



GLOBAL **EDGE**
Intelligence Of Things®

Signals

Abhijeet W

- What is signal ?
 - What is default action of signal ?
 - Data structure associated with signal
 - List of signals
- Signal Disposition
 - Signal system call & its drawback.
 - How to send signals via keystroke, command, function , system call ?
 - Sigaction system call & how to use it ?
- Sigevent

Contents (cnt ...)

- Some examples
- Summary
- Reference

What is signal ?

- A signal is a notification to a process that an event has occurred.
- Signals are two types
 - Traditional Signals or Standard Signals (1 to 31)
 - Real Time Signals

What is default action of signal ?

Default Action	Description
Term	Terminate process
Ing	Ignore signal
Core	Terminate process and dump core
Stop	Stop the process
Cont	Continue the process if process currently stopped

Data structure associated with signal

- Basic data structure used to store the signal sent to a process is a `sigset_t` array of bits.

```
typedef struct {  
    unsigned long sig [2];  
}sigset_t;
```

- Maximum number of signal that may be declared in linux is 64.

List of standard signals

Table 9-1. The First 31 Signals in Linux/i386

#	Signal Name	Default Action	Comment	POSIX
1	SIGHUP	Abort	Hangup of controlling terminal or process	Yes
2	SIGINT	Abort	Interrupt from keyboard	Yes
3	SIGQUIT	Dump	Quit from keyboard	Yes
4	SIGILL	Dump	Illegal instruction	Yes
5	SIGTRAP	Dump	Breakpoint for debugging	No
6	SIGABRT	Dump	Abnormal termination	Yes
6	SIGIOT	Dump	Equivalent to SIGABRT	No
7	SIGBUS	Abort	Bus error	No
8	SIGFPE	Dump	Floating point exception	Yes
9	SIGKILL	Abort	Forced process termination	Yes
10	SIGUSR1	Abort	Available to processes	Yes
11	SIGSEGV	Dump	Invalid memory reference	Yes
12	SIGUSR2	Abort	Available to processes	Yes
13	SIGPIPE	Abort	Write to pipe with no readers	Yes
14	SIGALRM	Abort	Real timer clock	Yes
15	SIGTERM	Abort	Process termination	Yes
16	SIGSTKFLT	Abort	Coprocessor stack error	No
17	SIGCHLD	Ignore	Child process stopped or terminated	Yes
18	SIGCONT	Continue	Resume execution, if stopped	Yes
19	SIGSTOP	Stop	Stop process execution	Yes
20	SIGTSTP	Stop	Stop process issued from tty	Yes
21	SIGTTIN	Stop	Background process requires input	Yes
22	SIGTTOU	Stop	Background process requires output	Yes
23	SIGURG	Ignore	Urgent condition on socket	No
24	SIGXCPU	Abort	CPU time limit exceeded	No
25	SIGXFSZ	Abort	File size limit exceeded	No
26	SIGVTALRM	Abort	Virtual timer clock	No
27	SIGPROF	Abort	Profile timer clock	No
28	SIGWINCH	Ignore	Window resizing	No
29	SIGIO	Abort	I/O now possible	No
29	SIGPOLL	Abort	Equivalent to SIGIO	No
30	SIGPWR	Abort	Power supply failure	No
31	SIGUNUSED	Abort	Not used	No

Signal Dispositions

- UNIX systems provide two ways of changing the disposition of a signal: `signal()` and `sigaction()`.
- Exception : A program can't handle signals of type SIGKILL, SIGSTOP.

Signal system call

- Signal () :-

ANSI C signal handling.

- Synopsis :-

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

```
typedef void (*sighandler_t )(int);
```

```
sighandler_t signal(int signum, sighandler_t handler);
```

- Return Value :-

On success signal() returns previous value of signal handler.

On error SIG_ERR or EINVAL invalid signum.

Description

- The `signal()` system call a new signal handler for the signal with number `signum`.
- The signal handler is set to `SIGHandler` which may be a user specified function, or either `SIG_IGN` or `SIG_DFL`.

Drawback of signal system call

- Signal function change the new action then return the previous value.
- When SIGhandler is executing due to one signal but suddenly process get another signal comes kernel delivering another signal.
- Using signal function not possible to inform kernel block other signal when SIGhandler is executing due to one signal.

Sending signal via signal

- **SIGINT : (ctrl-c)**
 - Signal number : 2
 - Default action terminate the process.
- **SIGTSTP : (ctrl-z)**
 - Signal number : 20
 - Default action terminate process.
- **SIGQUIT : (ctrl-\\)**
 - Signal Number : 3
 - Default action terminated the process and dump core.

Sending signal via commands

- Kill :-

Send a signal to process.

- Synopsis :

`kill -[options] <pid> [...]`

- Use -l or -L to list available signals. Particularly useful signals include HUP, INT, KILL, STOP, CONT, and 0.

- Example :

`kill -SIGINT 1234`

Sending signal via function call

- Function raise() :-

raise – send signal to caller.

- Synopsis :-

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

```
int raise(int sig);
```

- Return Value :-

raise() returns zero on success and nonzero on failure.

Sending signal via function call example

abhijeet@abhijeet: ~/b86_gw.abhijeet/Linux/Experiments/Presentation

abhijeet@abhijeet: ~/b86_gw.abhijeet/Linux/Experiments/Presentation 81x42

```
1  /*
2  *  How to use raise().
3  */
4
5  #include <stdio.h>
6  #include <signal.h>
7
8  void INThandler(int signum)
9  {
10     printf("\nExecuting SIGINT signal.\n\n");
11     return;
12 }
13
14 int main()
15 {
16     if(signal(SIGINT, SIG_IGN) != SIG_IGN)
17         signal(SIGINT, INThandler);
18     printf("In main() function.\n");
19
20     raise(SIGINT);
21     return 0;
22 }
```

"raise.c" 22L, 330C

1,1

All

abhijeet@abhijeet: ~/b86_gw.abhijeet/Linux/Experiments/Presentation 80x42

```
abhijeet@abhijeet:~/b86_gw.abhijeet/Linux/Experiments/Presentation$ gcc -o send_
sig raise.c
abhijeet@abhijeet:~/b86_gw.abhijeet/Linux/Experiments/Presentation$ ./send_sig
In main() function.
```

Executing SIGINT signal.

abhijeet@abhijeet:~/b86_gw.abhijeet/Linux/Experiments/Presentation\$

Sending signal via system call

- System call kill() :-

Send signal to process.

- Synopsis :-

```
#include <sys/types.h>
```

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

```
int kill (pid_t pid, int sig);
```

- Return Value :-

On success (at list one signal was sent) return zero.

On failure -1 returned , and errno is set appropriately.

Sending signal via system call example

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42

```
1 /*  
2  * How to use kill().  
3  */  
4  
5 #include <stdio.h>  
6 #include <signal.h>  
7  
8 void INThandler(int sigum)  
9 {  
10     printf("\nExecuting SIGINT signal.\n\n");  
11     return;  
12 }  
13  
14 int main()  
15 {  
16     int Pid;  
17     struct sigaction var;  
18  
19     var.sa_handler = INThandler;  
20     sigemptyset(&var.sa_mask);  
21     var.sa_flags = 0;  
22  
23     sigaction(SIGINT, &var, 0);  
24  
25     Pid = getpid();  
26  
27     printf("In main() function.\n");  
28  
29     kill(Pid, SIGINT);  
30     return 0;  
31 }
```

"kill.c" 31L, 435C

1,1

All

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ gcc -o kill_  
sig kill.c  
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./kill_sig  
In main() function.
```

Executing SIGINT signal.

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
```

Sigation system call

- Sigation() :-

Examine and change a signal action.

- Synopsis :-

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

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

- The sigaction() system call is used to change the action taken by a process, on receiving of specific signal.

Sigation system call (cnt..)

- Sigation structure members :-

```
struct sigaction {  
    void (*sa_handler) (int);  
    void (*sa_sigaction) (int, siginfo_t *, void*);  
    sigset_t sa_mask;  
    int sa_flags;  
    void (*sa_restorer) (void);  
};
```

- Return value :-

sigaction() returns 0 on success ,on error returns -1.

How to use sigaction()

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42

```
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <signal.h>
9
10 void INThandler (int);
11
12 void main (void)
13 {
14     struct sigaction var;
15
16     var.sa_handler = INThandler;
17     sigemptyset(&var.sa_mask);
18     var.sa_flags = 0;
19
20     sigaction(SIGINT, &var, 0);
21
22     /* if(signal(SIGINT, SIG_IGN) != SIG_IGN)
23        signal(SIGINT, INThandler); */
24     printf("In main() waiting for signal SIGINT\n");
25     while(1);
26     pause();
27 }
28
29 void INThandler (int signum)
30 {
31     char ch;
32     signal(signum, SIG_IGN);
33     printf("\nDid you hit ctrl-c ?\n");
34     printf("Do you really want to quit[Y/N].\n");
35     ch = getchar();
36     getchar();
37     if(ch == 'Y' || ch == 'y')
38         exit(0);
39     else{
40         signal(SIGINT, INThandler);
41         printf("In INThandler() waiting for signal SIGINT\n");
42     }
43 }
44
45
```

45,0-1

Bot

7:47 PM

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ gcc -o easy_
use easy_sigaction.c
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./easy_use
In main() waiting for signal SIGINT
^C
Did you hit ctrl-c ?
Do you really want to quit[Y/N].
N
In INThandler() waiting for signal SIGINT
^C
Did you hit ctrl-c ?
Do you really want to quit[Y/N].
n
In INThandler() waiting for signal SIGINT
^C
Did you hit ctrl-c ?
Do you really want to quit[Y/N].
Y
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./easy_use
In main() waiting for signal SIGINT
^C
Did you hit ctrl-c ?
Do you really want to quit[Y/N].
n
In INThandler() waiting for signal SIGINT
^C
Did you hit ctrl-c ?
Do you really want to quit[Y/N].
y
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
```

How to use sigaction() (cnt ...)

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

```
abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42
2  * How to use sigaction..
3  */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <signal.h>
8
9  void SIGhandler (int);
10
11 void main (void)
12 {
13     struct sigaction var;
14     //struct sigaction var1;
15
16     var.sa_handler = SIGhandler;
17     sigemptyset(&var.sa_mask);
18     var.sa_flags = 0;
19
20     sigaction(SIGALRM, &var, 0);
21     sigaction(SIGQUIT, &var, 0);
22
23     alarm(2);
24
25     printf("In main() waiting for signal SIGINT or SIGQUIT\n");
26     while(1);
27     pause();
28 }
29
30 void SIGhandler (int signum)
31 {
32     switch(signum){
33         case SIGALRM:
34             printf("\nExecuting SIGALRM signal handler.\n");
35             alarm(1);
36             break;
37         case SIGQUIT:
38             printf("\nYou hit ctrl-\\ \n");
39             printf("Executing SIGQUIT signal handler.\n");
40             break;
41     }
42 }
```

42,1

Bot

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./sigaction
In main() waiting for signal SIGINT or SIGQUIT

Executing SIGALRM signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.

Executing SIGALRM signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.

Executing SIGALRM signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.

Executing SIGALRM signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.

Executing SIGALRM signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
^\\
You hit ctrl-\\
Executing SIGQUIT signal handler.
```

Sigevent

■ Sigevent :-

Structure for notification from asynchronous routines.

■ Synopsis :-

```
union signal {           /* Data passed with notification */
    int    sival_int;     /* Integer value */
    void   *sival_ptr;    /* Pointer value */
};

struct sigevent {
    int      sigev_notify; /* Notification method */
    int      sigev_signo;  /* Notification signal */
    union signal sigev_value; /* Data passed with
                               notification */
    void      (*sigev_notify_function)(union signal);
                               /* Function used for thread
                               notification (SIGEV_THREAD) */
    void      *sigev_notify_attributes;
                               /* Attributes for notification thread
                               (SIGEV_THREAD) */
    pid_t     sigev_notify_thread_id;
                               /* ID of thread to signal (SIGEV_THREAD_ID) */
};
```

Sigevent (cnt ...)

- The sigevent structure is used by various APIs to describe the way a process is to be notified about an event. (e.g. Expiration of a timer, or the arrival of a message.)

Sigevent Example

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42

```
1 #include <stdlib.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <signal.h>
5 #include <time.h>
6
7 void SIGhandler (int signum)
8 {
9     printf ("SIGhandler caught..\n");
10    signal (signum, SIGhandler);
11 }
12
13 int main ()
14 {
15     int signo;
16     struct sigevent sevp;
17     sigset_t set;
18
19     if (sigemptyset (&set) == -1)
20         perror ("sigemptyset");
21     if (sigaddset (&set, SIGRTMIN) == -1)
22         perror ("sigaddset");
23     if (sigprocmask (SIG_BLOCK, &set, NULL) == -1)
24         perror ("sigprocmask");
25
26     sevp.sigev_notify = SIGEV_THREAD;
27     sevp.sigev_signo = SIGRTMIN;
28     sevp.sigev_value.sival_ptr = NULL;
29
30     kill (0, SIGRTMIN);
31     if (sigwait (&set, &signo) == 0)
32         SIGhandler (signo);
33     else
34         perror ("sigwait");
35 }
```

"sigevent.c" 35L, 750C

1,1

All

8:32 AM

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation\$ gcc -o event
sigevent.c

abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation\$./event

SIGhandler caught..

abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation\$

How to handle SIGCHLD signal

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42

```
2  /* When child process becomes zombie parent process signaled
3  * with SIGCHLD signal.
4  */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <signal.h>
10
11 void CHLDhandler(int signum);
12
13 int main()
14 {
15     switch(fork()){
16         case 0:
17             printf("In child process\n");
18             sleep(10);
19             printf("Child process execution done.\n");
20             break;
21         case -1:
22             printf("fork() fails.\n");
23             exit(0);
24         default:
25             printf("In parent process.\n");
26             signal(SIGCHLD, CHLDhandler);
27             while(1);
28     }
29 }
30
31 void CHLDhandler(int signum)
32 {
33     wait(0);
34     printf("Executing CHLDhandler()\n");
35 }
36
```

All

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ gcc sigchild
.c -o chld
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./chld
In parent process.
In child process
```

1

How to handle SIGCHLD signal (cnt...)

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 81x42

```
2  /* When child process becomes zombie parent process signaled
3  * with SIGCHLD signal.
4  */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <unistd.h>
9  #include <signal.h>
10
11 void CHLDhandler(int signum);
12
13 int main()
14 {
15     switch(fork()){
16         case 0:
17             printf("In child process\n");
18             sleep(10);
19             printf("Child process execution done.\n");
20             break;
21         case -1:
22             printf("fork() fails.\n");
23             exit(0);
24         default:
25             printf("In parent process.\n");
26             signal(SIGCHLD, CHLDhandler);
27             while(1);
28     }
29 }
30
31 void CHLDhandler(int signum)
32 {
33     wait(0);
34     printf("Executing CHLDhandler()\n");
35 }
36
```

,1

All

1

7:14 PM

abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42

```
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ gcc sigchild
.c -o chld
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./chld
In parent process.
In child process.
Child process execution done.
Executing CHLDhandler()
```

How to handle SIGSEGV signal.

```
abhiijeet@abhijee: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation
1 /*
2 * How to catch SIGSEGV signal..
3 */
4
5 #include <stdio.h>
6 #include <signal.h>
7 #include <stdlib.h>
8 #include <pthread.h>
9
10 void SEGVhandler (int sigum)
11 {
12     printf("Executing SEGVhandler\n");
13     return;
14 }
15
16 void * thread1 (void *ptr1)
17 {
18     if(signal(SIGSEGV, SIG_IGN) != SIG_IGN)
19         signal(SIGSEGV, SEGVhandler);
20
21     printf("Executing sub thread1\n");
22
23     int *ptr = (int *) 200;
24     printf("Pointer Value = %d\n", *ptr);
25     pause();
26 }
27
28 int main()
29 {
30     int status;
31     pthread_t first_id;
32
33     pthread_create(&first_id, NULL, thread1, NULL);
34     printf("In main thread.\n");
35
36     pthread_exit(NULL);
37
38     return 0;
39 }
~
"infir_sigsegr.c" 39L, 634C
```

How to handle SIGSEGV (cnt...)

```
abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation
1 /*
2  * How to catch SIGSEGV signal..
3  */
4
5 #include <stdio.h>
6 #include <signal.h>
7 #include <stdlib.h>
8 #include <pthread.h>
9
10 void SEGVhandler (int sigum)
11 {
12     printf("Executing SEGVhandler\n");
13     exit(0);
14     return;
15 }
16
17 void * thread1 (void *ptr1)
18 {
19     if(signal(SIGSEGV, SIG_IGN) != SIG_IGN)
20         signal(SIGSEGV, SEGVhandler);
21
22     printf("Executing sub thread1\n");
23
24     int *ptr = (int *) 200;
25     printf("Pointer Value = %d\n", *ptr);
26     pause();
27 }
28
29 int main()
30 {
31     int status;
32     pthread_t first_id;
33
34     pthread_create(&first_id, NULL, thread1, NULL);
35     printf("In main thread.\n");
36
37     pthread_exit(NULL);
38     return 0;
39 }
```

"sigsegv.c" 39L, 646C

1,1 All

```
abhiijeet@abhiijeet: ~/b86_gw.abhiijeet/Linux/Experiments/Presentation 80x42
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ gcc -pthread
-o segv_fault sigsegv.c
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$ ./segv_fault

In main thread.
Executing sub thread1
Executing SEGVhandler
abhiijeet@abhiijeet:~/b86_gw.abhiijeet/Linux/Experiments/Presentation$
```

Summary

In this presentation we discuss about basics of signal.

- What is signal , list of signals, default action of signals.
- Signal handling using `signal()` and `sigaction()`.
- How create a event using `sigevent`.

References

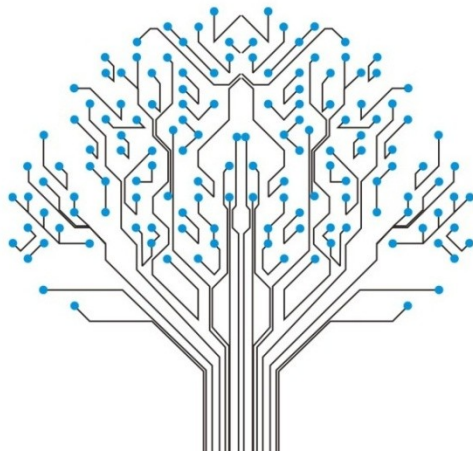
- The Linux Programming Interface.

By Michael Kerrisk.

- Understanding the Linux Kernel.

By Daniel P. Bovet & Marco Cesati.

Thank you



Fairness

Learning

Responsibility

Innovation

Respect