SignalsAndTraps

April 15, 2024

1 Signals and Traps?

In the context of Unix-like operating systems, signals are a form of inter-process communication. They are software interrupts that provide a way for a process to react to a specific event. For example, when you press `Ctrl+C` in the terminal, a `SIGINT` (Signal Interrupt) is sent to the current foreground process in the terminal.

Here are some common signals:

- `SIGINT`: Interrupt from keyboard (usually initiated by `Ctrl+C`).
- `SIGTERM`: Termination signal. This is the default signal sent by the `kill` command.
- `SIGHUP`: Hang up detected on controlling terminal or death of controlling process.
- `SIGKILL`: Kill signal. This signal cannot be caught or ignored, and is used to kill unresponsive processes.
- `SIGSTOP`: Stop the process. This signal cannot be caught or ignored.

Traps, on the other hand, are a way to specify actions that will be taken when a process receives specific signals. In shell scripting, you can use the `trap` command to catch signals and execute a piece of code when they are received. This can be used to clean up resources before a script exits, among other things.

Here's an example of how to use a trap in a bash script:

[ ]:
```bash
#!/bin/bash

# Define a cleanup function
cleanup() {
    echo "Cleaning up..."
    rm -f temp_file
}

# Set the trap
trap cleanup SIGINT SIGTERM

# Create a temporary file
touch temp_file

# Wait to simulate some work
sleep 60
```

In this script, the `cleanup` function will be called if the script receives a `SIGINT` or `SIGTERM` signal. The function removes a temporary file, simulating some cleanup work. The `sleep 60` command is used to keep the script running for a while to give you time to send a signal to it.

2 List of Signals

There is an easy way to list down all the signals supported by your system. Just issue the kill -l command and it would display all the supported signals –

$ kill -l 1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL 5) SIGTRAP 6) SIGABRT 7) SIGBUS 8) SIGFPE 9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP 21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO 30) SIGPWR 31) SIGSYS 34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8 43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2 63) SIGRTMAX-1 64) SIGRTMAX

3 How can I send a specific signal to a process in Unix-like operating systems?

In Unix-like operating systems, you can send a specific signal to a process using the `kill` command. Despite its name, `kill` is used to send signals, not just to terminate processes.

Here's the general syntax of the `kill` command:

[ ]:
```
kill -SIGNAL PID
```

In this command, `SIGNAL` is the signal that you want to send, and `PID` is the Process ID of the process that you want to send the signal to.

For example, to send the `SIGTERM` signal to a process with a PID of 1234, you would use the following command:

[ ]:
```
kill -TERM 1234
```

You can also use the signal number instead of the signal name. For example, `SIGTERM` is signal 15, so the following command is equivalent to the previous one:

[ ]:
```
kill -15 1234
```

If you don't specify a signal, `kill` will send `SIGTERM` by default.

To send a signal to multiple processes, you can specify multiple PIDs:

[ ]:
```
kill -TERM 1234 5678 9012
```

Note that you need to have the necessary permissions to send a signal to a process. Typically, you can send signals to your own processes, but you need to be the superuser (root) to send signals to other users' processes.

4 Trapping Signals?

Trapping signals is a method used in Unix-like operating systems to specify actions that should be taken when a process receives a specific signal. This is often used in shell scripts to handle signals and perform cleanup or other tasks before a script exits.

The `trap` command is used to trap signals. Here's the general syntax:

[ ]:
```
trap command signal
```

In this command, `command` is the command to be executed when the specified signal is received, and `signal` is the signal to be trapped.

For example, the following command traps the `SIGINT` signal and executes the `echo` command when it's received:

[ ]:
```
trap 'echo You pressed Ctrl+C!' SIGINT
```

If you run this command in a shell and then press `Ctrl+C`, the shell will print "You pressed Ctrl+C!" instead of terminating the process.

You can trap multiple signals by specifying them after the command:

[ ]:
```
trap 'echo Signal received!' SIGINT SIGTERM
```

In this example, the `echo` command will be executed when either `SIGINT` or `SIGTERM` is received.

You can also use the `trap` command with a function to perform more complex tasks:

[ ]:
```
cleanup() {
    echo "Cleaning up..."
    rm -f temp_file
}

trap cleanup SIGINT SIGTERM
```

In this example, the `cleanup` function will be called when either `SIGINT` or `SIGTERM` is received.

To remove a trap, you can use the `trap` command with `-` and the signal:

[ ]:
```
trap - SIGINT
```

This command removes the trap for `SIGINT`.

Note that not all signals can be trapped. For example, `SIGKILL` and `SIGSTOP` cannot be trapped, blocked, or ignored.

5 Ignoring Signals?

Ignoring signals in Unix-like operating systems means that a process or a shell script will not perform any action when it receives a specific signal. This can be useful in situations where you don't want a process to be interrupted by signals like `SIGINT` (which is sent when you press `Ctrl+C`) or `SIGTERM` (which is sent when the system is shutting down).

You can ignore signals in a shell script using the `trap` command with an empty string as the command to be executed:

[ ]:
```
trap '' signal
```

In this command, `signal` is the signal to be ignored.

For example, the following command ignores the `SIGINT` signal:

[ ]:
```
trap '' SIGINT
```

If you run this command in a shell and then press `Ctrl+C`, the shell will ignore the `SIGINT` signal and will not terminate the process.

You can ignore multiple signals by specifying them after the command:

[ ]:
```
trap '' SIGINT SIGTERM
```

In this example, both `SIGINT` and `SIGTERM` will be ignored.

To stop ignoring a signal, you can use the `trap` command with `-` and the signal:

[ ]:
```
trap - SIGINT
```

This command stops ignoring the `SIGINT` signal.

Note that not all signals can be ignored. For example, `SIGKILL` and `SIGSTOP` cannot be trapped, blocked, or ignored.

5.1 Resetting Traps?

Resetting traps in Unix-like operating systems means restoring the default action for a specific signal. This can be done using the `trap` command with `-` as the command to be executed:

[ ]:
```
trap - signal
```

In this command, `signal` is the signal for which the default action should be restored.

For example, the following command resets the trap for the `SIGINT` signal:

[ ]:
```
trap - SIGINT
```

If you run this command in a shell where `SIGINT` was previously trapped or ignored, the shell will restore the default action for `SIGINT` (which is to terminate the process) when it receives this signal.

You can reset traps for multiple signals by specifying them after the command:

[ ]:
```
trap - SIGINT SIGTERM
```

In this example, the traps for both `SIGINT` and `SIGTERM` will be reset.

Note that not all signals can be trapped or ignored. For example, `SIGKILL` and `SIGSTOP` cannot be trapped, blocked, or ignored.

6 Thank You!