System Calls

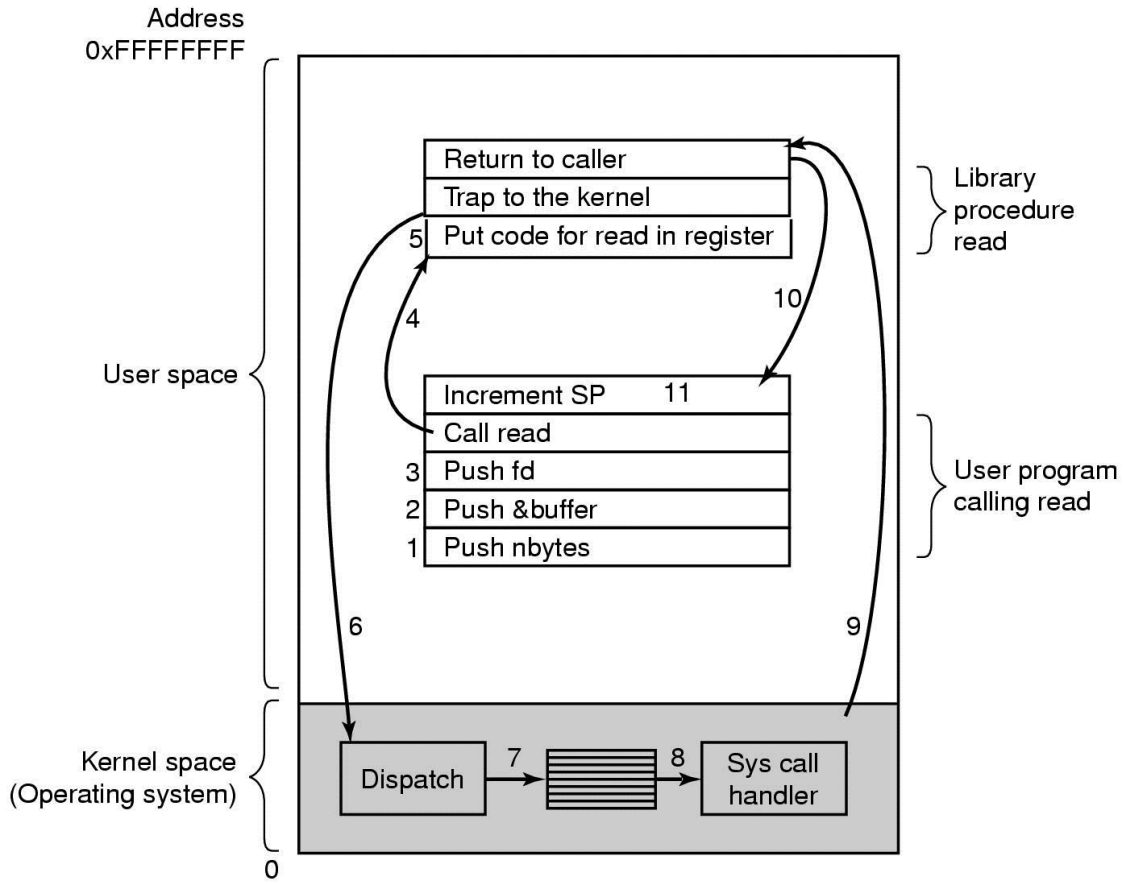
- Interface that programs can use in order to obtain a variety of services provided by the operating system
- Most CPUs have two modes, kernel mode and user mode.
  - A bit in the PSW(Program Status Word) usually controls the mode.
  - When running in kernel mode, the CPU can execute every instruction in its instruction set and use every feature of the hardware
  - The operating system runs in kernel mode, giving it access to the complete hardware
  - User programs run in user mode, which permits only a subset of the instructions to be executed and a subset of features to be accessed.
  - Generally, all instructions involving I/O and memory protection are disallowed in user mode.
- To obtain services from the operating system, a user program must make a system call which traps into the kernel and invokes the operating system
- The Trap instruction switches from user mode to kernel mode and starts the operating system.
- When the work has been completed, control is returned to the user program at the instruction following the system call.
- Most of the other traps are caused by hardware to warn of an exceptional situation such as an attempt to divide by 0.

System Calls

- A procedure library is provided to make it possible to make system calls from C programs and often from other languages as well
  - The actual mechanics of issuing a system call are highly machine dependent and often must be expressed in assembly code
- Steps in making system calls
  - If a process is running a user program in user mode and needs a system service such as reading data from a file, it executes a trap or system call instruction to transfer control to the operating system
  - The operating system then figures out what the calling process wants by inspecting the parameters.
  - Then it carries out the system call and returns control to the instruction following the system call.
- Making a system call is like making a special kind of procedure call, only system calls enter the kernel and procedure calls do not.

Steps in Making a System Call

- e.g. read system call
  - count = read (fd, buffer, nbytes)
    - fd: file
    - buffer: pointer to the buffer
    - nbytes: number of bytes to read
    - count: number of bytes actually read



System Calls

- For each system call in UNIX, there is roughly one library procedure that is called to invoke it.
- POSIX has about 100 procedure calls.(library procedures that make system calls)
- The services offered by these calls determine most of what the operating system has to do
- Services include things like
  - creating and terminating processes,
  - creating, deleting, reading, and writing files, managing directories,
  - performing input and output

Some of the Major POSIX System Calls

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

Directory and file system management

Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Miscellaneous

Call	Description
s = chdir(dirname)	Change the working directory
s = chmod(name, mode)	Change a file's protection bits
s = kill(pid, signal)	Send a signal to a process
seconds = time(&seconds)	Get the elapsed time since Jan. 1, 1970