Started on	Saturday, 30 April 2022, 2:05:41 PM
State	Finished
Completed on	Saturday, 30 April 2022, 6:31:46 PM
Time taken	4 hours 26 mins
Grade	**30.01** out of 40.00 (**75**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Predict the output of the program given here.

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`

--

```
int main() {
  int pid;
  printf("hi\n");
  pid = fork();
  if(pid == 0) {
    exit(0);
  }
  printf("bye\n");
  fork();
  printf("ok\n");
}
```

Answer: hi bye ok ok ✔

The correct answer is: hi bye ok ok

Question **2**

Partially correct

Mark 1.25 out of 2.00

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

"..." means some code.

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

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

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

Release the lock held by some another process ✔

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

Tell compiler not to reorder memory access beyond this line ✔

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

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

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

Disable interrupts to avoid deadlocks ✔

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

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)

✔

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

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

✔

Your answer is partially correct.

You have correctly selected 5.

The correct answer is: 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, void
acquire(struct spinlock *lk)
{
...
__sync_synchronize();

→ Tell compiler not to reorder memory access beyond this line, void sleep(void *chan, struct spinlock *lk)
{
...
if(lk != &ptable.lock){
acquire(&ptable.lock);
release(lk);
} → Avoid a self-deadlock, void acquire(struct spinlock *lk)
{
pushcli();
→ Disable interrupts to avoid deadlocks, 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)
{
...
getcallerpcs(&lk, lk->pcs);

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

Question **3**

Correct

Mark 1.00 out of 1.00

Which one of the following is not a system call

- a. mount
- b. lseek
- c. open
- d. lock ✔

The correct answer is: lock

Question **4**

Partially correct

Mark 1.13 out of 1.50

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

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

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

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

P1,P2,P3,P4,P3,P5,P3,P6,P2

✖

The turn-around time for the process P2 is:

85

✔

The turn-around time for the process P6 is:

15

✔

The process P5 finishes at time unit:

50

✔

0-15 P1

15-20 P2

20-40 P3

40-45 P4

45-50 P5

50-55 P4

55-70 P6

70-80 P4

80-95 P2

--

P2 turnaround time = 95 - 15 = 80

Question **5**

Partially correct

Mark 0.33 out of 1.00

Select all the correct statements about Shell

- [x] a. Examples of shell are: bash, ksh, csh, zsh, sh, etc. ✔
- [x] b. Shell converts the application code into system calls. ✖
- [x] c. The essential job of the shell is to fork-exec the specified application ✔
- [ ] d. Examples of shell are: bash, ksh, csh, msh, ooosh, nosh, etc.
- [ ] e. The default shell for a user is specified in /etc/passwd on typical GNU/Linux systems.
- [ ] f. Shell is a layer on top of hardware

The correct answers are: The essential job of the shell is to fork-exec the specified application, Examples of shell are: bash, ksh, csh, zsh, sh, etc., The default shell for a user is specified in /etc/passwd on typical GNU/Linux systems.

Question **6**

Correct

Mark 1.00 out of 1.00

In an ext2 file system, if the block size is 4KB and partition size is 64 GB, then the number of block groups will be:

Answer: 512 ✔

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

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

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

The correct answer is: 512.00

Question **7**

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: 5.00 ✔

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 **8**

Correct

Mark 1.00 out of 1.00

Not a FOSS: Which of the following is/are not a FOSS operating system ?

- [ ] a. BSD Unix
- [ ] b. Darwin (core kernel of Mac-OS)
- [ ] c. GNU/Linux
- [ ] d. FreeBSD
- [ ] e. Minix
- [x] f. Windows ✔
- [ ] g. Open Solaris
- [ ] h. xv6

The correct answer is: Windows

Question **9**

Incorrect

Mark 0.00 out of 1.00

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

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

- a. No, because this breaks the atomicity requirement of compare-and-test. ✖
- b. Yes, because there is no race to update the lock variable
- c. Yes, because no matter in which order the if-check and compare-and-swap run in multiple processes, only one process will succeed in compare-and-swap() and others will keep looping in while-loop.
- d. No, because in the case of both processes succeeding in the "if" condition, both may end up acquiring the lock.

Your answer is incorrect.

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

Question **10**

Correct

Mark 1.00 out of 1.00

The following diagram that explains the concept of multi-threading

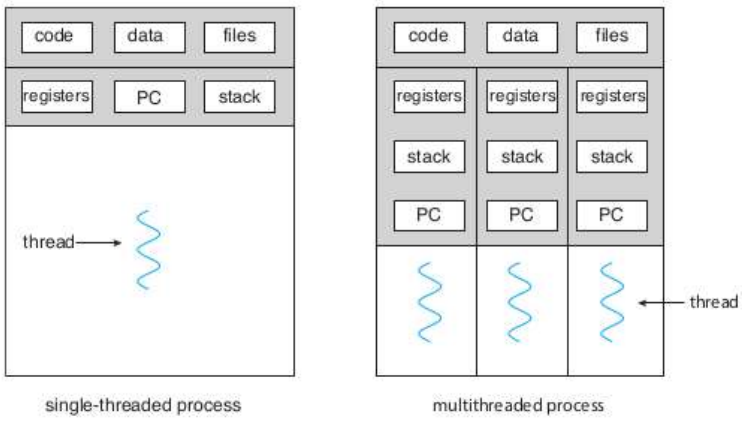


Figure 4.1 Single-threaded and multithreaded processes.

Leads to the following conclusions about implementation of threads

True	False		
○ ✗	◉ ✓	A multi-threaded program can be split in multiple ELF files.	✔
◉ ✓	○ ✗	The memory management for the process in the kernel should not (most typically) change due to creation of a thread.	✔
◉ ✓	○ ✗	Each thread should be represented by a function that starts execution on a separate stack.	✔
◉ ✓	○ ✗	Switching between threads requires a "context switch" with change of registers involved in it	✔

A multi-threaded program can be split in multiple ELF files.: False
The memory management for the process in the kernel should not (most typically) change due to creation of a thread.: True
Each thread should be represented by a function that starts execution on a separate stack.: True
Switching between threads requires a "context switch" with change of registers involved in it: True

Question **11**

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

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 ✔

Your answer is correct.

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

→ 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

Question **12**

Partially correct

Mark 1.85 out of 3.00

Suppose you are required to implement the priority scheduling algorithm in xv6. Select all the options that correctly reflect the changes that are required to be done in code:

Needed/Optional	Not Needed		
◉☑	○✖	Add a system call (like nice()) that allows to set priority of a process	✔
○☑	◉✖	Set the priority of the new process to the default(using inheritance) during fork()	✖
○✖	◉☑	Change yield() to re-set the timer value to zero for the process	✔
◉☑	○✖	Insert code in scheduler() after if(p->state != RUNNABLE); continue; by a code that selects the highest priority process.	✔
◉☑	○✖	Add a priority field to the struct proc	✔
○☑	◉✖	Optionally remove the default timer value of 10000000 in lapicinit()	✖
◉✖	○☑	Change scheduler() to calculate the priority of the process using user specified value	✖
◉✖	○☑	Change exec() to add a priority to the process.	✖
○☑	◉✖	Set a new value for the timer before you call swtch() in scheduler()	✖
○✖	◉☑	Change swtch() to set the timer value for the process	✔
◉☑	○✖	The nice() system call needs to acquire the ptable.lock for setting the priority.	✔
○✖	◉☑	Set the priority of the process as per information in ELF file	✔
○✖	◉☑	Change sched() to set the priority of the process.	✔

Add a system call (like nice()) that allows to set priority of a process: Needed/Optional
Set the priority of the new process to the default(using inheritance) during fork(): Needed/Optional
Change yield() to re-set the timer value to zero for the process: Not Needed
Insert code in scheduler() after if(p->state != RUNNABLE); continue; by a code that selects the highest priority process.: Needed/Optional

Add a priority field to the struct proc: Needed/Optional
Optionally remove the default timer value of 10000000 in lapicinit(): Needed/Optional
Change scheduler() to calculate the priority of the process using user specified value: Not Needed
Change exec() to add a priority to the process.: Not Needed
Set a new value for the timer before you call swtch() in scheduler(): Needed/Optional
Change swtch() to set the timer value for the process: Not Needed
The nice() system call needs to acquire the ptable.lock for setting the priority.: Needed/Optional
Set the priority of the process as per information in ELF file: Not Needed
Change sched() to set the priority of the process.: Not Needed

Question **13**

Partially correct

Mark 0.75 out of 1.00

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

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

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

Question **14**

Correct

Mark 1.00 out of 1.00

Map each signal with it's meaning

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

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

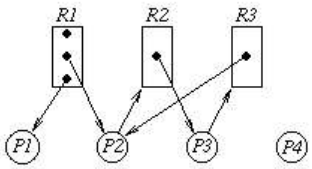
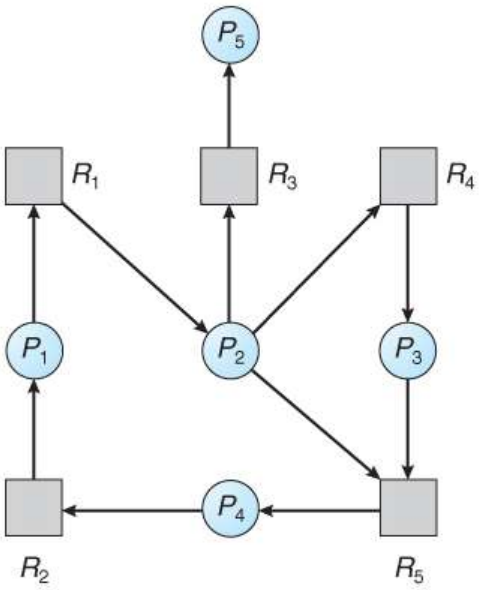
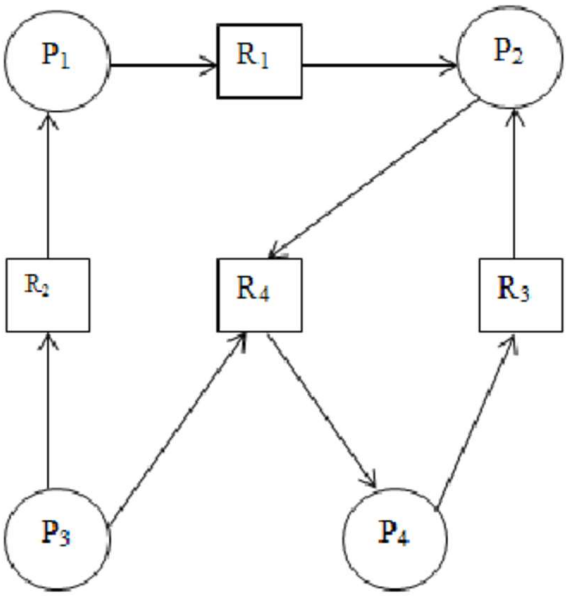
Question **15**

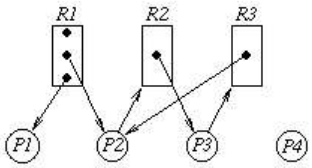
Correct

Mark 1.00 out of 1.00

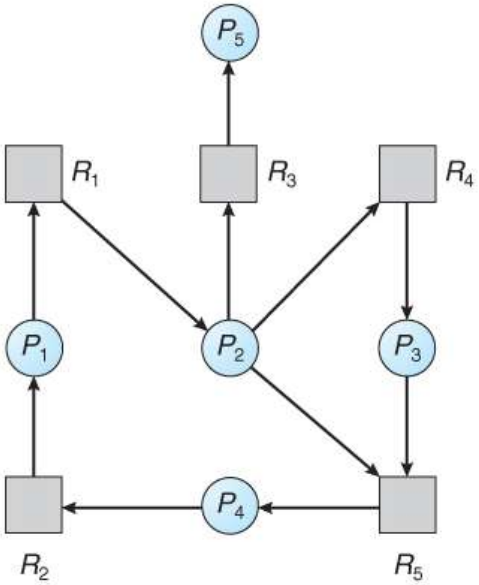
For each of the resource allocation diagram shown,

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

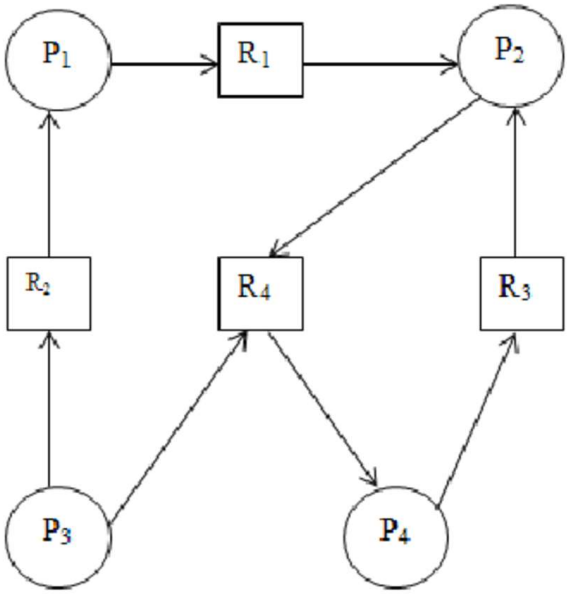
Yes	No		
◉	○		✔
◉	○		✔
◉	○		✔


R1
R2
R3
P1
P2
P3
P4

: Yes


P5
R1
R3
R4
P1
P2
P3
P4
R2
R5

: Yes


P1
R1
P2
R2
R4
R3
P3
P4

: Yes

Question **16**

Partially correct

Mark 0.67 out of 1.00

Match the pair

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

Question **17**

Correct

Mark 0.50 out of 0.50

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

Write the answer as a number.

The correct answer is: 6

Question **18**

Partially correct

Mark 0.40 out of 0.50

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

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

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

Question **19**

Partially correct

Mark 0.67 out of 1.00

Which of the following instructions should be privileged?

- [ ] a. Issue a trap instruction.
- [x] b. Access I/O device. ✔
- [x] c. Turn off interrupts. ✔
- [x] d. Set value of timer. ✔
- [x] e. Modify entries in device-status table. ✔
- [ ] f. Switch from user to kernel mode.
- [ ] g. Read the clock.

The correct answers are: Set value of timer., Access I/O device., Issue a trap instruction., Switch from user to kernel mode., Modify entries in device-status table., Turn off interrupts.

Question **20**

Correct

Mark 1.00 out of 1.00

Write the possible contents of the file /tmp/xyz after this program.

In the answer if you want to mention any non-text character, then write \0 For example abc\0\0 means abc followed by any two non-text characters

```
int main(int argc, char *argv[]) {
   int fd1, fd2, n, i;
   char buf[128];

   fd1 = open("/tmp/xyz", O_WRONLY | O_CREAT, S_IRUSR|S_IWUSR);
   write(fd1, "hello", 5);
   fd2 = open("/tmp/xyz", O_WRONLY, S_IRUSR|S_IWUSR);
   write(fd2, "bye", 3);
   close(fd1);
   close(fd2);
   return 0;
}
```

Answer: byelo ✔

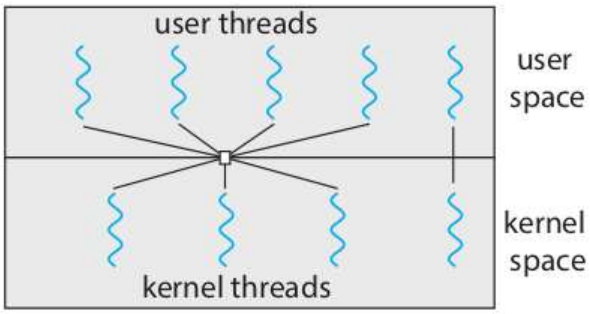
The correct answer is: byelo

Question **21**

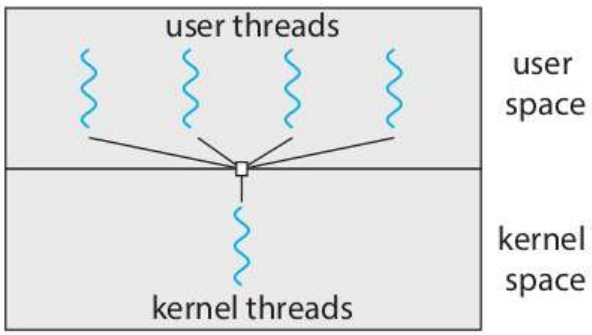
Partially correct

Mark 0.75 out of 1.00

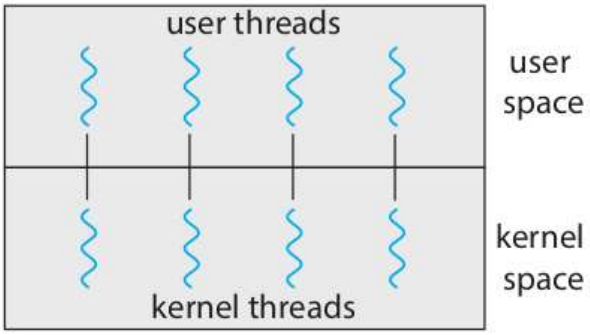
Match the diagram with the threading model



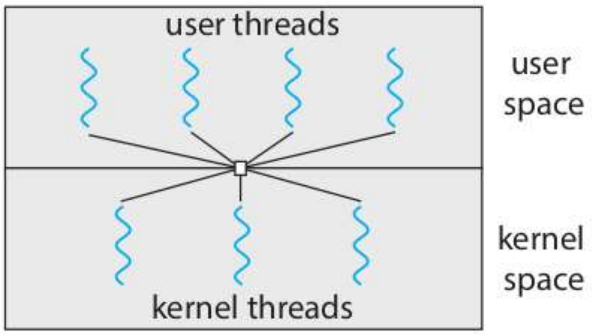
Many-Many ✖



Many-One ✔



One-One ✔

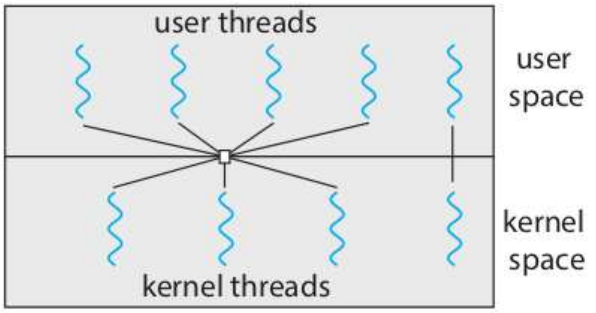


Many-Many ✔

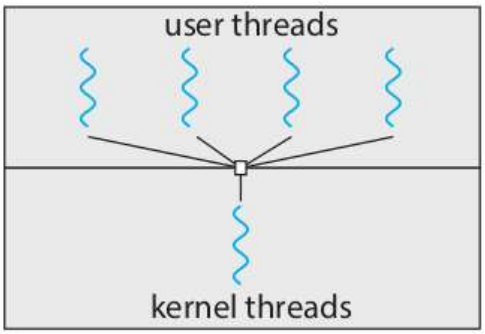
Your answer is partially correct.

You have correctly selected 3.

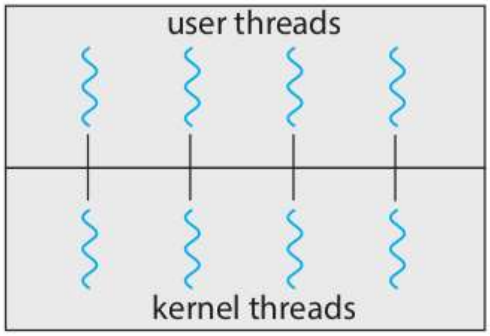
The correct answer is:



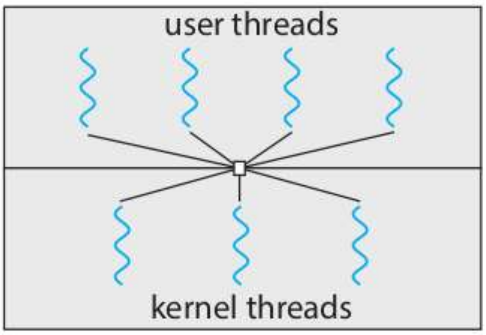
→ Two-Level,



→ Many-One,



→ One-One,



→ Many-Many

Question **22**

Correct

Mark 1.00 out of 1.00

Which one of the following is not a scheduling algorithm

- a. Priority
- b. paging ✔
- c. Multilevel Feedback Queue
- d. FCFS

The correct answer is: paging

Question **23**

Partially correct

Mark 1.86 out of 2.00

Given below is an incomplete code for reader-writer lock with preference for readers. Fill in the blanks to complete the code.

Note1:

In your answer, if an expression is to be written, then please separate all tokens by exactly one space in between. For example

`i = i + 1` is correct while

`i=i+1` is not correct or

`i= i +1` is also not correct.

Note2:

Correct: a->b++ (no spaces here!)

Any other notation is incorrect.

Note3: *The code of downgrade() and upgrade() has proportional to 4/7 marks.*

Code of rwlock:

```
typedef struct rwlock {
   int nActive;
   int nPendingReads;
   int nPendingWrites;
   spinlock_t
```

sl

✔ ;

```
   condition canRead;
   condition canWrite;
}rwlock;

void lockShared(rwlock *r) {
   spin_lock(&r->sl);
```

r->nPendingReads++

✔ ;

```
   while(r->nActive < 0)
      wait(
```

&r->canRead

✔ , &r->sl);

r->nActive++;

✘ ;

```
   r->nPendingReads--;
   spin_unlock(&r->sl);
}
void unlockShared(rwlock *r) {
   spin_lock(&r->sl);
```

r->nActive--

✔ ;

```
   if(r->nActive == 0) {
      spin_unlock(&r->sl);
      do_signal(
```

```
&r->canWrite
✔ );
  } else
      spin_unlock(&r->sl);
}
void lockExclusive(rwlock *r) {
  spin_lock(&r->sl);
  r->nPendingWrites++;
  while(r->nActive || r->nPendingReads)
      wait(
&r->canWrite,
✖ ,
&r->sl
✔ );
  r->nPendingWrites--;

r->nActive = -1
✔ ;
  spin_unlock(&r->sl);
}
void unlockExclusive(rwlock *r) {
  boolean_t wakeReaders;
  int i;
  spin_lock(&r->sl);

r->nActive = 0
✔ ;
  if(r->nPendingReads != 0) {
      for(i = 0; i <
r->nPendingReads
✔ ; i++)
          do_signal(&r->canRead);
  }
  else
      do_signal(
&r->canWrite
✔ );
  spin_unlock(&r->sl);
}
void downgrade(rwlock *r) {
  boolean_t wakeReaders;
  int i;
  spin_lock(&r->sl);
  r->nActive =
1
✔ ;
  if(r->nPendingReads != 0)
      for(i = 0; i <
r->nPendingReads
✔ ; i++)
          do_signal(
```

```
&r->canRead
);
    spin_unlock(&r->sl);
}
void upgrade(rwlock *r) {
    spin_lock(&r->sl);
    if(r->nActive == 1) {
        r->nActive =
-1
;
    } else {
        r->nPendingWrites++;
        r->nActive--;
        while(r->nActive !=
0
)
            wait(
&r->canWrite
, &r->sl);

r->nPendingWrites--
;
        r->nActive =
-1
;
    }
    spin_unlock(&r->sl);
}
```

Question **24**

Partially correct

Mark 0.17 out of 1.00

Select the correct statements about hard and soft links

Select one or more:

- [x] a. Deleting a soft link deletes both the link and the actual file ✗
- [ ] b. Deleting a hard link always deletes the file
- [ ] c. Soft link shares the inode of actual file
- [ ] d. Hard links can span across partitions while soft links can't
- [ ] e. Deleting a soft link deletes the link, not the actual file
- [x] f. Hard links share the inode ✓
- [x] g. Soft links increase the link count of the actual file inode ✗
- [x] h. Deleting a hard link deletes the file, only if link count was 1 ✓
- [x] i. Deleting a soft link deletes only the actual file ✗
- [x] j. Hard links enforce separation of filename from it's metadata in on-disk data structures. ✓
- [ ] k. Hard links increase the link count of the actual file inode
- [x] l. Soft links can span across partitions while hard links can't ✓

Your answer is partially correct.

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

Question **25**

Correct

Mark 1.00 out of 1.00

which of the following scheduling algorithms discriminate either in favor of or against short processes, from the perspective of minimizing waiting time.

For	Against		
✗	✓	FCFS	✓
✓	✗	Multilevel feedback queues	✓
✓	✗	Round Robin	✓

FCFS: Against
Multilevel feedback queues: For
Round Robin: For

Question **26**

Correct

Mark 1.00 out of 1.00

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

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

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

Question **27**

Partially correct

Mark 0.50 out of 1.00

Select which of the following data structures may need an update, in ext2 file system, when 5 bytes get removed from the end of an existing file

Yes	No		
☐ ☑	◉ ✖	Inode of the file	✖
☐ ✖	◉ ☑	The inode of the parent directory	✔
☐ ✖	◉ ☑	The inode bitmap	✔
◉ ✖	☐ ☑	A free block on the file-system	✖
◉ ☑	☐ ✖	One of the block bitmaps	✔
☐ ✖	◉ ☑	The boot sector	✔
☐ ☑	◉ ✖	The superblock	✖
☐ ☑	◉ ✖	The group descriptor	✖

Inode of the file: Yes
The inode of the parent directory: No
The inode bitmap: No
A free block on the file-system: No
One of the block bitmaps: Yes
The boot sector: No
The superblock: Yes
The group descriptor: Yes

Question **28**

Partially correct

Mark 0.67 out of 1.00

Consider this program.
Some statements are identified using the // comment at the end.
Assume that = is an atomic operation.

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

Which statements are part of the critical Section?

Yes	No		
◉✖	○☑	D	✖
◉☑	○✖	C	✔
○✖	◉☑	F	✔
◉☑	○✖	A	✔
◉✖	○☑	B	✖
○✖	◉☑	E	✔

As per Remzi's book: "A critical section is a piece of code that accesses a shared variable (or more generally, a shared resource and must not be concurrently executed by more than one thread."

As per Galvin's Book: a critical section, in which the process may be accessing — and updating — data that is shared with at least one other process. The important feature of the system is that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Since A and C refer to a variable that is shared, it's critical section. Here the hardware guarantees (as = is atomic) that the critical section is accessed by only one thread at a time.

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

Question **29**

Partially correct

Mark 0.25 out of 0.50

Select all the correct statements related to implementation of a Hypervisor like VirtualBox

- [ ] a. The HOST OS determines whether a process (in Guest) or the guest OS itself was doing privileged instruction depending on the privilege level.
- [ ] b. When an application runs a privileged instruction it traps into the actual hardware
- [x] c. Typically they run using CPU privilege level 1 or 2 (lower than kernel's but higher than applications) ✔
- [x] d. All Traps in hardware are handled by HOST OS, but Host OS may hand it over to Guest OS if trap was from within guest OS ✔

The correct answers are: Typically they run using CPU privilege level 1 or 2 (lower than kernel's but higher than applications), When an application runs a privileged instruction it traps into the actual hardware, All Traps in hardware are handled by HOST OS, but Host OS may hand it over to Guest OS if trap was from within guest OS, The HOST OS determines whether a process (in Guest) or the guest OS itself was doing privileged instruction depending on the privilege level.

Question **30**

Partially correct

Mark 0.75 out of 1.00

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

- [ ] a. The file system specific function pointers, for file system system-calls, need to be setup in the generic inode during lookup.
- [x] b. 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. ✔
- [x] 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() ✔
- [x] d. 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()) ✔
- [ ] e. 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.
- [x] f. The lookup() operation needs to check if it's crossing a mount point and call FS specific operations to read inodes/directories ✔
- [x] g. A mount() system call should be provided to mount a partition onto some directory in existing namespace rooted at "/" ✔
- [x] 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 **31**

Partially correct

Mark 0.29 out of 1.00

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

True	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.	✖
◉ ✖	☑	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.	✔
☑	◉ ✖	The kernel ensures that the MMU is setup before scheduling a process and then the CPU/MMU ensures that the address translation takes place.	✔
☐	◉ ✖	When a process is executing, each virtual address is converted into physical address by the CPU hardware directly.	✖
◉ ✖	☐	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.	✖
◉ ✖	☐	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.	✖

The compiler generates address references for code/data/stack/heap in the executable file, depending on the MM architecture provided by CPU and kernel.: True
The compiler 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 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 CPU hardware directly.: True
When a process is executing, each virtual address is converted into physical address by the kernel directly.: False
The kernel refers to the page table for converting each virtual address to physical address.: 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

Question **32**

Correct

Mark 1.00 out of 1.00

Consider a computer system with a 32-bit logical address and 8- KB page size. The system supports up to 1 GB of physical memory. How many entries are there in each of the following?

a. A conventional, single-level page table (write a decimal number):

524288

✔

b. An inverted page table (number of entries only, write a decimal number):

131072

✔

Question **33**

Correct

Mark 1.00 out of 1.00

Mark the statements as True/False with respect to Mobile systems and swapping.

True	False		
◉	○	Poor throughput between main memory and flash memory is one reason for restriction on use of swap on mobile devices.	✔
◉	○	Size of flash memory is one reason for restriction on use of swap on mobile devices.	✔
◉	○	Mobile systems generally do not support swapping	✔
◉	○	Limited number of write operations that flash memory can tolerate, is one reason for limitations of swapping on mobile devices.	✔

Poor throughput between main memory and flash memory is one reason for restriction on use of swap on mobile devices.: True
Size of flash memory is one reason for restriction on use of swap on mobile devices.: True
Mobile systems generally do not support swapping: True
Limited number of write operations that flash memory can tolerate, is one reason for limitations of swapping on mobile devices.: True

Question **34**

Partially correct

Mark 1.38 out of 3.00

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

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

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

Must	Not-Must		
✖	☑	Add a EUID field in the struct proc representing Effective user ID	✖

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

Question **35**

Partially correct

Mark 0.86 out of 1.00

Mark statements as T/F

All statements are in the context of preventing deadlocks.

True	False		
◉ ☑	○ ✖	Mutual exclusion is a necessary condition for deadlock because it brings in locks on which deadlock happens	✔
◉ ☑	○ ✖	Circular wait is avoided by enforcing a lock ordering	✔
◉ ☑	○ ✖	The lock ordering to be followed to avoid circular wait is a protocol to be followed by programmers.	✔
◉ ☑	○ ✖	If a resource allocation graph contains a cycle then there is a possibility of a deadlock	✔
◉ ✖	○ ☑	If a resource allocation graph contains a cycle then there is a guarantee of a deadlock	✖
○ ✖	◉ ☑	The lock ordering to be followed to avoid circular wait is a code in OS that checks for compliance with decided order	✔
◉ ☑	○ ✖	Deadlock is not possible if any of these conditions is not met: Mutual exclusion, hold and wait, no pre-emption, circular wait.	✔

Mutual exclusion is a necessary condition for deadlock because it brings in locks on which deadlock happens: True
Circular wait is avoided by enforcing a lock ordering: True
The lock ordering to be followed to avoid circular wait is a protocol to be followed by programmers.: True
If a resource allocation graph contains a cycle then there is a possibility of a deadlock: True
If a resource allocation graph contains a cycle then there is a guarantee of a deadlock: False
The lock ordering to be followed to avoid circular wait is a code in OS that checks for compliance with decided order: False
Deadlock is not possible if any of these conditions is not met: Mutual exclusion, hold and wait, no pre-emption, circular wait.: True

◄ Course Exit Feedback

Jump to...