**SYSTEM CALLS:**

**Fork:** **fork()** creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process. Both run in separate memory spaces.

Argument value is void.

**Kill:** The **kill()** call can be used to send any signal to any group or process.

Arguments are (pid\_t p) and (int sig): the pid declares where the sig have to go, for example: if pid is positive, then sig is sent to the process with the ID specified by pid.

**Stat:** These functions return information about a file, in the buffer pointed to by buf. No permissions are

required on the file itself, but—in the case of stat(), fstatat(), and lstat()—execute (search) permission is

required on all of the directories in pathname that lead to the file

Argument value is void.

**Mmap:**  mmap() creates a new mapping in the virtual address space of the calling process.

Arguments:Starting address for the new mapping is specified in (void) \*addr. The (size\_t) length argument specifies the length of the mapping. The (int) prot describes the desired memory protection of the mapping. The (int) flags argument determines whether updates to the mapping are visible to other processes mapping the same region.

**Chmod:**  chmod() changes the permissions of the file specified whose pathname is given in pathname, which is dereferenced if it is a symbolic link.

Arguments: In (const char\*) pathname is the name of the path. The new file permissions are specified in (mode\_t) mode.

**Waitpid:** The waitpid() system call suspends execution of the calling process until a child specified by pid argument has changed state. By default, waitpid() waits only for terminated children, but this behaviour is modifiable via the options argument.

Arguments: The options argument can modify for what waitpid() waits. (int\* status) Waitpid() store status information in the int to which it points. Value of (pid\_t) pid decides for what child process it waits.

**SYSTEM CALLS FAIL:**

fork(): Not enough memory for kernel to copy page tables and other accounting information of parent process.

exec: The new process file is not an ordinary file.

unlink: pathname points outside your accessible address space.

read: buf is outside your accessible address space.

mount: One of the pointer arguments points outside the user address space.

chmod: pathname points outside your accessible address space.

kill: An invalid signal was specified.

**What is a trap instruction?**

In [computing](https://en.wikipedia.org/wiki/Computing) and [operating systems](https://en.wikipedia.org/wiki/Operating_systems), a trap, also known as an exception or a fault, is typically a type of [synchronous](https://en.wikipedia.org/wiki/Synchronization_%28computer_science%29) [interrupt](https://en.wikipedia.org/wiki/Interrupt) typically caused by an [exceptional](https://en.wikipedia.org/wiki/Exception_handling) condition (e.g., [breakpoint](https://en.wikipedia.org/wiki/Breakpoint" \l "Hardware" \o "Breakpoint), [division by zero](https://en.wikipedia.org/wiki/Division_by_zero), [invalid memory access](https://en.wikipedia.org/wiki/Segmentation_fault)). A trap usually results in a switch to [kernel mode](https://en.wikipedia.org/wiki/Kernel_mode), wherein the operating system performs some action before returning control to the originating process. A trap in a system [process](https://en.wikipedia.org/wiki/Process_%28computing%29) is more serious than a trap in a user [process](https://en.wikipedia.org/wiki/Process_%28computing%29), and in some systems is fatal. In some usages, the term *trap* refers specifically to an interrupt intended to initiate a [context switch](https://en.wikipedia.org/wiki/Context_switch) to a [monitor program](https://en.wikipedia.org/wiki/Profiler_%28computer_science%29) or [debugger](https://en.wikipedia.org/wiki/Debugger).