.29 out of 1.00

Order the events that occur on a timer interrupt:

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

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

Question 12

Complete

Mark 1.00 out of 1.00

Consider the following code and MAP the file to which each fd points at the end of the code.

```
int main(int argc, char *argv[]) {  
    int fd1, fd2 = 1, fd3 = 1, fd4 = 1;  
  
    fd1 = open("/tmp/1", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    fd2 = open("/tmp/2", O_RDONLY);  
    fd3 = open("/tmp/3", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);  
    close(0);  
    close(1);  
    dup(fd2);  
    dup(fd3);  
    close(fd3);  
    dup2(fd2, fd4);  
    printf("%d %d %d %d\n", fd1, fd2, fd3, fd4);  
    return 0;  
}
```

fd1	/tmp/1
fd2	/tmp/2
fd4	/tmp/2
0	/tmp/2
2	stderr
1	/tmp/3
fd3	closed

The correct answer is: fd1 → /tmp/1, fd2 → /tmp/2, fd4 → /tmp/2, 0 → /tmp/2, 2 → stderr, 1 → /tmp/3, fd3 → closed

Question 13

Complete

Mark 0.50 out of 1.00

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)

Paging	Many continuous chunks of same size
Segmentation, then paging	Many continuous chunks each of page size
Relocation + Limit	one continuous chunk
Segmentation	many continuous chunks of variable size

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

Question 14

Complete

Mark 0.75 out of 1.00

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

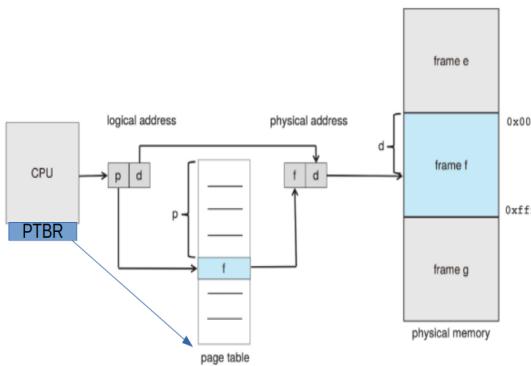


Figure 9.8 Paging hardware.

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

True False

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

Size of page table is always determined by the size of RAM: False

The PTBR is present in the CPU as a register: True

The page table is indexed using page number: True

The page table is indexed using frame number: False

Maximum Size of page table is determined by number of bits used for page number: True

The physical address may not be of the same size (in bits) as the logical address: True

The page table is itself present in Physical memory: True

The locating of the page table using PTBR also involves paging translation: False

Question 15

Complete

Mark 1.00 out of 1.00

A process blocks itself means

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

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

[◀ Random Quiz - 2: bootloader, system calls, fork-exec, open-read-write, linux-basics, processes](#)

Jump to...

[Random Quiz 4 : Scheduling, signals, segmentation, paging, compilation, process-state ►](#)

Started on Monday, 16 January 2023, 9:00 PM

State Finished

Completed on Monday, 16 January 2023, 10:05 PM

Time taken 1 hour 4 mins

Grade 11.52 out of 15.00 (76.78%)

Question 1

Correct

Mark 1.00 out of 1.00

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

- a. yes
- b. no ✓ wrong

The correct answer is: no

Question 2

Partially correct

Mark 0.67 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.

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

Your answer is partially correct.

You have correctly selected 2.

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 3

Correct

Mark 2.00 out of 2.00

What will this program do?

```
int main() {  
    fork();  
    execl("/bin/ls", "/bin/ls", NULL);  
    printf("hello");  
}
```

- a. run ls twice✓
- b. run ls twice and print hello twice
- c. run ls once
- d. one process will run ls, another will print hello
- e. run ls twice and print hello twice, but output will appear in some random order

Your answer is correct.

The correct answer is: run ls twice

Question 4

Correct

Mark 1.00 out of 1.00

Compare multiprogramming with multitasking

- a. A multitasking system is not necessarily multiprogramming
- b. A multiprogramming system is not necessarily multitasking✓

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

Question 5

Correct

Mark 1.00 out of 1.00

A process blocks itself means

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

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

Question 6

Correct

Mark 1.00 out of 1.00

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

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

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

Question 7

Correct

Mark 1.00 out of 1.00

When you turn your computer ON, you are often shown an option like "Press F9 for boot options". What does this mean?

- a. The choice of booting slowly or fast
- b. The choice of the boot loader (e.g. GRUB or Windows-Loader)
- c. The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded✓
- d. The choice of which OS to boot from

The correct answer is: The BIOS allows us to choose the boot device, the device from which the boot loader will be loaded

Question 8

Partially correct

Mark 0.83 out of 1.00

Select the correct statements about hard and soft links

Select one or more:

- a. Deleting a hard link always deletes the file
- b. Deleting a soft link deletes only the actual file
- c. Soft links increase the link count of the actual file inode
- d. Hard links increase the link count of the actual file inode✓
- e. Deleting a soft link deletes both the link and the actual file
- f. Deleting a hard link deletes the file, only if link count was 1✓
- g. Hard links enforce separation of filename from its metadata in on-disk data structures.
- h. Soft links can span across partitions while hard links can't✓
- i. Hard links can span across partitions while soft links can't
- j. Hard links share the inode✓
- k. Soft link shares the inode of actual file
- l. Deleting a soft link deletes the link, not the actual file✓

Your answer is partially correct.

You have correctly selected 5.

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

Question 9

Correct

Mark 1.00 out of 1.00

Consider the following programs

[exec1.c](#)

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

[exec2.c](#)

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

Compiled as

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

And run as

```
$ ./exec1
```

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

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

Select one:

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

Your answer is correct.

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

Question 10

Partially correct

Mark 0.60 out of 1.00

Select all the correct statements about two modes of CPU operation

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.

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

Question 11

Partially correct

Mark 0.67 out of 1.00

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

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

Your answer is partially correct.

You have correctly selected 4.

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

Question 12

Partially correct

Mark 0.75 out of 3.00

Select correct statements about mounting

Select one or more:

- a. 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
- b. The mount point must be a directory
- c. Mounting deletes all data at the mount-point
- d. Mounting makes all disk partitions available as one name space
- e. On Linuxes mounting can be done only while booting the OS
- f. It's possible to mount a partition on one computer, into namespace of another computer. ✓
- g. Mounting is attaching a disk-partition with a filesystem on it, into another file system name-space ✓
- h. The existing name-space at the mount-point is no longer visible after mounting
- i. The mount point can be a file as well ✗
- j. 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. ✓

Your answer is partially correct.

You have correctly selected 3.

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.

[◀ Random Quiz - 1 \(Pre-Requisite Quiz\)](#)

Jump to...

[Random Quiz - 3 \(processes, memory management, event driven kernel\), compilation-linking-loading, ipc-pipes ►](#)

Started on Friday, 17 March 2023, 2:29 PM

State Finished

Completed on Friday, 17 March 2023, 4:33 PM

Time taken 2 hours 3 mins

Grade 6.75 out of 10.00 (67.46%)

Question 1

Correct

Mark 0.50 out of 0.50

The struct buf has a sleeplock, and not a spinlock, because

- a. sleeplock is preferable because it is used in interrupt context and spinlock can not be used in interrupt context
- b. struct buf is used as a general purpose cache by kernel and cache operations take lot of time, so better to use sleeplock rather than spinlock
- c. It could be a spinlock, but xv6 has chosen sleeplock for purpose of demonstrating how to use a sleeplock.
- d. struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is preferred to spinning/busy-wait for the desired buf. ✓
- e. struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is the only option available.

Your answer is correct.

The correct answer is: struct buf is used for disk I/o which takes lot of time, so sleeping/blocking is preferred to spinning/busy-wait for the desired buf.

Question 2

Partially correct

Mark 0.91 out of 1.00

Given below is code of sleeplock in xv6.

```
// Long-term locks for processes
struct sleeplock {
    uint locked;          // Is the lock held?
    struct spinlock lk;  // spinlock protecting this sleep lock

    // For debugging:
    char *name;           // Name of lock.
    int pid;              // Process holding lock
};
```

```
void
acquiresleep(struct sleeplock *lk)
{
    acquire(&lk->lk);
    while (lk->locked) {
        sleep(lk, &lk->lk);
    }
    lk->locked = 1;
    lk->pid = myproc()->pid;
    release(&lk->lk);
}
```

```
void
releasesleep(struct sleeplock *lk)
{
    acquire(&lk->lk);
    lk->locked = 0;
    lk->pid = 0;
    wakeup(lk);
    release(&lk->lk);
}
```

Mark the statements as True/False w.r.t. this code.

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	The spinlock lk->lk is held when the process comes out of sleep()
<input type="radio"/>	<input checked="" type="radio"/>	sleep() is called holding a spinlock. This could be avoided by releasing the lock before calling sleep() and acquiring it again after call to sleep()
<input type="radio"/>	<input checked="" type="radio"/>	acquire(&lk->lk); while (!lk->locked) { sleep(lk, &lk->lk); } could also be written as acquire(&lk->lk); if (!lk->locked) { sleep(lk, &lk->lk); }
<input checked="" type="radio"/>	<input type="radio"/>	All processes waiting for the sleeplock will have a race for aquiring lk->lk spinlock, because all are woken up

True False

<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/> <input type="radio"/>	the 'spinlock lk' protects 'locked' variable, but not the 'name' nor the 'pid'	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> <input type="radio"/>	<input type="radio"/>	sleep() is the function which blocks a process.	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> <input type="radio"/>	<input type="radio"/>	the 'spinlock lk' is needed in a sleeplock, because access to the sleeplock for locking/unlocking itself creates a critical section	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> <input type="radio"/>	<input type="radio"/>	Sleeplock() will ensure that either the process gets the lock or the process gets blocked.	<input checked="" type="checkbox"/>
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Wakeup() will wakeup the first process waiting for the lock	<input checked="" type="checkbox"/> Wakeup() will wakeup all processes waiting for the lock
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	A process has acquired the sleeplock when it comes out of sleep()	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> <input type="radio"/>	<input type="radio"/>	The process which called acquiresleep() and then got blocked, is woken up by the timer interrupt	<input checked="" type="checkbox"/> it's woken up by another process which called releasesleep() and then wakeup()

The spinlock lk->lk is held when the process comes out of sleep(): True

sleep() is called holding a spinlock. This could be avoided by releasing the lock before calling sleep() and acquiring it again after call to sleep():

False

```
acquire(&lk->lk);
while (!lk->locked) {
    sleep(lk, &lk->lk);
}
```

could also be written as

```
acquire(&lk->lk);
if (!lk->locked) {
    sleep(lk, &lk->lk);
}
```

: False

All processes waiting for the sleeplock will have a race for aquiring lk->lk spinlock, because all are woken up: True

the 'spinlock lk' protects 'locked' variable, but not the 'name' nor the 'pid': False

sleep() is the function which blocks a process.: True

the 'spinlock lk' is needed in a sleeplock, because access to the sleeplock for locking/unlocking itself creates a critical section: True

Sleeplock() will ensure that either the process gets the lock or the process gets blocked.: True

Wakeup() will wakeup the first process waiting for the lock: False

A process has acquired the sleeplock when it comes out of sleep(): False

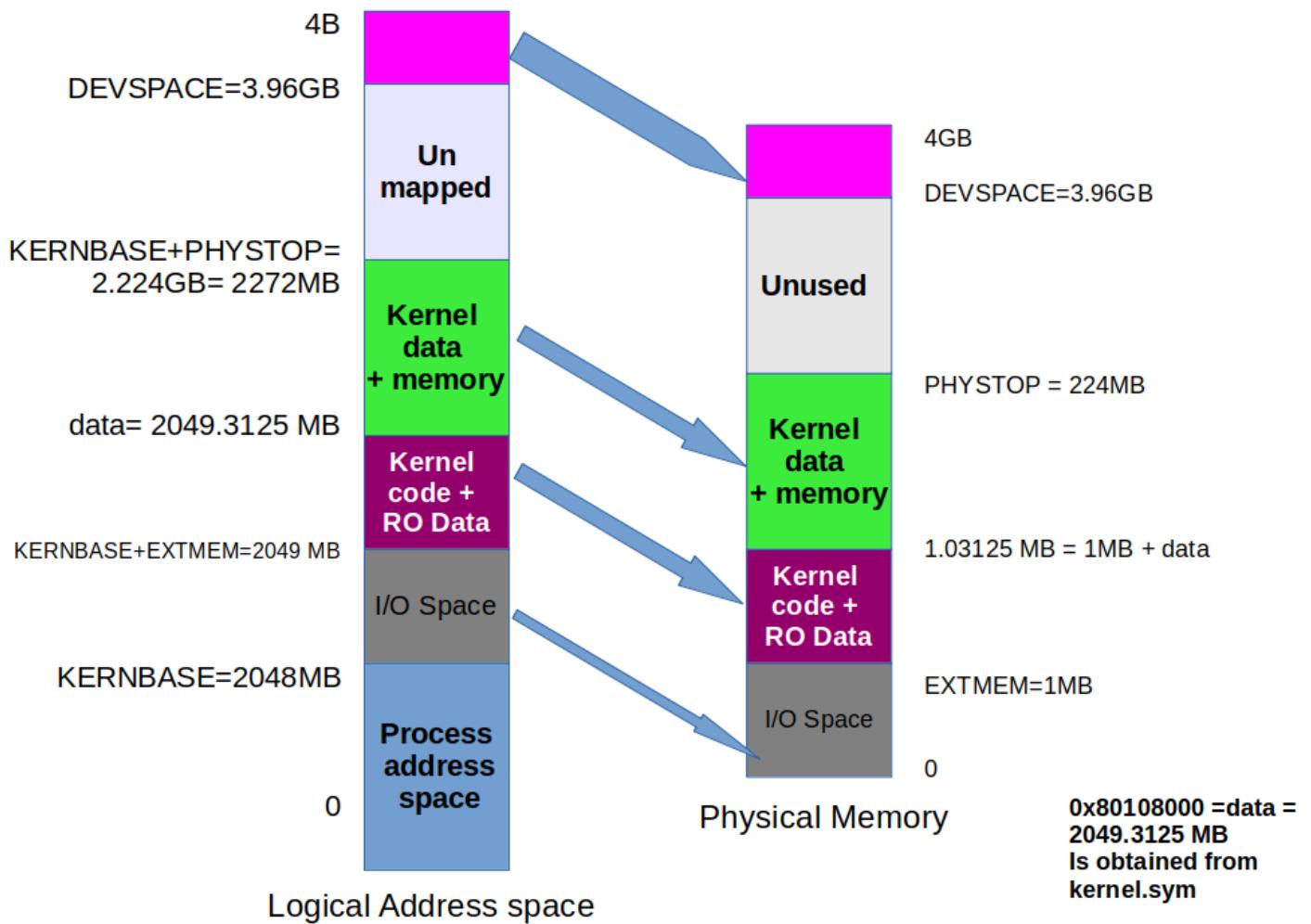
The process which called acquiresleep() and then got blocked, is woken up by the timer interrupt: True

Question 3

Partially correct

Mark 0.36 out of 0.50

With respect to this diagram, mark statements as True/False.



True False

<input checked="" type="radio"/>	<input type="radio"/>	This diagram only shows the absolutely defined virtual->physical mappings, not the mappings defined at run time by kernel.	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/>	The kernel virtual addresses start from $\text{KERNLINK} = \text{KERNBASE} + \text{EXTMEM}$	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input type="radio"/>	PHYSTOP can be changed , but that needs kernel recompilation and re-execution.	<input checked="" type="checkbox"/>
<input type="radio"/>	<input checked="" type="radio"/>	"Kernel data + memory" on right side, here refers to the region from which pages are allocated to the kernel and process both.	<input checked="" type="checkbox"/> "Kernel data + memory" on LEFT side, here refers to the virtual addresses of kernel used at run time.
<input checked="" type="radio"/>	<input checked="" type="radio"/>	The process's pages are mapped into physical memory from 1.03125 MB to PHYSTOP .	<input checked="" type="checkbox"/>
<input checked="" type="radio"/>	<input checked="" type="radio"/>	When bootloader loads the kernel, then physical memory from EXTMEM upto $\text{EXTMEM} + \text{data}$ is occupied.	<input checked="" type="checkbox"/>

True **False**



The kernel file, after compilation, has maximum virtual address up to "data" as shown in the diagram, which is equal to "end" variable



This diagram only shows the absolutely defined virtual->physical mappings, not the mappings defined at run time by kernel.: True

The kernel virtual addresses start from KERNLINK = KERNBASE + EXTMEM: True

PHYSTOP can be changed , but that needs kernel recompilation and re-execution.: True

"Kernel data + memory" on right side, here refers to the region from which pages are allocated to the kernel and process both.: True

The process's pages are mapped into physical memory from 1.03125 MB to PHYSTOP.: True

When bootloader loads the kernel, then physical memory from EXTMEM upto EXTMEM + data is occupied.: True

The kernel file, after compilation, has maximum virtual address up to "data" as shown in the diagram, which is equal to "end" variable: True

Question 4

Incorrect

Mark 0.00 out of 0.25

Select the odd one out

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

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

Question 5

Partially correct

Mark 0.80 out of 1.00

Mark the statements as True/False w.r.t. swtch()

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	swtch() is written in assembly language, because it violates calling convention, by changing the stack itself.
<input checked="" type="radio"/>	<input type="radio"/>	push in swtch() happens on old stack, while pop happens from new stack
<input type="radio"/>	<input checked="" type="radio"/>	swtch() called from scheduler() changes the stack from the process's kernel stack to the scheduler's kernel stack.
<input type="radio"/>	<input checked="" type="radio"/>	switch stores the old context on new stack, and restores new context from old stack.
<input checked="" type="radio"/>	<input type="radio"/>	p->context used in scheduler()->swtch() was Generally set when the process was interrupted earlier, and came via sched()->swtch()
<input checked="" type="radio"/>	<input type="radio"/>	swtch() changes the context from "old" to "new"
<input type="radio"/>	<input checked="" type="radio"/>	sched() is the only place when p->context is set
<input type="radio"/>	<input checked="" type="radio"/>	swtch() is written in assembly language because it violates the calling convention by pushing parameters on the stack on its own.
<input type="radio"/>	<input checked="" type="radio"/>	movl %esp, (%eax) means, *(c->scheduler) = contents of esp When swtch() is called from scheduler()
<input checked="" type="radio"/>	<input type="radio"/>	swtch() is called only from sched() or scheduler()

swtch() is written in assembly language, because it violates calling convention, by changing the stack itself.: True

push in swtch() happens on old stack, while pop happens from new stack: True

swtch() called from scheduler() changes the stack from the process's kernel stack to the scheduler's kernel stack.: False

switch stores the old context on new stack, and restores new context from old stack.: False

p->context used in scheduler()->swtch() was **Generally** set when the process was interrupted earlier, and came via sched()->swtch(): True

swtch() changes the context from "old" to "new": True

sched() is the only place when p->context is set: False

swtch() is written in assembly language because it violates the calling convention by pushing parameters on the stack on its own.: False

movl %esp, (%eax)

means, *(c->scheduler) = contents of esp

When swtch() is called from scheduler(): False

swtch() is called only from sched() or scheduler(): True

Question 6

Partially correct

Mark 0.44 out of 0.50

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

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

when chan is NULL, the 'state' in proc must be RUNNABLE.: False

in xv6, the address of an appropriate variable is used as a "condition" for a waiting process.: True

When chan is not NULL, the 'state' in struct proc must be SLEEPING: True

Changing the state of a process automatically changes the value of 'chan': False

chan is the head pointer to a linked list of processes, waiting for a particular event to occur: False

chan stores the address of the variable, representing a condition, for which the process is waiting.: True

The value of 'chan' is changed only in sleep(): True

'chan' is used only by the sleep() and wakeup1() functions.: True

Question 7

Partially correct

Mark 0.15 out of 0.25

Match function with its meaning

ideintr	disk interrupt handler, transfer data from controller to buffer, wake up processes waiting for this buffer, start I/O for next buffer	✓
idewait	Wait for disc controller to be ready	✓
ideinit	Initialize the disc controller	✓
iderw	tell disc controller to complete I/O for all pending requests	✗
idestart	Issue a disk read/write for a buffer, block the issuing process	✗

Your answer is partially correct.

You have correctly selected 3.

The correct answer is: ideintr → disk interrupt handler, transfer data from controller to buffer, wake up processes waiting for this buffer, start I/O for next buffer, idewait → Wait for disc controller to be ready, ideinit → Initialize the disc controller, iderw → Issue a disk read/write for a buffer, block the issuing process, idestart → tell disc controller to start I/O for the first buffer on idequeue

Question 8

Partially correct

Mark 0.25 out of 0.50

when is each of the following stacks allocated?

kernel stack of process	during fork() in allocproc()	✓
kernel stack for scheduler, on first processor	in entry.S	✓
user stack of process	during exec()	✗
kernel stack for the scheduler, on other processors	in entry.S	✗

Your answer is partially correct.

You have correctly selected 2.

The correct answer is: kernel stack of process → during fork() in allocproc(), kernel stack for scheduler, on first processor → in entry.S, user stack of process → during fork() in copyuvm(), kernel stack for the scheduler, on other processors → in main()->startothers()

Question 9

Correct

Mark 0.25 out of 0.25

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

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

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

Question 10

Correct

Mark 0.25 out of 0.25

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

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

The correct answer is: 801154a8

Question 11

Partially correct

Mark 0.23 out of 0.50

Which of the following is DONE by allocproc() ?

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

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

Question 12

Incorrect

Mark 0.00 out of 0.50

The first instruction that runs when you do "make qemu" is

cli

from bootasm.S

Why?

- a. "cli" clears all registers and makes them zero, so that processor is as good as "new"
- b. It disables interrupts. It is required because the interrupt handlers of kernel are not yet installed.
- c. "cli" that is Command Line Interface needs to be enabled first
- d. "cli" enables interrupts, it is required because the kernel supports interrupts.
- e. "cli" stands for clear screen and the screen should be cleared before OS boots.
- f. "cli" clears the pipeline of the CPU so that it is as good as "fresh" CPU
- g. "cli" disables interrupts. It is required because as of now there are no interrupt handlers available X
- h. "cli" enables interrupts, it is required because the kernel must handle interrupts.

Your answer is incorrect.

The correct answer is: It disables interrupts. It is required because the interrupt handlers of kernel are not yet installed.

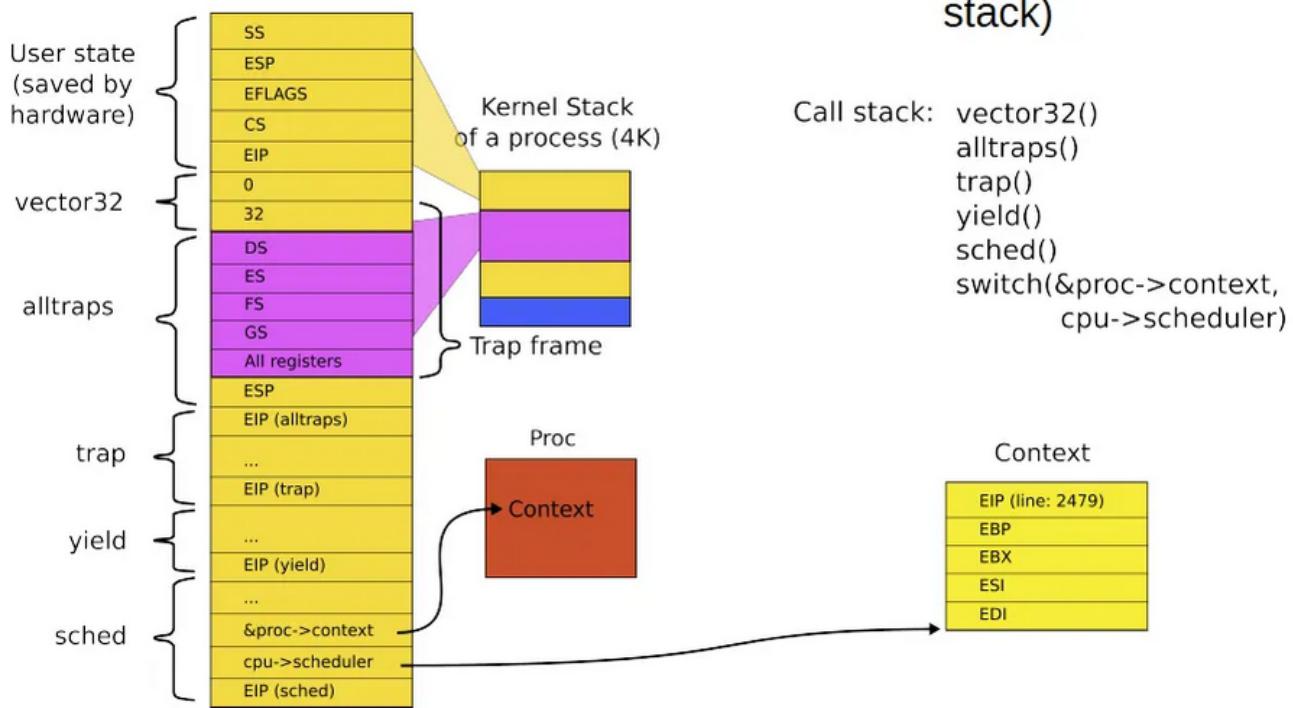
Question 13

Partially correct

Mark 0.42 out of 0.50

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

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



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

True False

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

This is a diagram of swtch() called from scheduler(): False

The "ESP" (second entry from top) is stack pointer of user-stack of process, while the "ESP" (first entry below pink region) is the trapframe pointer on kernel stack of process.: True

The diagram is wrong because it shows the user stack and kernel stack together (continuous), but in practice they are separate: False

The diagram is correct: True

The blue shaded part in "kernel stack of a process(4k)" refers to remaining part of stack (not used yet): True
The "context" yellow coloured box, pointed to by cpu->scheduler is on the kernel stack of the scheduler.: False

Question 14

Partially correct

Mark 0.66 out of 0.75

Mark statements as True/False w.r.t. ptable.lock

True	False	
<input checked="" type="radio"/>	<input type="radio"/>	One sequence of function calls which takes and releases the ptable.lock is this: iderw->sleep, acquire(ptable.lock)->sched->swtch()->scheduler()->swtch()->yield(), release(ptable.lock)
<input checked="" type="radio"/>	<input type="radio"/>	It is taken by one process but released by another process, running on same processor
<input type="radio"/>	<input checked="" type="radio"/>	A process can sleep on ptable.lock if it can't acquire it.
<input checked="" type="radio"/>	<input type="radio"/>	ptable.lock protects the proc[] array and all struct proc in the array
<input type="radio"/>	<input checked="" type="radio"/>	ptable.lock is acquired but never released
<input type="radio"/>	<input checked="" type="radio"/>	The swtch() in scheduler() is called without holding the ptable.lock when control jumps to it from sched()
<input type="radio"/>	<input checked="" type="radio"/>	ptable.lock can be held by different processes on different processors at the same time
<input checked="" type="radio"/>	<input type="radio"/>	the rule of "never block holding a spinlock" does not apply to ptable.lock in xv6

One sequence of function calls which takes and releases the ptable.lock is this:
iderw->sleep, acquire(ptable.lock)->sched->swtch()->scheduler()->swtch()->yield(), release(ptable.lock): True
It is taken by one process but released by another process, running on same processor: True
A process can sleep on ptable.lock if it can't acquire it.: False
ptable.lock protects the proc[] array and all struct proc in the array: True
ptable.lock is acquired but never released: False
The swtch() in scheduler() is called without holding the ptable.lock when control jumps to it from sched(): False
ptable.lock can be held by different processes on different processors at the same time: False
the rule of "never block holding a spinlock" does not apply to ptable.lock in xv6: True

Question 15

Incorrect

Mark 0.00 out of 0.25

Why is there a call to knit2? Why is it not merged with knit1?

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

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

Question 16

Partially correct

Mark 0.54 out of 0.75

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

movl \$(V2P_WO(entrypgdir)), %eax	protected mode with segmentation and 4 MB pages	✗
movw %ax, %gs	protected mode with only segmentation	✓
ljmp \$(SEG_KCODE<<3), \$start32	real mode	✓
inb \$0x64,%al	real mode	✓
readseg((uchar*)elf, 4096, 0);	protected mode with only segmentation	✓
orl \$CR0_PE, %eax	protected mode with segmentation and 4 MB pages	✗
jmp *%eax	protected mode with segmentation and 4 MB pages	✓

The correct answer is: movl \$(V2P_WO(entrypgdir)), %eax → protected mode with only segmentation, movw %ax, %gs → protected mode with only segmentation, ljmp \$(SEG_KCODE<<3), \$start32 → real mode, inb \$0x64,%al → real mode, readseg((uchar*)elf, 4096, 0); → protected mode with only segmentation, orl \$CR0_PE, %eax → real mode, jmp *%eax → protected mode with segmentation and 4 MB pages

Question 17

Incorrect

Mark 0.00 out of 0.50

We often use terms like "swtch() changes stack from process's kernel stack to scheduler's stack", or "the values are pushed on stack", or "the stack is initialized to the new page", etc. while discussing xv6 on x86.

Which of the following most accurately describes the meaning of "stack" in such sentences?

- a. the region of memory which is currently used as stack by processor X
- b. The stack segment
- c. The ss:esp pair
- d. The region of memory where the kernel remembers all the function calls made
- e. The "stack" variable declared in "stack.S" in xv6
- f. The stack variable used in the program being discussed
- g. The region of memory allocated by kernel for storing the parameters of functions

Your answer is incorrect.

The correct answer is: The ss:esp pair

Question 18

Incorrect

Mark 0.00 out of 0.25

Which of the following call sequence is impossible in xv6?

- a. Process 1: timer interrupt -> trap() -> yield() -> sched() -> switch() -> scheduler() -> Process 2 runs -> write -> sys_write() -> trap() -> ...
- b. Process 1: write() -> sys_write() -> file_write() -> writei() -> bread() -> bget() -> iderw() -> sleep() -> sched() -> switch() (jumps to) -> scheduler() -> swtch() (jumps to) -> Process 2 (return call sequence) sched() -> yield() -> trap -> user-code
- c. Process 1: write() -> sys_write() -> file_write() -- timer interrupt -> trap() -> yield() -> sched() -> switch() (jumps to) -> scheduler() -> swtch() X (jumps to) -> Process 2 (return call sequence) sched() -> yield() -> trap -> user-code

Your answer is incorrect.

The correct answer is: Process 1: timer interrupt -> trap() -> yield() -> sched() -> switch() -> scheduler() -> Process 2 runs -> write -> sys_write() -> trap() -> ...

Question 19

Correct

Mark 0.50 out of 0.50

Consider the following command and its output:

```
$ ls -lht xv6.img kernel
-rw-rw-r-- 1 abhijit abhijit 4.9M Feb 15 11:09 xv6.img
-rwxrwxr-x 1 abhijit abhijit 209K Feb 15 11:09 kernel*
```

Following code in bootmain()

```
readseg((uchar*)elf, 4096, 0);
```

and following selected lines from Makefile

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

```
kernel: $(OBJS) entry.o entryother initcode kernel.ld
$(LD) $(LDFLAGS) -T kernel.ld -o kernel entry.o $(OBJS) -b binary initcode entryother
$(OBJDUMP) -S kernel > kernel.asm
$(OBJDUMP) -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
```

Also read the code of bootmain() in xv6 kernel.

Select the options that describe the meaning of these lines and their correlation.

- a. readseg() reads first 4k bytes of kernel in memory ✓
- b. Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes. ✓
- c. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(). ✓
- d. Although the size of the kernel file is 209 Kb, only 4Kb out of it is the actual kernel code and remaining part is all zeroes.
- e. The kernel.ld file contains instructions to the linker to link the kernel properly ✓
- f. The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files ✓
- g. The kernel.asm file is the final kernel file
- h. The bootmain() code does not read the kernel completely in memory
- i. The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is not read as it is user programs.

Your answer is correct.

The correct answers are: The kernel disk image is ~5MB, the kernel within it is 209 kb, but bootmain() initially reads only first 4kb, and the later part is read using program headers in bootmain(), readseg() reads first 4k bytes of kernel in memory, The kernel is compiled by linking multiple .o files created from .c files; and the entry.o, initcode, entryother files, The kernel.ld file contains instructions to the linker to link the kernel properly, Although the size of the xv6.img file is ~5MB, only some part out of it is the bootloader+kernel code and remaining part is all zeroes.

Question 20

Correct

Mark 0.50 out of 0.50

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

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

the kmem.lock is used by kfree() and kalloc() only.: True

if(kmem.use_lock)

acquire(&kmem.lock);

this "if" condition is true, when kinit2() runs because multi-processor support has been enabled by now.: False

free page list is a singly circular linked list.: False

kmem.use_lock is set to 1 after free page list is created, so that kmem.lock is taken before accessing kmem.freelist.: True

The pointers that link the pages together are in the first 4 bytes of the pages themselves: True

if(kmem.use_lock)

acquire(&kmem.lock);

is not done when called from kinit1() because there is no need to take the lock when kinit1() is running because interrupts are disabled and only one processor is running: True

[◀ Quiz-1\(24 Feb 2023\)](#)

Jump to...

[Pre-requisite Quiz \(old\) - use it for practice ►](#)

Started on Friday, 24 February 2023, 2:44 PM

State Finished

Completed on Friday, 24 February 2023, 4:26 PM

Time taken 1 hour 42 mins

Grade 9.53 out of 10.00 (95.27%)

Question 1

Correct

Mark 0.50 out of 0.50

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

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 2

Partially correct

Mark 0.36 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	
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	On Linux, the message is printed only because the memory management scheme is segmentation
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	On Linux, the process was sent a SIGSEGV signal and the default handler for the signal is "Term", so the process is terminated.
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The term "core" refers to the core code of the kernel.
<input type="radio"/> ✗	<input checked="" type="radio"/> ✗	The illegal memory access was detected by the kernel and the process was punished by kernel.
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The image of the process is stored in a file called "core", if the ulimit allows so.
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The core file can be analysed later using a debugger, to determine what went wrong.
<input checked="" type="radio"/> ✗	<input type="radio"/> ✗	The process has definitely performed illegal memory access.

On Linux, the message is printed only because the memory management scheme is segmentation.: 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

The term "core" refers to the core code of the kernel.: False

The illegal memory access was detected by the kernel and the process was punished by kernel.: False

The image of the process is stored in a file called "core", if the ulimit allows so.: True

The core file can be analysed later using a debugger, to determine what went wrong.: True

The process has definitely performed illegal memory access.: True

Question 3

Correct

Mark 0.50 out of 0.50

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

Select one:

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

Your answer is correct.

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

Question 4

Correct

Mark 0.50 out of 0.50

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

Write the answer as a number.

The correct answer is: 6

Question 5

Correct

Mark 0.50 out of 0.50

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"

One or more data bitmap blocks for the parent directory

 No ✓

Superblock

 Yes ✓

Possibly one block bitmap corresponding to the parent directory

 Yes ✓

One or multiple data blocks of the parent directory

 No ✓

Data blocks of the file

 No ✓

Block bitmap(s) for all the blocks of the file

 Yes ✓

Your answer is correct.

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

Question 6

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 ✓

Paging

 one continuous chunk ✓

Segmentation, then paging

 many continuous chunks of variable size ✓

Your answer is correct.

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

Question 7

Partially correct

Mark 0.36 out of 0.50

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

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

The kernel refers to the page table for converting each virtual address to physical address.: False

When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.: True

The compiler generates the address references for code/data/stack/heap in the executable file as per the memory management schema chosen by the compiler itself, and then the kernel ensures that program is executed with this schema.: False

The compiler interacts with the kernel continuously while compiling a program and obtains the correct set of memory addresses for code/stack/heap/data and then generates the machine code file.: False

The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True

The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.: True

When a process is executing, each virtual address is converted into physical address by the kernel directly.: False

Question 8

Correct

Mark 0.50 out of 0.50

Map each signal with its meaning

SIGCHLD	Child Stopped or Terminated	✓
SIGSEGV	Invalid Memory Reference	✓
SIGPIPE	Broken Pipe	✓
SIGUSR1	User Defined Signal	✓
SIGALRM	Timer Signal from alarm()	✓

The correct answer is: SIGCHLD → Child Stopped or Terminated, SIGSEGV → Invalid Memory Reference, SIGPIPE → Broken Pipe, SIGUSR1 → User Defined Signal, SIGALRM → Timer Signal from alarm()

Question 9

Partially correct

Mark 0.45 out of 0.50

Match the elements of C program to their place in memory

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

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

Question 10

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

Answer: hi one one



The correct answer is: hi one one

Question 11

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)

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 12

Partially correct

Mark 0.44 out of 0.50

How does the compiler calculate addresses for the different parts of a C program, when paging is used?

Global variables	Immediately after the text	✓
Static variables	Immediately after the text, along with globals	✓
#include files	No memory allocated, they are handled by linker	✗
malloced memory	Heap (handled by the malloc-free library, using OS's system calls)	✓
typedef	No memory allocated, as they are not variables, but only conceptual definition of a type	✓
#define	No memory allocated, they are handled by pre-processor	✓
Local variables	An offset with respect to stack pointer (esp)	✓
Text	starting with 0	✓

Your answer is partially correct.

You have correctly selected 7.

The correct answer is: Global variables → Immediately after the text, Static variables → Immediately after the text, along with globals, #include files → No memory allocated for the file, but if it contains variables, then variables may be allocated memory, malloced memory → Heap (handled by the malloc-free library, using OS's system calls), typedef → No memory allocated, as they are not variables, but only conceptual definition of a type, #define → No memory allocated, they are handled by pre-processor, Local variables → An offset with respect to stack pointer (esp), Text → starting with 0

Question 13

Correct

Mark 0.50 out of 0.50

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

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

A named pipe has a name decided by the kernel.: False

Named pipes can exist beyond the life-time of processes using them.: True

Un-named pipes can be used for communication between only "related" processes, if the common ancestor created it.: True

The buffers for named-pipe are in process-memory while the buffers for the un-named pipe are in kernel memory.: False

Both types of pipes provide FIFO communication.: True

The pipe() system call can be used to create either a named or un-named pipe.: False

Named pipes can be used for communication between only "related" processes.: False

Un-named pipes are inherited by a child process from parent.: True

Both types of pipes are an extension of the idea of "message passing": True

Named pipe exists as a file: True

Question 14

Partially correct

Mark 0.45 out of 0.50

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

Select No otherwise.

Yes	No	
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	List of opened files
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	EIP at the time of context switch
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Pointer to IDT
<input checked="" type="radio"/>	<input checked="" type="radio"/> <input type="checkbox"/>	Pointer to the parent process
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	PID
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Memory management information about that process
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Process context
<input checked="" type="radio"/>	<input type="radio"/> <input checked="" type="checkbox"/>	Process state
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	PID of Init
<input type="radio"/> <input checked="" type="checkbox"/>	<input checked="" type="radio"/>	Function pointers to all system calls

List of opened files: Yes

EIP at the time of context switch: Yes

Pointer to IDT: No

Pointer to the parent process: Yes

PID: Yes

Memory management information about that process: Yes

Process context: Yes

Process state: Yes

PID of Init: No

Function pointers to all system calls: No

Question 15

Correct

Mark 0.50 out of 0.50

Which of the following parts of a C program do not have any corresponding machine code ?

- a. #directives✓
- b. function calls
- c. local variable declaration
- d. pointer dereference
- e. global variables✓
- f. typedefs✓
- g. expressions

Your answer is correct.

The correct answers are: #directives, typedefs, global variables

Question 16

Correct

Mark 0.50 out of 0.50

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

True False

<input checked="" type="radio"/>	<input type="radio"/> X	Device driver is an intermediary between the hardware controller and OS	✓
<input type="radio"/> X	<input checked="" type="radio"/>	Device driver is part of hardware	✓
<input checked="" type="radio"/>	<input type="radio"/> X	Device driver is part of OS code	✓
<input type="radio"/> X	<input checked="" type="radio"/>	Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.	✓
<input checked="" type="radio"/>	<input type="radio"/> X	It's possible that a particular hardware has multiple device drivers available for it.	✓
<input checked="" type="radio"/>	<input type="radio"/> X	Writing a device driver mandatorily demands reading the technical documentation about the hardware.	✓

Device driver is an intermediary between the hardware controller and OS.: True

Device driver is part of hardware: False

Device driver is part of OS code: True

Different devices of the same type (e.g. 2 IDE hard disks) must need different device drivers.: False

It's possible that a particular hardware has multiple device drivers available for it.: True

Writing a device driver mandatorily demands reading the technical documentation about the hardware.: True

Question 17

Partially correct

Mark 0.48 out of 0.50

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", "
```

```
        -l
```

```
    );
```

```
    }
```

```
    pipe(
```

```
    pfd[1]
```

```
    );
```

```
    pid2
```

```
    = fork();
```

```
    if(pid2 == 0) {
```

```
        close(
```

```
        pfd[0][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 18

Correct

Mark 0.50 out of 0.50

What is meant by formatting a disk/partition?

- a. storing all the necessary programs on the disk/partition
- b. erasing all data on the disk/partition
- c. writing zeroes on all sectors
- d. creating layout of empty directory tree/graph data structure✓

The correct answer is: creating layout of empty directory tree/graph data structure

Question 19

Correct

Mark 0.50 out of 0.50

Which of the following instructions should be privileged?

Select one or more:

- a. Read the clock.
- b. Access memory management unit of the processor ✓
- c. Set value of a memory location
- d. Turn off interrupts. ✓
- e. Set value of timer. ✓
- f. Access a general purpose register
- g. Access I/O device. ✓
- h. Switch from user to kernel mode. ✓ This instruction (like INT) is itself privileged - and that is why it not only changes the mode, but also ensures a jump to an ISR (kernel code)
- i. Modify entries in device-status table ✓

Your answer is correct.

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

Question 20

Correct

Mark 0.50 out of 0.50

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

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

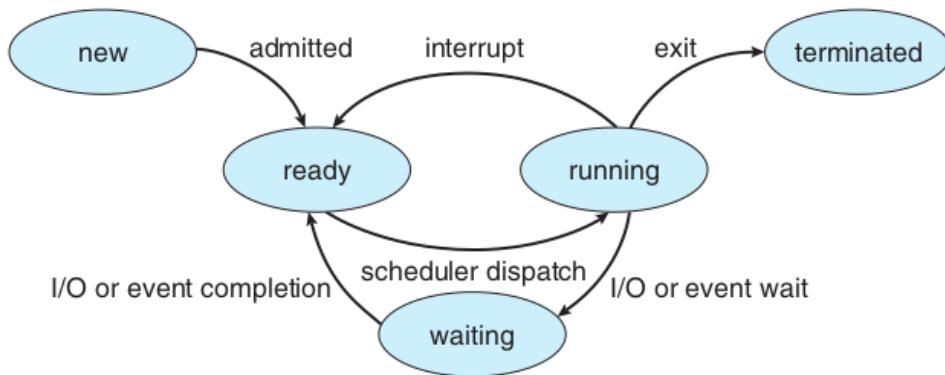


Figure 3.2 Diagram of process state.

True	False
<input checked="" type="radio"/>	<input type="radio"/> ✗
<input type="radio"/> ✗	<input checked="" type="radio"/>
<input type="radio"/> ✗	<input checked="" type="radio"/>
<input type="radio"/> ✗	<input checked="" type="radio"/>
<input checked="" type="radio"/>	<input type="radio"/> ✗
<input checked="" type="radio"/>	<input type="radio"/> ✗

Only a process in READY state is considered by scheduler



A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first



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.



Every forked process has to go through ZOMBIE state, at least for a small duration.



A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet



Only a process in READY state is considered by scheduler: True

A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: 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

Every forked process has to go through ZOMBIE state, at least for a small duration.: True

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True

[◀ Quiz-1 Preparation questions](#)

Jump to...