When a user process calls a function in user space, and that function makes no system calls,
A) The User-Kernel protection boundary is not crossed.
B) The User-Kernel protection boundary is crossed sometimes and not crossed other times.
C) The User-Kernel protection boundary is only crossed when there is a page fault or error.
D) The User-Kernel protection boundary is always crossed.

When a user process invokes a system call,
A) The User-Kernel protection boundary can never be crossed.
B) The User-Kernel protection boundary is crossed sometimes and not crossed other times.
C) The User-Kernel protection boundary is only crossed when there is a page fault or error.
D) The User-Kernel protection boundary is always crossed.

Suppose a running job (or task or thread, different words for what the scheduler schedules) makes a blocking system call, say to wait for a person to type something. From which jobs does the scheduler pick a job?
(A) Blocked (or waiting)
(B) Running
(C) Ready
(D) Paged In
(E) Zombie

When a user space function is called, what determines the address of the code that implements the function?
A) An operand of a call instruction in user space, like \texttt{fun} in \texttt{call fun}
B) An operand of a call instruction in kernel space, like ... but in kernel space.
C) User space in a table entry indexed by a number, set up by code like \texttt{table[351] = fun}
D) Kernel space in a table entry indexed by a number, set up by code like \texttt{table[351] = fun}

When a system call service routine (like \texttt{sys_csi500(...)}) is called, what determines the address of the code that implements the function?
A) An operand of a call instruction in user space, like \texttt{fun} in \texttt{call sys_csi500}
B) An operand of a call instruction in kernel space, ... but in kernel space.
C) User space in a table entry indexed by a number, set up by code like \texttt{table[351] = sys_csi500}
D) Kernel space in a table entry indexed by a number, set up by code like \texttt{table[351] = sys_csi500}
Suppose the linker creates an executable file (like hello in Lab01). `objdump --disassemble` reports that files contents. The addresses at which the machine instructions will be located are reported. (You saw these addresses to the left of the hex and disassembled instructions.) Assume the program is run in a user space Linux process. What are those addresses?

A) Offsets in the .text section  
B) Physical addresses  
C) Virtual addresses  
D) Page addresses  
E) Segment addresses

Some easy fork-exec questions: For each scenario, assume the system call is successful. Which returns twice? Hint: One of these does not return at all! (A) fork (B) exec. (C) neither

Which replaces the program? (A) fork (B) exec. (C) neither

Which returns exactly once? Hint: One of these does not return at all! (A) fork (B) exec (C) neither

Which makes a new process? (A) fork (B) exec (C) neither

In multilevel feedback queue scheduling, observations by the scheduler (an OS kernel component) of each job's history of CPU usage might be used to raise and/or lower the priority of that job. In other words, recorded information about the past might be used to make wiser decisions about what to do which affects the future.

(A) When more CPU time is used, the priority is lowered.  
(B) When more CPU time is used, the priority is raised.

Suppose you take breakfast for 15 minutes each of the 6 days of the week except Sunday, and on Sunday your take breakfast for a relaxing 1 hour (60 minutes). What is your average time for breakfast?

(A) 15 minutes  
(B) \((15 \times 6 + 60)/7 = 150/7 = 21 \frac{3}{7} \) minutes  
(C) \((15+60)/2 = 75/2 = 37.5 \) minutes  
(C) 60 minutes

The code below is written in assembly language. The assembler makes a label like LabelA into a symbol and puts it into the symbol table. The symbol table stores what each symbol means. What a symbol means is what it symbolizes. What does a symbol like LabelA symbolize?

A) The machine instruction that the symbol labels.  
B) A memory address.  
C) The contents of memory at some address

Unusual code which finds the address of the next instruction by pushing it and popping it right away:

```assembly
    call LabelA
    LabelA: popl %eax
    /*Now, the address of the popl instruction is in register eax*/
```

In this unusual code:

A) It is false that LabelA labels a function that should be called from anywhere.  
B) It is true that LabelA labels a function that should be called from anywhere.
A big data analysis program (written in C) makes lots of small allocations of memory using `malloc` and frees many of them by calls to `free`. These allocations are freed in a random looking pattern. Then, the program tries to use `malloc` to allocate a big buffer for writing the results. This call to `malloc` fails (returning 0) because there is no big enough contiguous region of unused space available, even though the total number of bytes in the space freed by the previous `free` calls exceeds the requested big buffer size. This cause of this failure is classified as:

A) Internal fragmentation.
B) External fragmentation.
C) File fragmentation

The `vm_area_structs` (generically called segment descriptors) of Linux are organized into Red-Black balanced binary search tree data structure. What is that data structure used for by the virtual memory system when there is a page fault?

A) To quickly find which virtual memory page frame contains the faulting virtual address.
B) Implement paging out: To quickly find a page to evict when the supply of free physical page frames becomes small.
C) To quickly find out which segment, if any, contains the faulting virtual address.
D) To quickly find the physical page frame number from the virtual page frame number.

What word(s) mean(s) that an OS spends too much time using the disk to read and write page contents because there is not enough physical memory?

A) Starvation
B) Thrashing
C) Priority Inversion
D) Deadlock

Which scheduling strategy or algorithm does NOT use a quantum or time slice value to determine how long a job is allowed to run before the scheduler is called again?

A) Multilevel Feedback Queue (MLFQ)
B) MLFQ with periodic priority increases
C) First Come First Serve
D) Round Robin
From the last lab: Familiar csi500 syscall code you have added into the kernel:

```c
#include <linux/syscalls.h>
#include <linux/printk.h>
SYSCALL_DEFINE2(csi500, long unsigned __user *, pmajreport, /*PVars*/
                long unsigned __user *, pminreport)
{
    printk(KERN_EMERG "Hello from the csi500 system call written by ... \n");
    printk(KERN_EMERG "Param values: %p %p\n", pmajreport, pminreport);
    return 0;
}
```

Familiar user level code to call your csi500 syscall (an excerpt):
```
int main(int argc, char *argv[])
{
    unsigned long majpfaults, minpfaults;
    majpfaults = 99999999; /*Phony values..will be overwritten*/
    minpfaults = 99999999; /*Phony values*/
    /*Vals*/    int retval = syscall(351, &majpfaults, &minpfaults);
    /* Error return check omitted */
    printf("My syscall's parameters: %p %p\n",
           &majpfaults, &minpfaults);
    printf("Current values: majpfaults=%lu minpfaults=%lu\n",
            majpfaults, minpfaults);
}
```

The two variables named `majpfaults` and `minpfaults`

A) are both located in user virtual memory space.
B) are both located in kernel virtual memory space.

Note: The main problem of this question is about the C language.

The two parameter values to the system call are specified by the code marked /*Vals*/ The system call service routine defined by above kernel code receives these two parameter values in two parameter variables near the mark /*PVars*/ So far, the system call only prints the two parameter values into the system log.

What best describes those two parameter values? (They are data. What is that data's significance?)

A) Each value is a value read from a variable in memory.
B) Each value is the address of a variable.
The rest of the problems require some lines of calculations or sketching of simulations.

Here is the famous page reference string that exhibits Belady's anomaly. The virtual page numbers are A, B, C, D and E so marking your answers is easy (think of them as hex digits).

A B C D A B E A B C D E

For all three questions below, ASSUME the physical memory size in 4 page frames. Your job is to demonstrate each of the three page replacement algorithms that were covered in the course.

### FIFO (First In First Out)

How many times are pages paged out (if the same page is paged out say twice, count it twice of course)?

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5 or more.

Which is the first page paged out?

- A
- B
- C
- D
- E

### LRU (Least recently used)

How many times are pages paged out (if the same page is paged out say twice, count it twice of course)?

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5 or more.

Which is the first page paged out?

- A
- B
- C
- D
- E

### Optimal

How many times are pages paged out (if the same page is paged out say twice, count it twice of course)?

- A) 1
- B) 2
- C) 3
- D) 4
- E) 5 or more.

Which is the first page paged out?

- A
- B
- C
- D
- E
Suppose the hit time in some two level cache system is 1 unit of time, and the miss time is 1000 units of time.

(A) The lower (smaller) cache level is 1000 times faster than the higher (larger) cache level.
(B) The higher (larger) cache level is 1000 times faster than the lower (smaller) cache level.

Consider the above two level cache system. Approximately how small must the miss % (that is, miss ratio) be so that the on the average, the cached system is at least half as fast as the fast, higher level cache?

Hints: Half as fast means the time is twice as long as the time for a hit.
Do some math after applying the chief cache performance formula:
Average Memory Access Time = (Hit %)*(Hit time) + (Miss %)*(Miss time)

(A) 10% or 0.1 or 1 in 10
(B) 1% or 0.01 or 1 in 100
(C) 0.1 % or 0.001 or 1 in 1000
(D) 0.01 % or 0.0001 or 1 in 10,000
(E) 0.001% or 0.00001 or 1 in 100,000