Read chapter-1 from the xv6 book.

When the PC starts, it first loads the boot loader into a predefined location in the physical memory and jumps to it. The boot loader code loads the kernel text and data at physical address 0x10000. Interestingly, the xv6 kernel is compiled for the base address (0x80100000), and all the symbols addresses are greater than or equal to (0x80100000). The boot loader code jumps to the routine entry:1144 in the kernel. The entry code doesn't try to access any symbols (global variables and functions) until it loads a page table that maps the first 4 MB of the physical address space at virtual address 0 and 2 GB. Notice, that the entire xv6 kernel and data can fit into first 4 MB of physical address space.

After loading the page table, "entry" jumps to the "main" routine. The "main" routine initializes different subsystems of the kernel and devices. At some point main calls userinit:2502 to create the first user process. userinit calls allocproc:2455. allocproc:2455 finds an unused process control block (PCB) from a list of PCBs for the new process. PCB contains metadata for a process, e.g., size, page directory, kernel stack, process state, pid, trap frame, context, open files, etc. It then allocates kernel stack for the new process and allocates space for trap frame and the process context on the stack. After returning from allocproc, userinit creates a page directory and map kernel pages in that directory. "initcode.S" is the first user process. The entire text, data, and stack of "initcode.S" fits on a single page. The text section of "initcode.S" is loaded at virtual address 0 followed by the data section. The stack is set to the bottom of the virtual page. The "esp" and "eip" in the trap frame are set to 4096 and zero, such that after "iret" the usermode execution will start at virtual address zero. It also sets the segment registers in the trap frame appropriately.

After userinit, at some point scheduler is called. Scheduler looks for a runnable process in the list of PCBs. If it finds a runnable process, it loads the page directory of the target process in cr3, and call swtch:2958 to load the new process context. The swtch routine saves the context of the scheduler in cpu->scheduler and loads the "esp" with the context of the target process. It then pops the context and return. The stack was set up in such a way the "swtch" returns to "forkret". "forkret" releases the scheduler lock and return. On returning from "forkret", "trapret" (allocproc setup the kernel stack in this way) is called. "trapret" pops the trap frame and executes "iret", that resumes user mode execution.

How does scheduler regain control after calling swtch?

"yield" calls "sched:2758" that saves the current process context in PCB and loads the scheduler context from the "cpu->scheduler".