**P. 34**

**Q4**. suppose a computer system and all of its applications were completely bug free. Suppose further that everyone in the world were completely honest and trustworthy. In other words, we need not consider fault isolation.

1. How should an operating system allocate time on the processor? Should it give the entire processor to each application until it no longer needs it? If there were multiple tasks ready to go at the same time, should it schedule first the task with the least amount of work to do or the one with the most? Justify your answer.

The operating system shouldnt allocate the whole processor to one program. This goes against the multiprogramming design on p.9. The processor should be able to switch between the program and the OS kernel regularly. If there are multiple tasks ready to go at the same time the processor should break them up and handle them at the same time in smaller chunks spread across the processors. The cpu should break up each of the tasks and put them in a queue so that while the cpu is working on a small task it's also loading the next task to be done.

1. How should the operating system allocate physical memory to applications? What should happen if the set of applications not fit in memory at the same time?

The processor should allocate most of the memory to the active application and OS. the other applications should pull more resources if they are activated. The cpu will stop other not used applications to provide focus to the active and allow it to process through quicker, this is talked about on page 8 and 9 when talking about the referee and illusionist.

1. How should the operating system allocate its disk space? Should the first user to ask acquire all of the free space? What would the likely outcome be for that policy?

No, the pc should be greedy with its resources. The pc should only allow the user to use a certain amount of space and allow other users the same respect. If one user is using some space the other users should see that the space is taken but still have access to the same memory pool. If the policy was that one user could request all of it and lock all the memory then there wouldn't ever be any space. It should be divided evenly, regardless of first come.

**Q9**. How would you design a system to update complex data structures on disk in a consistent fashion despite machine crashes?

By placing the structure in cache memory and then updating the data and writing to memory. This would ensure that the data is not corrupt and allow the best chance of writing the data correctly and giving persistence.

**P.90**

**Q1**. When a user process is interrupted or causes a processor exception, the x86 hardware switches the stack pointer to a kernel stack, before saving the current process state. Explain Why?

In x86 seperate stacks used for kernel and user processes due to security purpose. The kernel delegates the interrupt dispatch to a common kernel interrupt entry/exit mechanism which saves pre-interrupt register state before calling the registered handlers.

When dispatching an interrupt execute a privilege and/or stack switch interrupts that occur while executing user process will switch to use the kernel stack in place at that point, without saving current state.

**Q7**. Most hardware architectures provide an instruction to return from an interrupt, such as uret. This instruction switches the mode of operation from kernel-mode to user-mode.

1. Explain where in the operating system this instruction would be used.
2. Explain what happens if an application program executes this instruction.
3. It is used when the software or hardware needs immediate attention. This will give feedback to the cpu to let it know how immediately the issue needs to be dealt with. There is also the callback which happens when the interruptor command is issued and will store overflow interrupts in a stack.

The callback routine sorts the interrupts and prioritizes them.

1. The cpu checks to see if they are allowed to execute the command. This helps keep the system secure. Then the cpu restores all the affected registers to the flags they contained prior to the interrupt the EIP registers. Then the EFLAGS register and the stack pointer ESP.