SIGNAL HANDLING IN LINUX

Varunnavie TN
23pw39

What is a Signal?

- A signal is an **asynchronous** event which is delivered to a process.
- Asynchronous means that the event can occur at any time may be unrelated to the execution of the process.
- Signals are raised by some error conditions, such as memory segment violations, floating point processor errors, or illegal instructions.

– e.g. user types ctrl-C

Why Do Signals Exist?

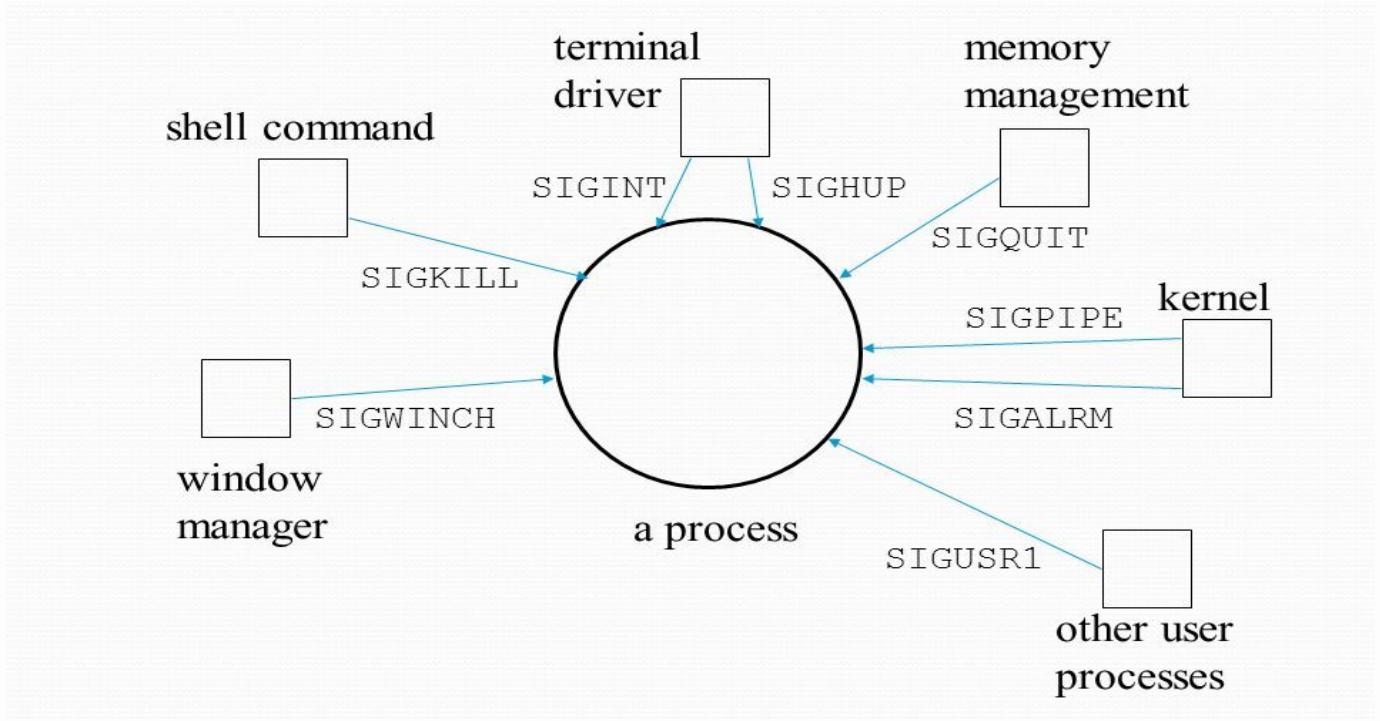
Signals exist to facilitate communication and coordination between processes. They provide an effective way to handle events and conditions in an operating system. Some common examples of signals include process termination (SIGTERM), process interruption (SIGINT), or detection of an error (SIGSEGV).

When Are Signals Delivered and by Whom?

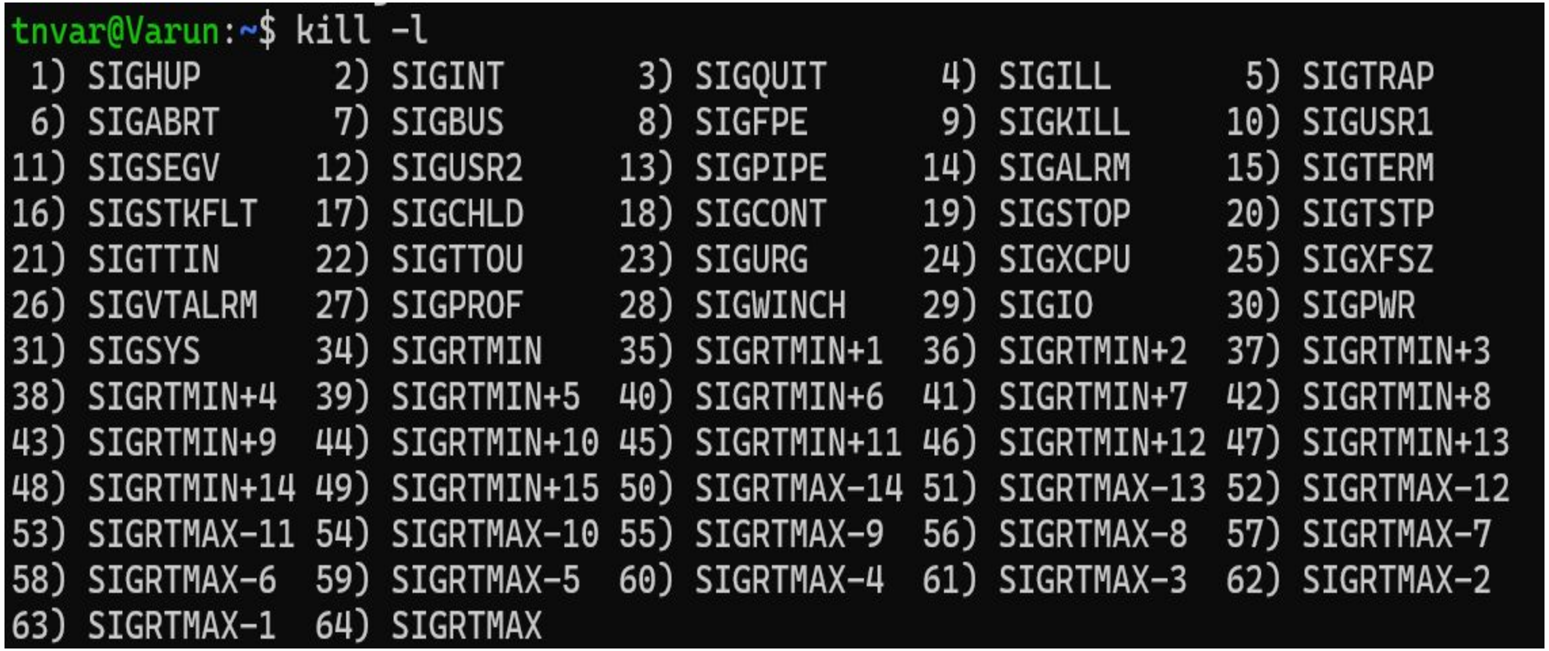
Signals can be delivered at various times:

- ***By the Operating System:*** The operating system can send signals to processes to inform them about system events, such as the termination of a child process or a request for interruption.
- ***By Other Processes:*** Processes can also send signals to other processes, allowing communication between them. This is useful for coordinating activities or notifying events. (kill())

Signal Sources



POSIX Predefined Signals



```
tnvar@Varun:~$ 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
```

- **SIGALRM**: Alarm timer time-out. Generated by alarm( ) API.
- **SIGABRT**: Abort process execution. Generated by abort( ) API.
- **SIGFPE**: Illegal mathematical operation.
- **SIGHUP**: Controlling terminal hang-up.
- **SIGILL**: Execution of an illegal machine instruction.
- **SIGINT**: Process interruption. Can be generated by or keys.
- **SIGKILL**: Sure kill a process. Can be generated by – "kill -9 " command.
- **SIGPIPE**: Illegal write to a pipe.
- **SIGQUIT**: Process quit. Generated by keys.
- **SIGSEGV**: Segmentation fault. generated by dereferencing a NULL pointer.
- **SIGTERM**: process termination. Can be generated by – "kill " command.
- **SIGUSR1**: Reserved to be defined by user.
- **SIGUSR2**: Reserved to be defined by user.
- **SIGCHLD**: Sent to a parent process when its child process has terminated.
- **SIGCONT**: Resume execution of a stopped process.
- **SIGSTOP**: Stop a process execution.
- **SIGTTIN**: Stop a background process when it tries to read from from its controlling terminal.
- **SIGTSTP**: Stop a process execution by the control_Z keys.
- **SIGTTOU**: Stop a background process when it tries to write to its controlling terminal.

Actions on signals

Process that receives a signal can take one of three action:

- Perform the system-specified default for the signal
  - notify the parent process that it is terminating;
  - generate a core dump file; (a file containing the current memory image of the process)
  - terminate.

- Ignore the signal
  - A process can do ignoring with all signal but two special signals: SIGSTOP and SIGKILL.

- Catch the Signal
  - When a process catches a signal, except SIGSTOP and SIGKILL, it invokes a special signal handling routine.

Example of signals

When user types Ctrl-c

- Event gains attention of OS
- OS stops the application process immediately, sending it a 2/SIGINT signal
- Signal handler for 2/SIGINT signal executes to completion
- Default signal handler for 2/SIGINT signal exits process

Signal Number

Process makes illegal memory reference

- Event gains attention of OS
- OS stops application process immediately, sending it a 11/SIGSEGV signal
- Signal handler for 11/SIGSEGV signal executes to completion
- Default signal handler for 11/SIGSEGV signal prints “segmentation fault” and exits process

Send signals via commands

kill Command

- kill -signal pid
  - Send a signal of type signal to the process with id pid
  - Can specify either signal type name (SIGINT) or number (2)

  - **If no signal type name or number specified => sends 15/SIGTERM signal**

Default 15/SIGTERM handler exits process

- Better command name would be sendsig

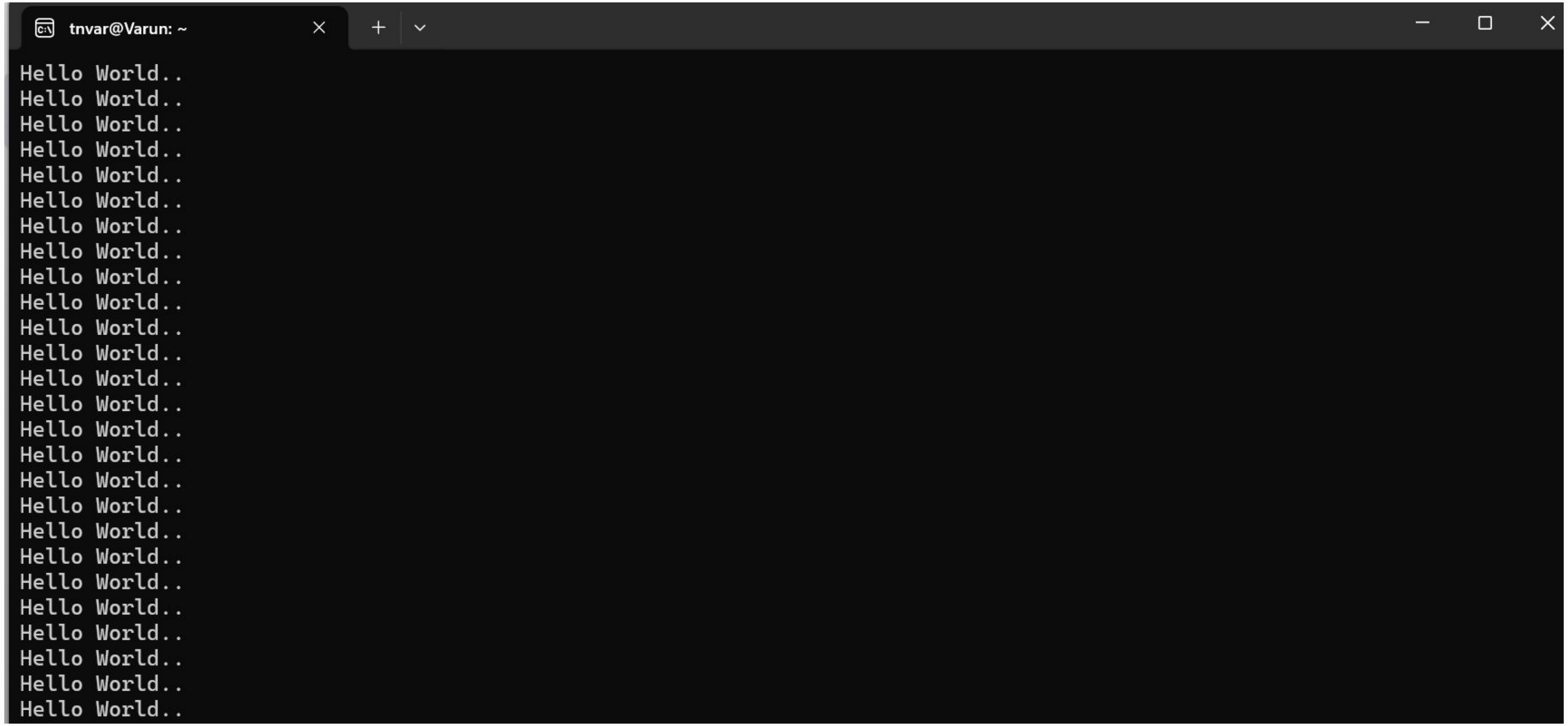
Examples

- kill -2 1234
- kill SIGINT 1234
  - Same as pressing Ctrl-c if process 1234 is running in foreground

Demonstration

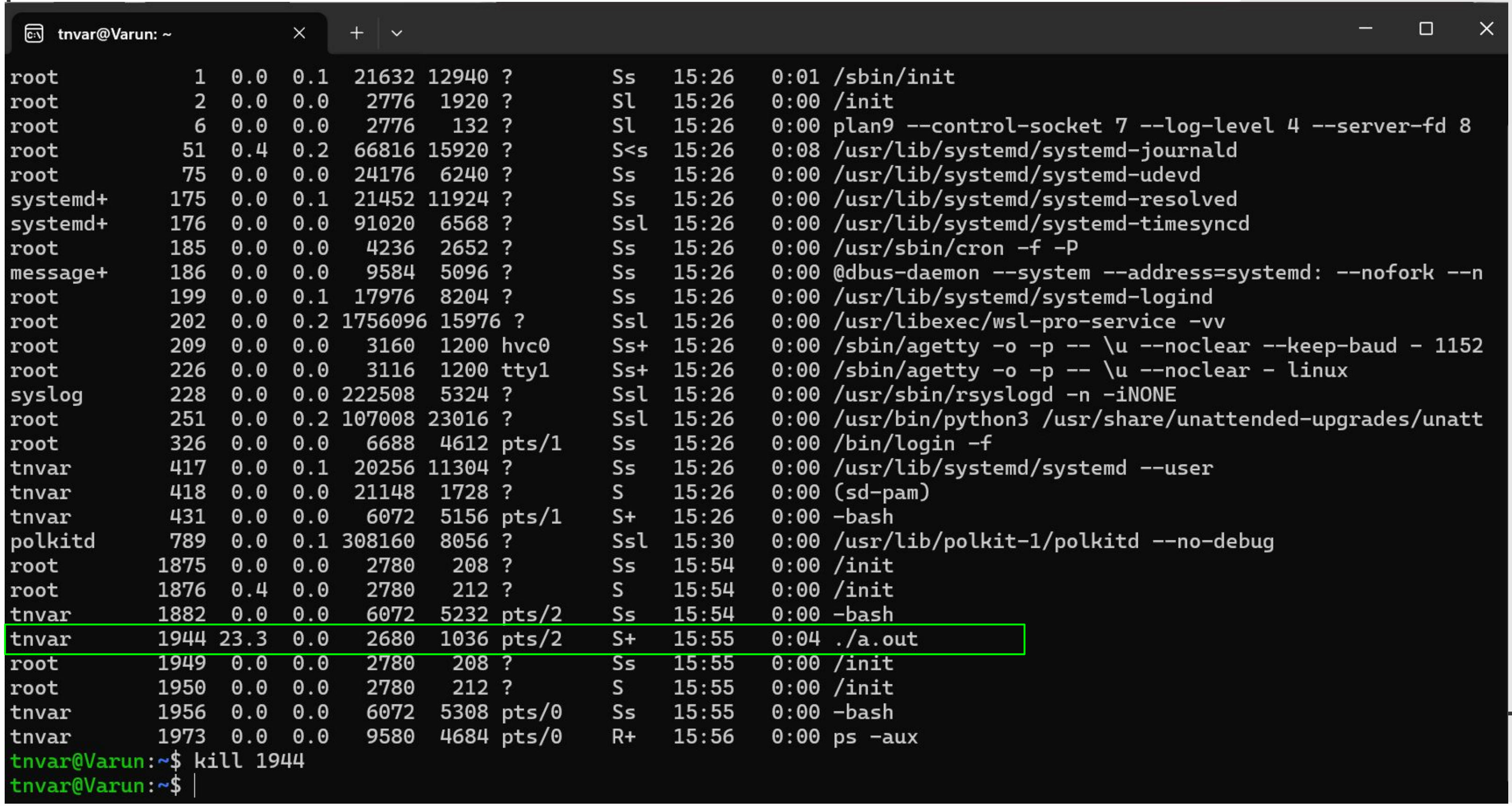
```
#include int main() {
    while(1) {

        printf("Hello World...\n");
        return 0;
    }
}
```

OUTPUT

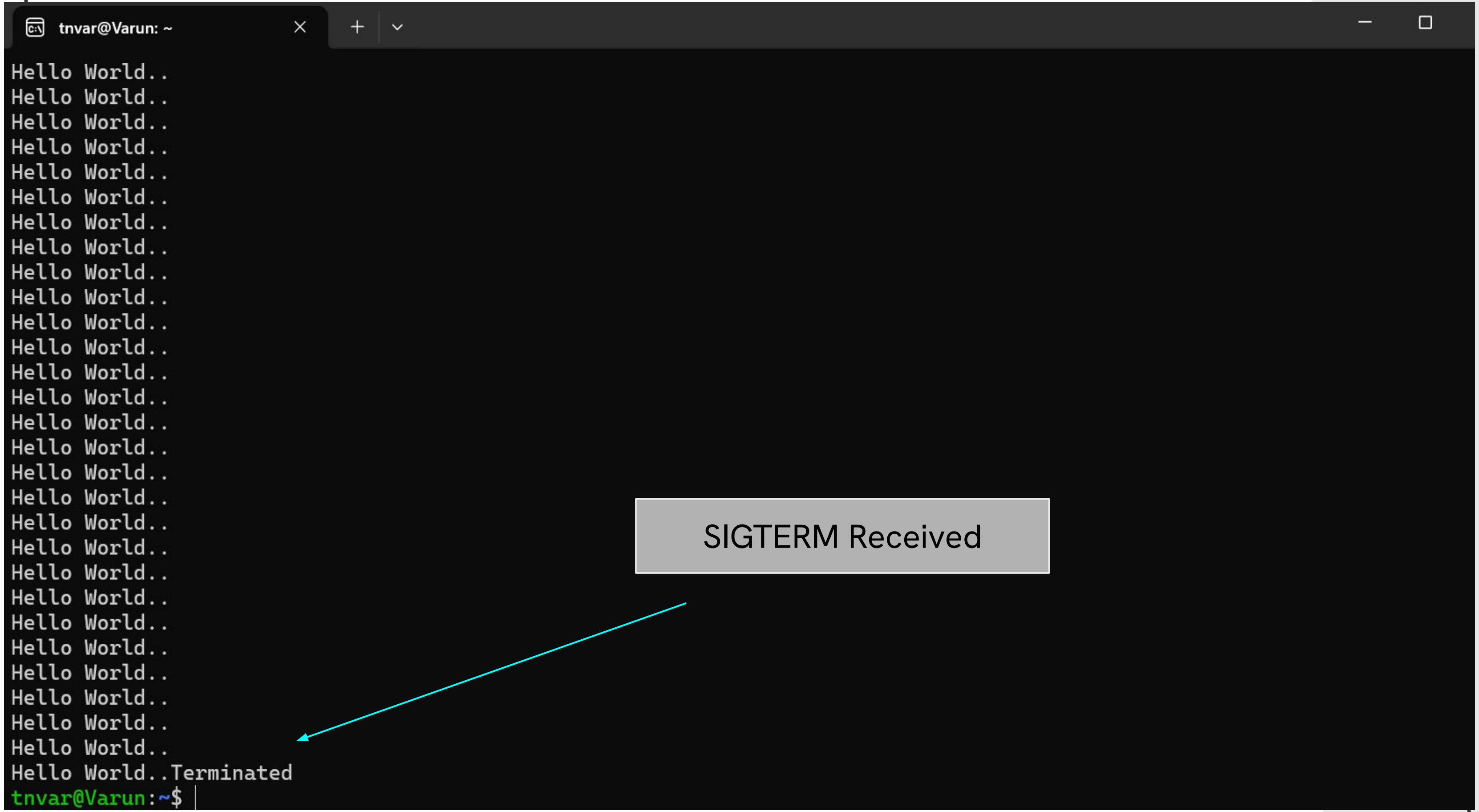


```
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
```

Go to new terminal and check the process list (ps -aux)



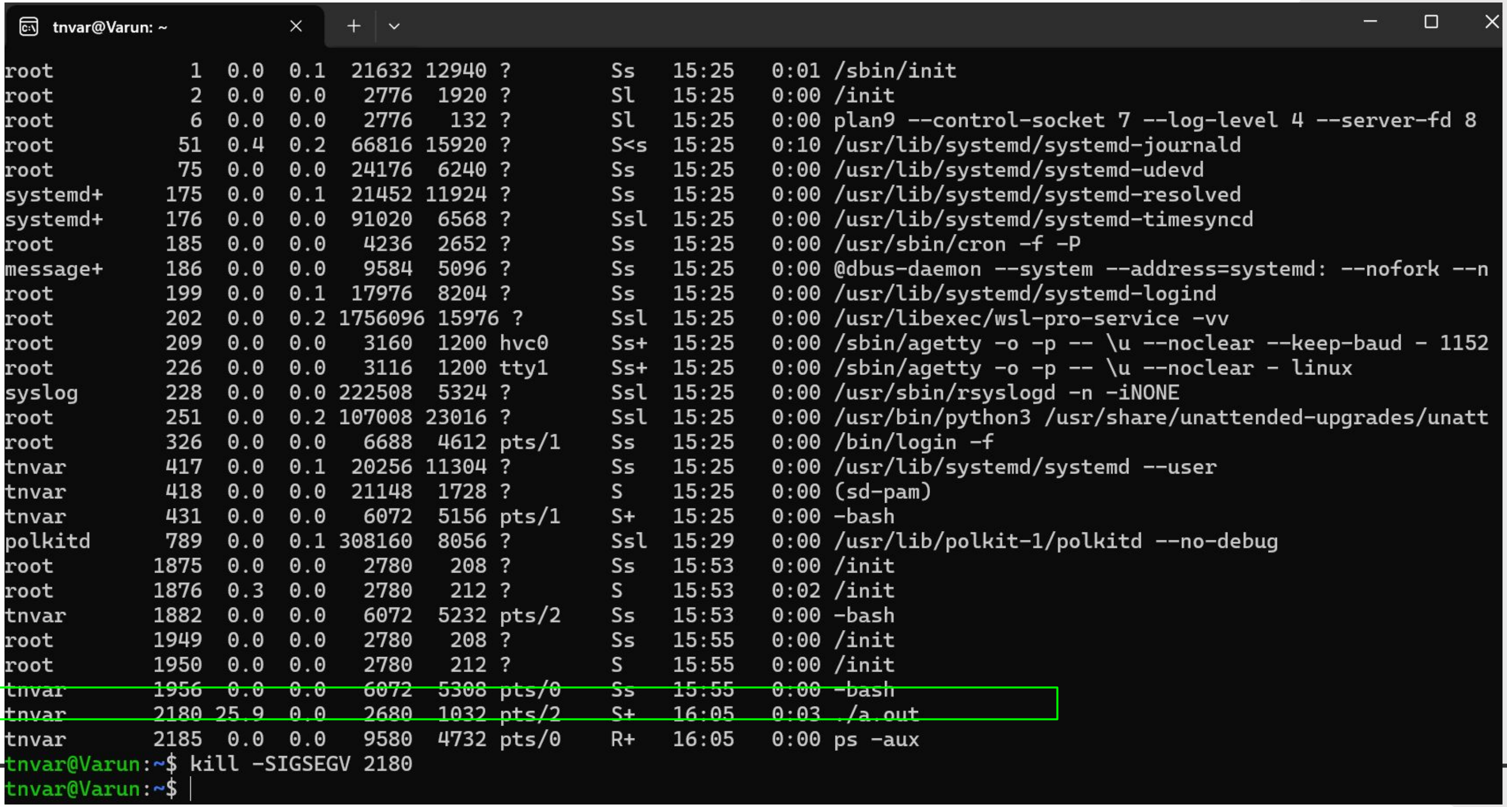
```
root           1  0.0  0.1  21632 12940 ?        Ss   15:26   0:01 /sbin/init
root           2  0.0  0.0   2776  1920 ?        Sl   15:26   0:00 /init
root           6  0.0  0.0   2776   132 ?        Sl   15:26   0:00 plan9 --control-socket 7 --log-level 4 --server-fd 8
root          51  0.4  0.2  66816 15920 ?        S<s  15:26   0:08 /usr/lib/systemd/systemd-journald
root          75  0.0  0.0  24176  6240 ?        Ss   15:26   0:00 /usr/lib/systemd/systemd-udevd
systemd+     175  0.0  0.1  21452 11924 ?        Ss   15:26   0:00 /usr/lib/systemd/systemd-resolved
systemd+     176  0.0  0.0  91020  6568 ?        Ssl  15:26   0:00 /usr/lib/systemd/systemd-timesyncd
root         185  0.0  0.0   4236  2652 ?        Ss   15:26   0:00 /usr/sbin/cron -f -P
message+     186  0.0  0.0   9584  5096 ?        Ss   15:26   0:00 @dbus-daemon --system --address=systemd: --nofork --n
root         199  0.0  0.1  17976  8204 ?        Ss   15:26   0:00 /usr/lib/systemd/systemd-logind
root         202  0.0  0.2 1756096 15976 ?       Ssl  15:26   0:00 /usr/libexec/wsl-pro-service -vv
root         209  0.0  0.0   3160  1200 hvc0     Ss+  15:26   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud - 1152
root         226  0.0  0.0   3116  1200 tty1     Ss+  15:26   0:00 /sbin/agetty -o -p -- \u --noclear - linux
syslog       228  0.0  0.0 222508  5324 ?        Ssl  15:26   0:00 /usr/sbin/rsyslogd -n -iNONE
root         251  0.0  0.2 107008 23016 ?        Ssl  15:26   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unatt
root         326  0.0  0.0   6688  4612 pts/1    Ss   15:26   0:00 /bin/login -f
tnvar        417  0.0  0.1  20256 11304 ?        Ss   15:26   0:00 /usr/lib/systemd/systemd --user
tnvar        418  0.0  0.0  21148  1728 ?        S    15:26   0:00 (sd-pam)
tnvar        431  0.0  0.0   6072  5156 pts/1    S+   15:26   0:00 -bash
polkitd      789  0.0  0.1 308160  8056 ?        Ssl  15:30   0:00 /usr/lib/polkit-1/polkitd --no-debug
root        1875  0.0  0.0   2780   208 ?        Ss   15:54   0:00 /init
root        1876  0.4  0.0   2780   212 ?        S    15:54   0:00 /init
tnvar       1882  0.0  0.0   6072  5232 pts/2    Ss   15:54   0:00 -bash
tnvar       1944 23.3  0.0   2680  1036 pts/2    S+   15:55   0:04 ./a.out
root        1949  0.0  0.0   2780   208 ?        Ss   15:55   0:00 /init
root        1950  0.0  0.0   2780   212 ?        S    15:55   0:00 /init
tnvar       1956  0.0  0.0   6072  5308 pts/0    Ss   15:55   0:00 -bash
tnvar       1973  0.0  0.0   9580  4684 pts/0    R+   15:56   0:00 ps -aux
tnvar@Varun:~$ kill 1944
tnvar@Varun:~$ |
```



```
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..Terminated
tnvar@Varun:~$
```

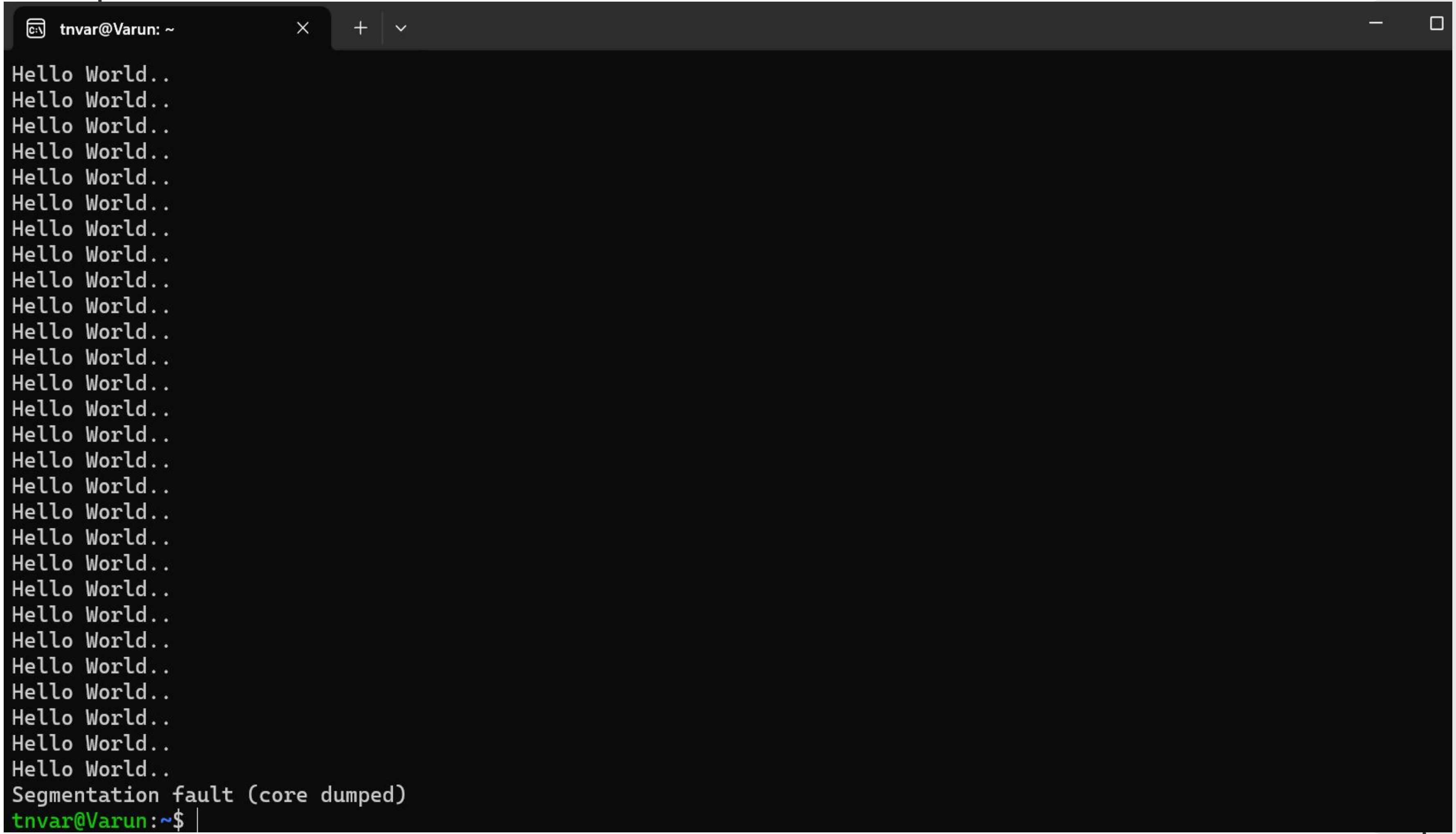
SIGTERM Received

Killing process by different signals



tnvar@Varun: ~

```
root           1  0.0  0.1  21632 12940 ?        Ss   15:25   0:01 /sbin/init
root           2  0.0  0.0   2776  1920 ?        Sl   15:25   0:00 /init
root           6  0.0  0.0   2776   132 ?        Sl   15:25   0:00 plan9 --control-socket 7 --log-level 4 --server-fd 8
root          51  0.4  0.2  66816 15920 ?        S<s  15:25   0:10 /usr/lib/systemd/systemd-journald
root          75  0.0  0.0  24176  6240 ?        Ss   15:25   0:00 /usr/lib/systemd/systemd-udevd
systemd+     175  0.0  0.1  21452 11924 ?        Ss   15:25   0:00 /usr/lib/systemd/systemd-resolved
systemd+     176  0.0  0.0  91020  6568 ?        Ssl  15:25   0:00 /usr/lib/systemd/systemd-timesyncd
root         185  0.0  0.0   4236  2652 ?        Ss   15:25   0:00 /usr/sbin/cron -f -P
message+     186  0.0  0.0   9584  5096 ?        Ss   15:25   0:00 @dbus-daemon --system --address=systemd: --nofork --n
root         199  0.0  0.1  17976  8204 ?        Ss   15:25   0:00 /usr/lib/systemd/systemd-logind
root         202  0.0  0.2 1756096 15976 ?       Ssl  15:25   0:00 /usr/libexec/wsl-pro-service -vv
root         209  0.0  0.0   3160  1200 hvc0     Ss+  15:25   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud - 1152
root         226  0.0  0.0   3116  1200 tty1     Ss+  15:25   0:00 /sbin/agetty -o -p -- \u --noclear - linux
syslog       228  0.0  0.0 222508  5324 ?        Ssl  15:25   0:00 /usr/sbin/rsyslogd -n -iNONE
root         251  0.0  0.2 107008 23016 ?        Ssl  15:25   0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unatt
root         326  0.0  0.0   6688  4612 pts/1    Ss   15:25   0:00 /bin/login -f
tnvar        417  0.0  0.1  20256 11304 ?        Ss   15:25   0:00 /usr/lib/systemd/systemd --user
tnvar        418  0.0  0.0  21148  1728 ?        S    15:25   0:00 (sd-pam)
tnvar        431  0.0  0.0   6072  5156 pts/1    S+   15:25   0:00 -bash
polkitd      789  0.0  0.1 308160  8056 ?        Ssl  15:29   0:00 /usr/lib/polkit-1/polkitd --no-debug
root        1875  0.0  0.0   2780   208 ?        Ss   15:53   0:00 /init
root        1876  0.3  0.0   2780   212 ?        S    15:53   0:02 /init
tnvar       1882  0.0  0.0   6072  5232 pts/2    Ss   15:53   0:00 -bash
root        1949  0.0  0.0   2780   208 ?        Ss   15:55   0:00 /init
root        1950  0.0  0.0   2780   212 ?        S    15:55   0:00 /init
tnvar       1956  0.0  0.0   6072  5308 pts/0    Ss   15:55   0:00 -bash
tnvar       2180 25.9  0.0   2680  1032 pts/2    S+   16:05   0:03 ./a.out
tnvar       2185  0.0  0.0   9580  4732 pts/0    R+   16:05   0:00 ps -aux
tnvar@Varun:~$ kill -SIGSEGV 2180
tnvar@Varun:~$ 
```



```
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Hello World..
Segmentation fault (core dumped)
tnvar@Varun:~$
```

Signal Concepts

Signals are defined in

- man 7 signal for complete list of signals and their numeric values.
- kill –l for full list of signals on a system.
- 64 signals. The first 32 are traditional signals, the rest are for real time applications

Signal Function

Programs can handle signals using the signal library function.

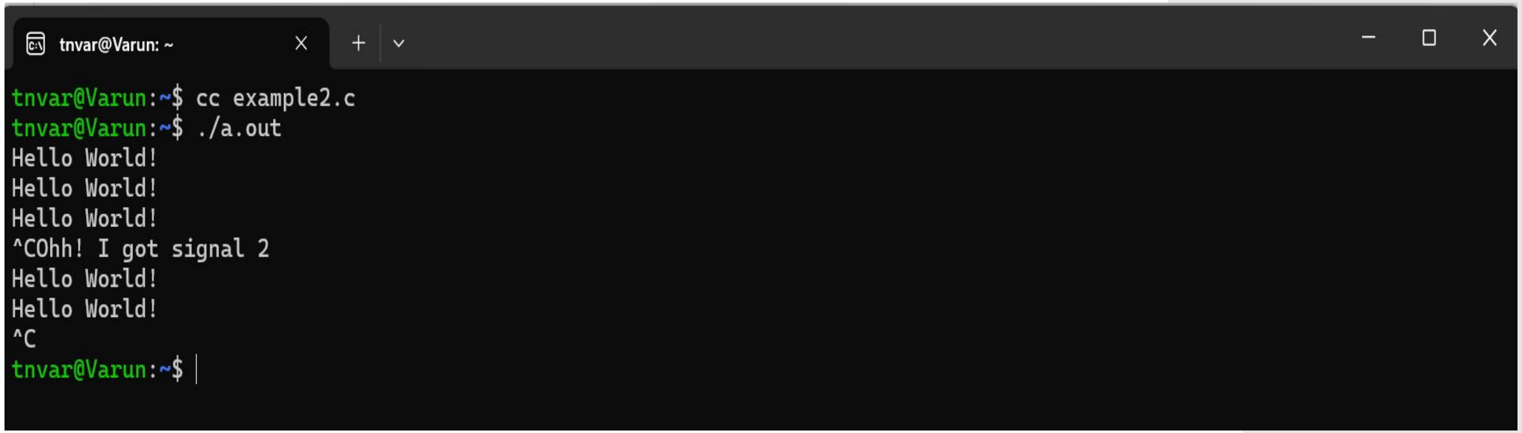
void (*signal(int signo, void (*func)(int)))(int);

- signo is the signal number to handle
- func defines how to handle the signal
  - SIG_IGN
  - SIG_DFL
  - Function pointer of a custom handler

Example 1:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void ohh(int sig) {
    printf("Ohh! I got signal %d\n", sig);
    (void) signal(SIGINT, SIG_DFL);
}

int main() {
    (void) signal(SIGINT, ohh);

    while (1) {
        printf("Hello World!\n");
        sleep(1);
    }

    return 0;
}
```

OUTPUT

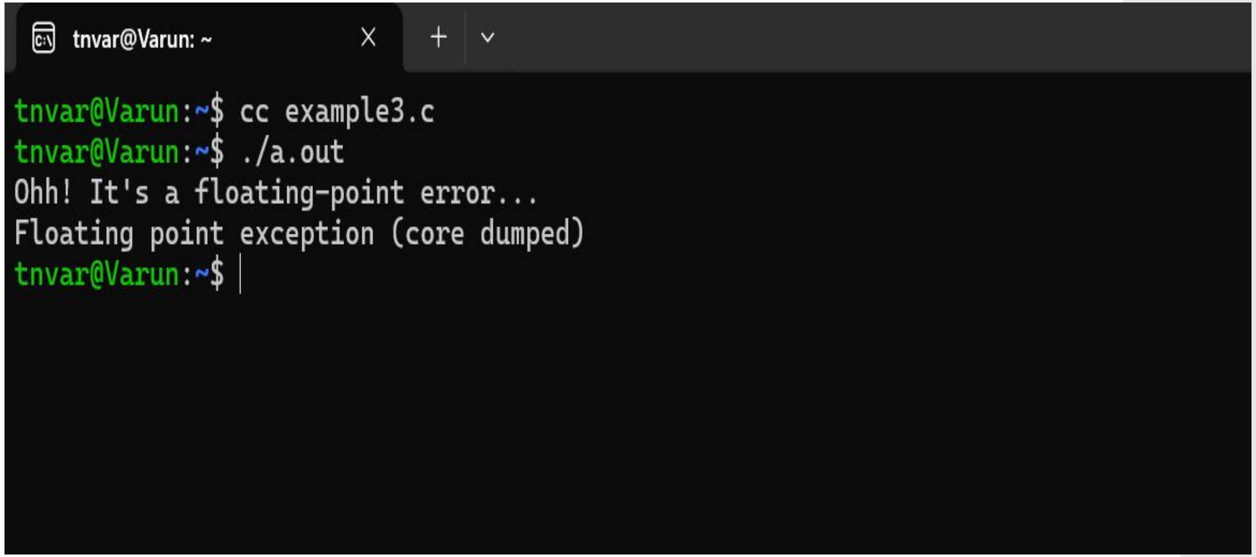

tnvar@Varun: ~

```
tnvar@Varun:~$ cc example2.c
tnvar@Varun:~$ ./a.out
Hello World!
Hello World!
Hello World!
^COhh! I got signal 2
Hello World!
Hello World!
^C
tnvar@Varun:~$
```

Example 2:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

void error(int sig) {
    printf("Ohh! It's a floating-point error...\n");
    (void) signal(SIGFPE, SIG_DFL);
}

int main() {
    (void) signal(SIGFPE, error);

    int a = 12, b = 0, result;
    result = a / b;  // This will cause a floating-point exception (division by zero)

    printf("Result is: %d\n", result);
    return 0;
}
```

OUTPUT



```
tnvar@Varun:~$ cc example3.c
tnvar@Varun:~$ ./a.out
Ohh! It's a floating-point error...
Floating point exception (core dumped)
tnvar@Varun:~$
```

sigaction

int sigaction(int sig, const struct sigaction *act, struct sigaction *oact);

*act : A pointer to a `sigaction` structure that specifies **how to handle the signal**.

*oact : If not NULL, stores the **previous signal handler settings** (optional).

The sigaction structure, used to define the actions to be taken on receipt of the signal specified by sig, is defined in signal.h and has at least the following members:

void (*) (int) sa_handler	**function, SIG_DFL or SIG_IGN**
sigset_t sa_mask	**signals to block in sa_handler**
int sa_flags	**signal action modifiers**

The sigaction function sets the action associated with the signal sig . If oact is not null, sigaction writes the previous signal action to the location it refers to. If act is null, this is all sigaction does. If act isn’t null, the action for the specified signal is set.

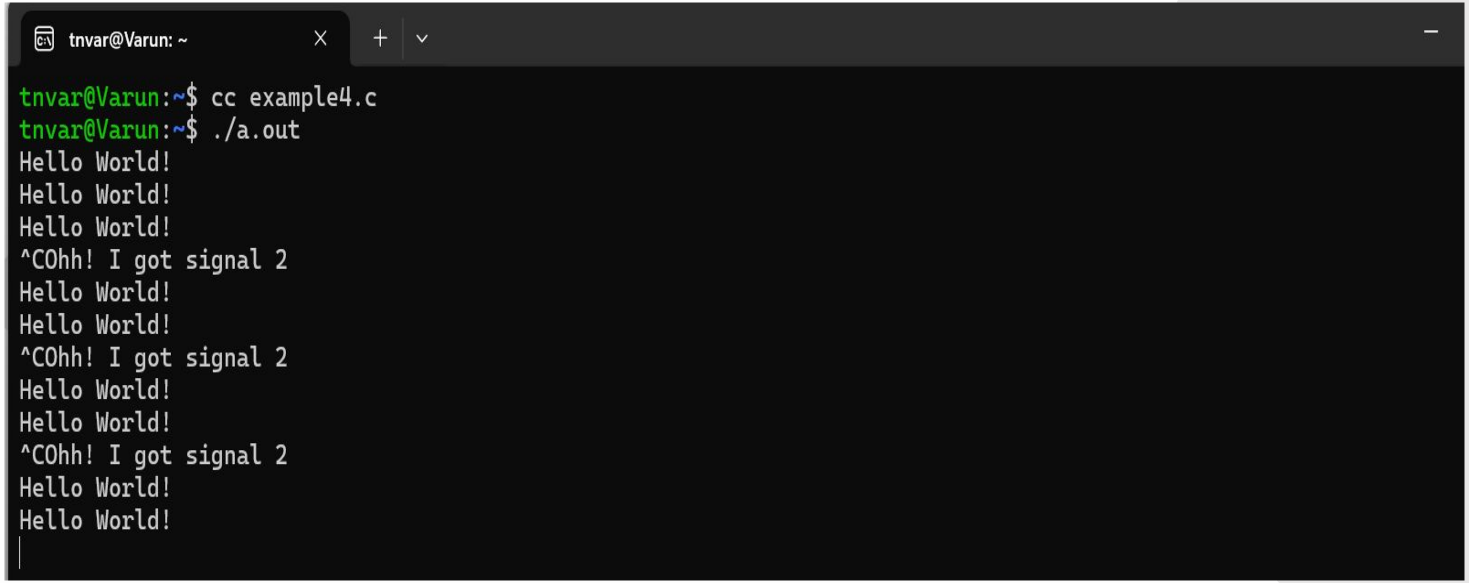
Sigaction Contd..

- As with signal , sigaction returns 0 if successful and -1 if not. The error variable errno will be set to EINVAL if the specified signal is invalid or if an attempt is made to catch or ignore a signal that can’t be caught or ignored.
- Within the sigaction structure pointed to by the argument act , sa_handler is a pointer to a function called when signal sig is received. This is much like the function func you saw earlier passed to signal .
- You can use the special values SIG_IGN and SIG_DFL in the sa_handler field to indicate that the signal is to be ignored or the action is to be restored to its default, respectively.

Example :

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void ohh(int sig) {
    printf("Ohh! I got signal %d\n", sig);
}

int main() {
    struct sigaction act;
    act.sa_handler = ohh;     //Calls the ohh( ) function when SIGINT occurs.
    sigemptyset(&act.sa_mask);     //No signals are blocked while handling SIGINT.
    act.sa_flags = 0;   //No special behavior.
   //Registering the Signal Handler
    sigaction(SIGINT, &act, NULL);  //NULL means we don’t store the previous handler.

    while (1) {
        printf("Hello World!\n");
        sleep(1);
    }

    return 0;
}
```

OUTPUT



tnvar@Varun: ~

```
tnvar@Varun:~$ cc example4.c
tnvar@Varun:~$ ./a.out
Hello World!
Hello World!
Hello World!
^COhh! I got signal 2
Hello World!
Hello World!
^COhh! I got signal 2
Hello World!
Hello World!
^COhh! I got signal 2
Hello World!
Hello World!
```

Thank You :)