System call

A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it requests the operating system's kernel. System call provides the services of the operating system to the user programs via the Application Program Interface(API). System calls are the only entry points into the kernel system and are executed in kernel mode.

Kill() system call

In operating systems, process management is essential for controlling the execution of programs. It involves process creation, scheduling, and termination to ensure efficient use of system resources.

One of the essential tools in this regard is the kill system call. It allows one process to send signals to another process, facilitating communication and control between processes. While the name might suggest that it only kills processes, not always so. It can kill, continue, or communicate with other processes depending on the type of signal it sends.

int kill(pid_t pid, int sig);

This system call takes two parameters: a process ID (PID) and a signal number. The process ID tells the system which program to operate upon. The signal number tells it what kind of operation to do. If the signal is 9, for example, that tells the system to kill the process right now. That's the signal called SIGKILL. If the signal is 15, or SIGTERM, it requests the process to terminate gracefully, allowing it to clean up first. There's also SIGSTOP to stop a process and SIGCONT to continue one.

When a program call kill, the system will first determine whether that program is allowed to send the signal. Usually, that is determined by user privileges. If everything is okay, the system finds the process whose PID was given and delivers the signal. Depending on how the receiving process is set up, that's what happens next. Certain programs ignore certain signals. Certain ones have particular rules that must be obeyed when receiving them. If there is no special rule, the system executes the default action for the signal, normally killing the program.

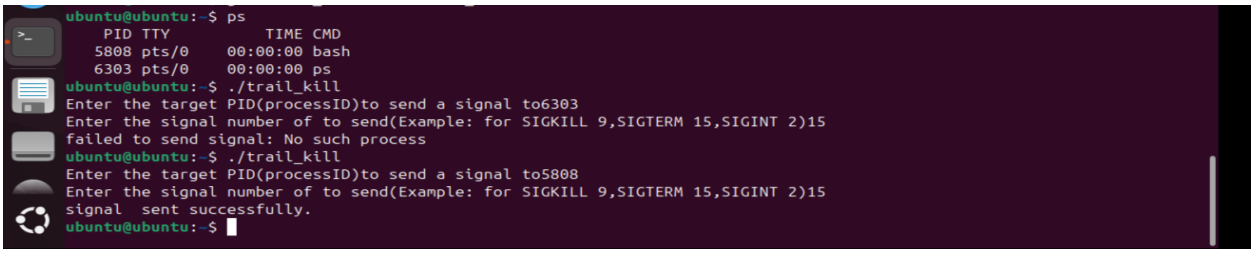
In everyday use, kill can be employed for controlling background programs. It can terminate hung or unwanted tasks. It can also be called from scripts or utilities that control other programs.

Kill() system call implementation

OpenHarmony is an open-source operating system for a broad range of devices from IoT, to mobile, to laptops and desktops. It is built with kernel_liteos_a which is a lightweight, POSIX-compliant kernel. POSIX compliance allows any programs in standard C, including programs with system calls in the OpenHarmony environment to run successfully, including the `kill` command.

However, interactive testing of the kill system call on OpenHarmony was not able to be accomplished. Virtualizing OpenHarmony was attempted in different virtualization platforms, such as QEMU, was also not possible because neither the boot nor the run of the system were successful. These limitations made runtime testing of the kill system call in OpenHarmony directly not possible.

Testing was done on Ubuntu, which is a well-known POSIX-compliant Linux. OpenHarmony and Ubuntu both are POSIX-compliant systems so the functionality of the kill system call is equivalent behavior. Because of this equivalence, Ubuntu is a suitable substitute for functional verification and usage condition testing of the kill system call when direct testing on OpenHarmony is not possible.


ubuntu@ubuntu:~$ ps
PID TTY TIME CMD
5808 pts/0 00:00:00 bash
6303 pts/0 00:00:00 ps
ubuntu@ubuntu:~$ ./trail_kill
Enter the target PID(processID)to send a signal to6303
Enter the signal number of to send(Example: for SIGKILL 9,SIGTERM 15,SIGINT 2)15
failed to send signal: No such process
ubuntu@ubuntu:~$ ./trail_kill
Enter the target PID(processID)to send a signal to5808
Enter the signal number of to send(Example: for SIGKILL 9,SIGTERM 15,SIGINT 2)15
signal sent successfully.
ubuntu@ubuntu:~$