A compiler generates an executable after compiling a program. The executable is essentially a file which contains the CPU instructions, global variables, and their contents. The CPU instructions and global data are loaded into the RAM during the exec system call. The data and instructions are stored in different sections of the executable file, and an ELF format describes the structure of the executable file.

The top of the file contains elfhdr:1005. The first field of the elfhdr is “magic” which must be equal to 0x464C457F. This is just a sanity check that the file is actually in ELF format. Other fields are “phoff” (offset of first program header), and “phnum” (number of program headers). Each program header contains the metadata corresponding to a section. The metadata includes: the offset of the section in the executable file (where the content of the section is stored), the size of the section. The program header contains two types of size variables: filesz and memsz. memsz is the total size in memory, filesz is the number of bytes that need to be read from the file. memsz should always be greater and equal to the filesz. The loader must read filesz bytes of data in the memory. Rest of memory (memsz-filesz) bytes are filled with zeros. The memsz can be different from filesz when the compiler doesn’t need to save the contents of a section in the file. E.g., all uninitialized global variables can be moved to another section. The compiler does not need to save the contents of uninitialized variables, because doing so will unnecessary increases the size of the executable file.

Let us see how exec: 6310 system call works. Exec system call first opens the executable file and read the elf header. It verifies the magic bytes are equal to the expected value. Then exec calls setupkvm:1837 to map kernel pages. setupkvm first allocates space for page directory and then calls mappages:1779 to map I/O space, kernel text, kernel data, and rest of the RAM respectively at the virtual address starting at KERNBASE in the page directory. setupkvm uses a linear mapping of VA to PA for kernel pages.

After mapping the kernel pages, exec iterates all the program headers to identify the virtual address (vaddr) and the memory size (memsz) for each section. Then it calls allocuvm:1953 to allocate physical pages for virtual address range {vaddr, vaddr+memsz}. Finally, exec calls loaduvm to read the contents of the section from the executable file (of size filesz) in the memory buffer starting at vaddr. exec maps each section from the executable in the process memory.

After loading the sections, exec allocates two pages for stack in the process address space using allocuvm. It revokes the user permission from the first page, such that access to that memory page would trigger a page fault. This page acts as a guard page to detect stack overflow. The second page is used as the stack. The exec routine copies the arguments of main to the stack in such a way that the main routine of the executable would receive two parameters argc and argv on the user stack. xv6 requires main routine to explicitly do the exit system call and hence the exec routine puts a garbage return address on the stack (because main never returns).