Difference with Procedure Call

- The TRAP instruction also differs from the procedure call instruction in **two** fundamental ways:
  - First, it switches into kernel mode. The procedure call instruction does not change the mode
  - Second, rather than giving a relative or absolute address where the procedure is located, the TRAP instruction cannot jump to an arbitrary address. Depending on the architecture, it either jumps to a single fixed location or equivalent

Example of a SysCall

Example: count = **read**(fd, buffer, nbytes);

Explanation of the action: The system call returns the number of bytes actually read in *count*.

Question: why count!=nbytes all the time?

Ways of Passing Parameters

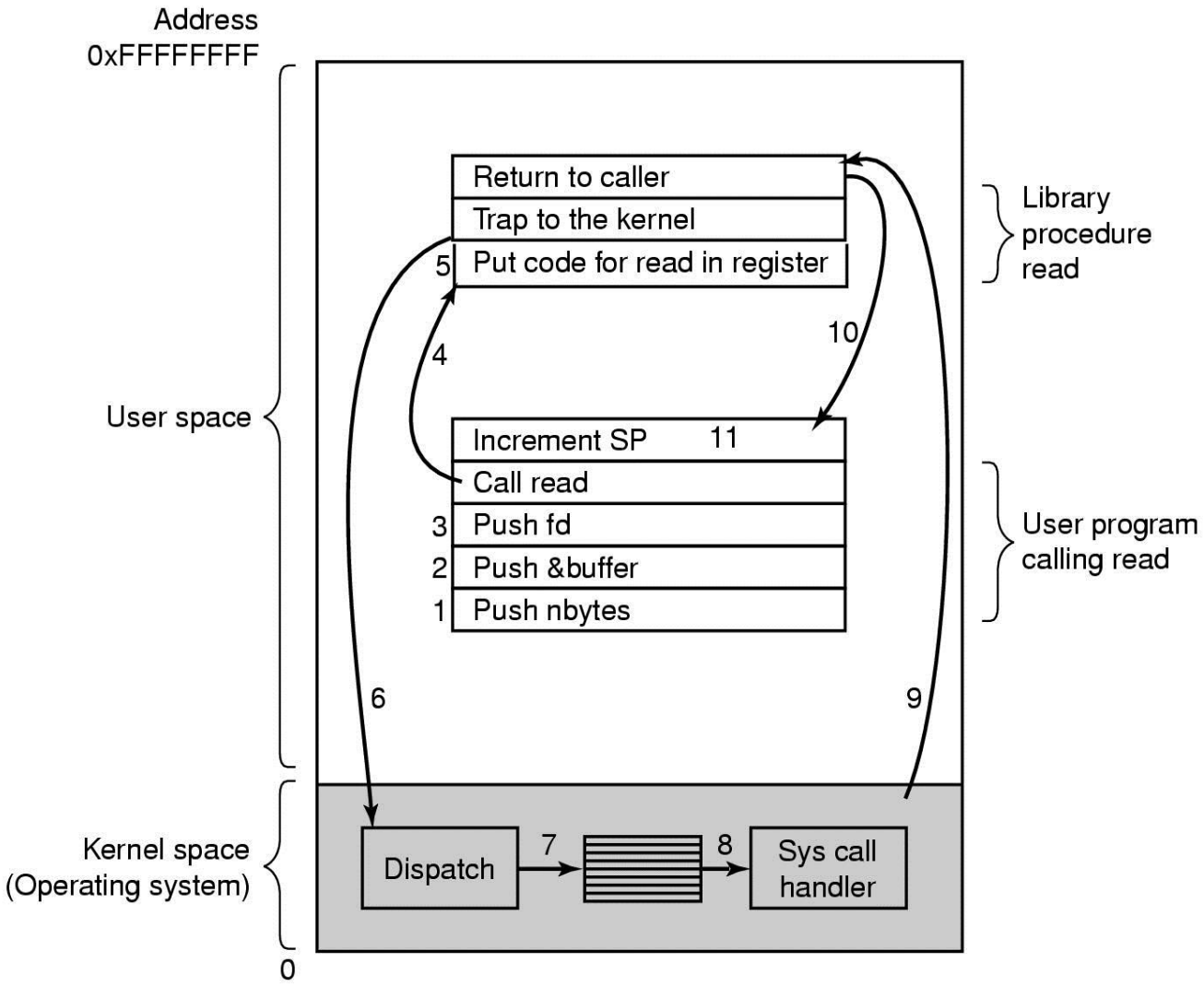
- **Three** general methods for passing parameters:

  Method#1: Pass parameters in *registers*.

  Methosd#2: Store the parameters in a table in memory, and the table address is passed as a parameter in a register.

  Method#3: *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

Steps of Making a SysCall



Steps of Making a SysCall

1. C and C++ compilers pass parameters between a running program and the operating system using parameter passing method#3. (Steps: 1-3)
2. Actual call to library procedure happens. (Step: 4)
3. The library procedure, possibly written in assembly language, passes the system call number using parameter passing method#1. (step 5)
4. A TRAP instruction is performed to switch from user mode to kernel mode. (step 6)

Steps of Making a SysCall

5. The kernel examines the system call number and then dispatches to the correct system call handler, using parameter passing method#2. (step 7)

6. System call handler runs. (step 8)

7. Control may be returned to the user-space library procedure at the instruction following the TRAP instruction. (step 9)

Steps of Making a SysCall

8. This procedure then returns to the user program in the usual way procedure calls return. (Step 10)

9. To finish the job, the user program has to clean up the stack, as it does after any procedure call. (Step 11)