Part A:

How system call works.

1. In the xv6 OS, we can run programs from the terminal by implementing user programs. In these user programs, we can use system calls as functions.
2. These system calls are provided in the “user.h” file.
3. We have assigned a number to each system call in the “syscall.h” file.
4. In the “usys.S” file, we read this number from “syscall.h” and store the number in the “eax” register.
5. Then the control is transferred from user mode to kernel mode.
6. In kernel mode, “syscall.c” file reads the value of the “eax” register and calls the appropriate system call from an array of functions in which pointers to system call functions are stored.
7. These system call functions are defined in “sysfile.c” and “sysproc.c” files.
8. Once the system call execution is complete, the return value is stored in the “eax” register and the control returns to user mode.
9. In user mode, the output is handled as per return value. If an error code is returned, it gets handled appropriately by the user program.

The user can pass arguments to the system call. In the system call, these arguments are read using argptr, argint, argstr functions defined in the “syscall.c” file.

There are many kernel functions that need to be accessed by multiple files. Therefore these functions are provided in the “defs.h” file and can be very useful while implementing system calls.

Part B:

Observations in Part 3:

1. Yes, the number of valid entries changes from 2 to 12. This is because for global variables, the memory is allocated at compile time in the data segment because multiple functions might want to access the global variables. Thus the address for global variables needs to be known at compile time. Thus memory is allocated at compile time.
2. No, the number of valid entries does not change. This is because memory for local variables is allocated on the stack at the first instance of the variable being accessed. Local variables are uninitialized and thus the page table does not have entries for storing local variables.
3. The number of entries does not change over multiple executions. This is to be expected, since we are not changing the program the amount of required memory should not change. But the virtual and physical addresses are changing. This is reasonable as it is up to the OS where it wants to allocate memory to the process.