

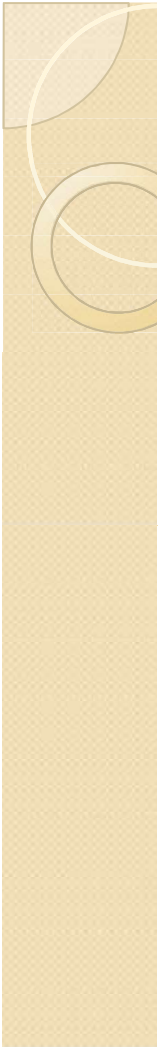


Signal Handling in Linux

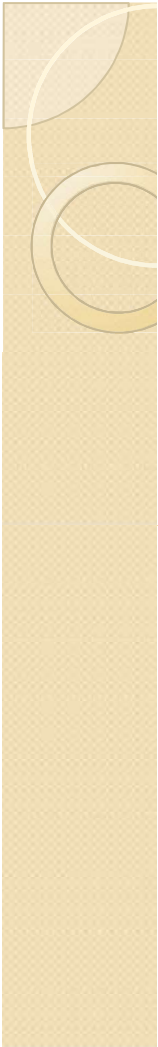
- Broad classification - synchronous and asynchronous
- A signal is an asynchronous event which is delivered to a process. •
- Asynchronous means that the event can occur at any time may be unrelated to the execution of the process. •
- Signals are raised by some error conditions, such as memory segment violations, floating point processor errors, or illegal instructions. – e.g. user types ctrl-C, or the modem hangs

Signal Handling in Linux

- A signal – an event which is generated to notify a process or thread that some important situation has risen
- a process or thread on receipt of a signal, will stop what its doing and take some action
- Can be seen as a form of inter-process communication.
- signals are defined in the header file *signal.h* as a macro constant
- Every signal has a name and an associated number. Advisable to track on Signal names to avoid changing numbers on different systems



Signal Name	Description
SIGHUP	Hang-up the process. The SIGHUP signal is used to report disconnection of the user's terminal, possibly because a remote connection is lost or hangs up.
SIGINT	Interrupt the process. When the user types the INTR character (normally Ctrl + C) the SIGINT signal is sent.
SIGQUIT	Quit the process. When the user types the QUIT character (normally Ctrl + \) the SIGQUIT signal is sent.
SIGILL	Illegal instruction. When an attempt is made to execute garbage or privileged instruction, the SIGILL signal is generated. Also, SIGILL can be generated when the stack overflows, or when the system has trouble running a signal handler.



SIGTRAP	Trace trap. A breakpoint instruction and other trap instruction will generate the SIGTRAP signal. The debugger uses this signal.
SIGABRT	Abort. The SIGABRT signal is generated when abort() function is called. This signal indicates an error that is detected by the program itself and reported by the abort() function call.
SIGFPE	Floating-point exception. When a fatal arithmetic error occurred the SIGFPE signal is generated.
SIGUSR1 and SIGUSR2	The signals SIGUSR1 and SIGUSR2 may be used as you wish. It is useful to write a signal handler for them in the program that receives the signal for simple inter-process communication.

Default Action Of Signals

- Each signal has a default action, one of the following:
- **Term:** The process will terminate.
- **Core:** The process will terminate and produce a core dump file.
- **Ign:** The process will ignore the signal.
- **Stop:** The process will stop.
- **Cont:** The process will continue from where it stopped.
- Default action may be changed using handler function. Some signal's default action cannot be changed.
- **SIGKILL** and **SIGABRT** signal's default action cannot be changed or ignored.

Signal Handling

- On Signal receipt, the process has a choice of action .
- The process can ignore the signal, can specify a handler function, or accept the default action for that kind of signal.
- If the specified action for the signal is ignored, then the signal is discarded immediately.
- The program can register a handler function using function such as ***signal*** or ***sigaction***. Called as handler catch the signal.
- If the signal has not been neither handled nor ignored, its default action takes place.
- We can handle the signal using ***signal*** or ***sigaction*** function.

Signal Handling

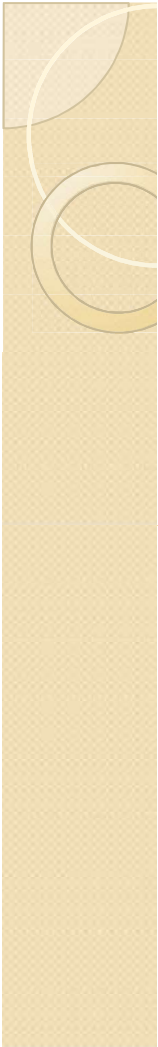
- **int signal () (int signum, void (*func)(int))**
- **signal()** will call the **func** function if the process receives a signal **signum**. **signal()** returns a pointer to function **func** if successful or it returns an error to **errno** and **-1** otherwise.
- The **func** pointer can have three values:
- **SIG_DFL**: It is a pointer to system default function **SIG_DFL()**, declared in **h** header file. Accounts for default action of the signal.
- **SIG_IGN**: It is a pointer to system ignore function **SIG_IGN()**, declared in **h** header file.
- **User defined handler function pointer**: The user defined handler function type is **void(*) (int)**, means return type is void and one argument of type int.

First Signal Handling Code

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

void sig_handler(int signum) {
    //Return type of the handler function should be void
    printf("\n Inside handler function\n");
}

int main(){
    signal(SIGINT,sig_handler); // Register signal handler
    for(int i=1;;i++){    //Infinite loop
        printf("%d : Inside main function\n",i);
        sleep(1); // Delay for 1 second ??
    } return 0;}
```

```
1 : Inside main function
2 : Inside main function
^C
Inside handler function
3 : Inside main function
4 : Inside main function
^\Quit (core dumped)
```

Signal Interrupt User Handler

```
1 : Inside main function
2 : Inside main function
^C3 : Inside main function
4 : Inside main function
^C5 : Inside main function
^\Quit (core dumped)
```

Signal Ignoring

Signal Ignoring

- ```
#include<stdio.h>
#include<unistd.h>
#include<signal.h>
int main(){
 signal(SIGINT,SIG_IGN); // Register signal handler for ignoring
the signal
 for(int i=1;;i++) { //Infinite loop
 printf("%d : Inside main function\n",i);
 sleep(1); // Delay for 1 second
 } return 0;}
```

## Signal Re-registration

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

void sig_handler(int signum){
    printf("\nInside handler function\n");
    signal(SIGINT,SIG_DFL); // Re Register signal handler for
    default action
}

int main(){
    signal(SIGINT,sig_handler); // Register signal handler
    for(int i=1;;i++){    //Infinite loop
        printf("%d : Inside main function\n",i);
        sleep(1); // Delay for 1 second
    } return 0; }
```

Signal Re-registration

- Notice in the above code on first SIGINT press it takes user handler action;
- Subsequent SIGINT input triggers call to default behaviour becoz of the re registration and terminates the program.

```
1 : Inside main function
2 : Inside main function
3 : Inside main function
^C
Inside handler function
4 : Inside main function
^C
```