Question 1. What is the purpose of copyin() and copyout()?
copyin() copies a block of memory in user space to kernel space.
copyout() does the opposite, copying a block of memory in kernel space to user space.

Question 2. What is the difference between copyin() and copyinstr()?
copyinstr() is used only for copying strings from user space to kernel space, whereas copyin() is used for a block of memory containing any kind of data. The difference in what these functions do is that copyinstr() will stop copying memory once it finds a zeroed byte (null character), but copyin() will copy everything within the given range.

Question 3. In runprogram(), why is it important to call vfs close() before going to user mode?
At this point, the program has been loaded into memory as executable code, and the file (which is opened as read-only) is no longer needed, so we need to close it now or it will never be closed.

Question 4. Which kernel function is used to make a thread switch to executing user-level code?
md_usermode()

Question 5. Which types of MIPS exceptions are handled by the kill_curthread() function? Your answer should be a list of MIPS symbolic exception codes, e.g., EX_OVF.

EX_IRQ	(Interrupt)	EX_DBE	(Bus error on data load or store)
EX_MOD	(TLB write to read-only page)	EX_SYS	(Syscall)
EX_TLBL	(TLB miss on load)	EX_BP	(Breakpoint)
EX_TLBS	(TLB miss on store)	EX_RI	(Illegal instruction)
EX_ADEL	(Address error on load)	EX_CPU	(Coprocessor unusable)
EX_ADES	(Address error on store)	EX_OVF	(Arithmetic overflow)
EX_IBE	(Bus error on instruction fetch)		

Question 6. Why do you "probably want to change" the implementation of kill_curthread()?
The implementation should be changed to kill the current thread and free the memory it used, then switch to another thread instead of crashing the operating system.

Question 7. What is the system call number for a reboot? Is this value available to userspace programs? Why or why not?
The system call number for reboot is 8. Yes, this system call is available to user-space programs since you want the user to be able to reboot their operating system using a user-space program such as the indicator applet on the panel (or the start menu on Windows).

Question 8. OS/161 has two separate implementations of malloc, one for use by the kernel, and one for use by application programs. The kernel's malloc (kmalloc) is in kern/lib/kheap.c. The application malloc is missing, but it belongs in lib/libc/malloc.c. Explain why there are two separate implementations, and describe the implications (for applications) of the missing implementation.
There are two separate implementations, because the kernel has it's own memory space that only the kernel can use. This prevents applications from using up the kernel's memory, so if an application has a memory leak, it will not crash the operating system. The implication of the missing implementation of malloc means that user applications are unable to allocate memory on the heap.

Question 9. The kernel uses vnodes to represent open files or devices. Which kernel function is used to open a file or device and obtain a vnode?
vfs_open()

Question 10. What operations can you do on a vnode? If two different processes open the same file, do we need to create two vnodes?
The following operations can be performed on a vnode:
VOP_INIT, VOP_OPEN, VOP_CLOSE, VOP_RECLAIM, VOP_READ, VOP_READLINK, VOP_GETDIRENTRY, VOP_WRITE, VOP_IOCTL, VOP_STAT, VOP_GETTYPE, VOP_TRYSEEK, VOP_FSYNC, VOP_MMAP, VOP_TRUNCATE, VOP_NAMEFILE, VOP_CREATE, VOP_SYMLINK, VOP_MKDIR, VOP_LINK, VOP_REMOVE, VOP_RMDIR, VOP_RENAME, VOP_LOOKUP, VOP_LOOKPARENT
If two different processes open the same file, we do not need to create two vnodes. This is what the opencount variable is used for,