1. The xv6 kernel uses a non-cooperative approach to gain control while a user process is running. It operates by utilizing interrupts to initiate context switches between user space and kernel space. When an interrupt occurs, such as a timer or a system call, the kernel's interrupt handler is invoked. This saves the user process's current state, switches the CPU to kernel mode, and executes the required operations. After the interrupt is handled, the saved state of the user process is restored, and the CPU switches back to user mode, allowing the process to resume from where it was interrupted.
2. After fork() is called, the parent process runs before the child process in most cases due to several factors, including the operating system often giving a higher priority to the parent, and the parent is already running and continuing execution from the point immediately after fork(). The child process will run before the parent process if the parent's process time slice ends after it begins handling the fork() system call but before it resumes execution of the application code.
3. The code which saves the content of the old process and load the context of the new is in switch.S and is as follows:

```
.globl swtch
swtch:
movl 4(%esp), %eax
movl 8(%esp), %edx

 # Save old callee-save registers
pushl %ebp
pushl %ebx
pushl %esi
pushl %edi

# Switch stacks
movl %esp, (%eax)
movl %edx, %esp

# Load new callee-save registers
popl %edi
popl %esi
popl %ebx
popl %ebp
Ret
```

This code is reached from within **void scheduler(void)** in **proc.c** specifically from this line: **swtch(&cpu->scheduler, proc->context);**