Signal Handling

Dr. Dipali Meher

Source: The Design of Univ Operating System- Maurice Bach

Introduction

Signals are software generated interrupts that are sent to a process when a event happens.

Signals can be synchronously generated by an error in an application such as SIGEPE and SIGSEGV.

Most of the signals are asynchronous.

Signals are posted to a process when the system detects a software event such as user entering an interrupt or stop or a kill request from another process.

Some signals can also come directly from kernel when an hardware event such as a bus error or an illegal instruction is encountered.

signals delivery is analogous to hardware interrupts (such signals can be blocked in future)

Most of signals cause termination of a process when not attempted.

Each signal has a default action which is one of the following:

- The signal is discarded after being received
- The process is terminated after the signal is received
- A core file is written then the process is terminated
- Stop the process after the signal is received

Each signal defined by the system falls into one of five classes:

- Hardware conditions
- Software conditions
- Input/output notification
- Process control
- Resource control

Macros are defined in <signal.h> header file for common signals are

SIGHUP 1//hangup
SIGINT 2//interrupt
SIGQUIT 3 //quit
SIGILL 4 //illegal instruction
SIGABRT 6 //used by abort
SIGKILL 9 //hard kill
SIGALRM 14 //alaram clock
SIGCONT 19 //continue a stopped process
SIGCHLD 20 //to parent on child stop or exit
(signals are numbered from 0 to 31 . Above are some important)

Signal concepts

- Signals are classic example of asynchronous events.
- Every signal has same name that starts with name SIG
- These name are defined by positive integer constant in the header <signal.h>
- No any signal has number zero. A kill function uses the signal number zero for a special case.

Signals are generated for number of conditions such as

- The terminal generated signals occurs when user press certain terminal keys.
- Hardware exceptions generated signals occur when an invalid memory reference is made or divide by zero occurs
- Software conditions can generate signals when the process should be notified about certain event.
- The kill(1) command allow us to send signals to other processes, This command is often used to terminate a runaway background process.
- The kill(2) function allows a process to send any signal to other process or a process group.

Kernel performs following three actions when a signal occurs. This is known as disposition of the signal or the action associated with a signal.

A) Ignore the signal: Most of the signals are ignored except SIGKILL and SIGSTOP

B)Catch the Signal: A user defined function can be called by a kernel to take the necessary action

C) Let the default action apply: every signal has a default action which is carried out

When default action is labeled' terminate + core' it means that a memory image of the process is left in the file named 'care'. This file is used to examine the state of the process at the time it terminated.

The core file will not be generated if:

- The process was set user ID and the current user is not the owner of the program file.
- The process was set user id and the current user is not the group owner of the file.
- The user does not have the permission to write in the current working directory
- The file already exists and the user does not have permission to write to it.
- The file is too big
- The permission of the core file are usually user read and user write

Basic signal handling

An application program can specify a function called a signal handler to be invoked when a specific signal is received. When a signal handler is invoked on receipt of a signal, it is said to catch the signal. A process can deal with a signal in one of the following ways:

i) The process can let the default action happen

ii) The process can block the signal

iii)The process can catch the signal with a handler

Basic Signal Handling

Signal handlers usually execute on the current stack of the process. This lets the signal handler return to the point that execution was interrupted in the process.

This can be changes per signal basis so that a signal handler executes on a special stack. If a process must resume in a different context than the interrupted one it must restore the previous context itself.

The Signal function provides a simple interface for the establishing an action for a particular signal, The function and associated macros are declared in the handler file<signal.h>

data type: sighandler_t

This is type of signal handaler functions. Signal handlers take one integer argument specifying the signal number and have return type void so to define handler function like

void handler(int signum{….}

The name sighandler_t for this data type is GNU extension.

Function: sighandler_t signal(int signum,sighandler_t action)

The signal function establishes action as the action for the signal function.

The first argument, signum identifies the signal whose behaviour you want to control and should be a signal number. The proper way to specify a signal number is with one of the symbolic signal names don't use an explicit number, because the numerical code for a given kind of signal may vary from operating system to operating system.

The second argument action specifies the action to use for the signal *signum*.

List of Unix System signals

Name	Description	Default Action
SIGABRT	Abnormal termination(abort)	Terminate + core
SIGALRM	Timer expired(alarm)	Terminate
SIGBUS	Hardware fault	Terminate
SIGCONT	Continue stopped process	Terminate + code
SIGFREEZE	Checkpoint freeze	Continue/ignore
SIGKILL	Termination	Ignore
SIGSTOP	Stop	Terminate
SIGSYS	Invalid system call	Stop process
SIGINT	Terminal interrupt character	Terminate + core
SIGURG	Urgent condition(sockets)	Terminate
SIGUSR1	User defined signal	Ignore
SIGUSR2	User defined signal	Terminate
SIGINFO	Status request from keyboard	Terminate
SIGIO	Asynchronous I/O	Ignore
SIGPWR	Power fail/restart	Terminate/ignore
SIGQUIT	Terminal quit character	Terminate/ignore
SIGSEGV	Invalid memory references	Terminate + core
SIGTERM	Abnormal termination	Terminate + core
SIGTRAP	When PT_PTRACED flag is set the execve call sends a SIGTRAP signal to itself. On return from execve the SIGTRAP signal is processes the process is stopped and the parent process is informed by being sent a SIGCHLS signal	

Signal function

The simplest interface to use signal features of the UNIX system is the signal function.

#include<signal.h>

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

The signal function is defined and the semantic of the c function depends upon the version of C library.

The prototype of the signal function states that the function requires two arguments and returns a pointer to a function which returns void.

Signo→ It is an integer value specifying the signal name.

Simple program for signal handler

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
void sig_handler(int signo)
{
  if (signo == SIGINT)
    printf("received SIGINT\n");
}
int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR)
  printf("\ncan't catch SIGINT\n");
  // A long long wait so that we can easily issue a signal to this process
  while(1)
    sleep(1);
  return 0;
}
```

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
int main()
{
   if(signal(SIGUSR1, sig_usr)==SIG_ERR)
   err_sys("cannot catch SIGNUSR1");
   if(signal(SIGUSR2,sig_usr)==SIG_ERR)
   err_sys("cannot catch SIGNUSR2");
   return 0;
}
static void sig_usr(int signo)
{
   if(signo==SIGUSR1)
   printf("received SIGUSR1\n");
   else if(signo==SIGUSR2)
   printf("received siguser2");
   else err_sump("received signal%d",signo);
}
```

Sending signals

A process may send a signal to itself by calling raise(). A process may send a signal to another process, including itself by calling kill().

Int kil(int pid, int signal)

A system call that send a signal to a process pid. If pid is greater than zero the signal is sent to the process whose process ID is equal to pid. If pid is 0 the signal is sent to all processes except system processes.

Kill() returns 0 for success call, -1 otherwise and sets errno accordingly.

Int raise(int sig)

It sends the signal sig to the executing program raise() actually uses kill() to send the signal to the executing program.

Kill(getpid(),sing);

this is also a unix command called kill that can be used to send signals from the command line.

Basic Rule

Only processes that have the same user can send /receive messages.

The SIGKILL signal cannot caught or ignored and will always terminate a process.

For example kill(getpid(),SIGINT);

It would send the interrupt signal to the id of the calling process. This would have a similar effect to exit() command, Also ctrl-C type from the command sends a SIGINT to the process currently being.

unsigned int alaram(unsigned int seconds)

It sends the signal SIGALRM to the invoking process after seconds seconds.

When ta a process terminated abnormally it usually tries to send a signal indicating what went wrong.

C programs can trap these for diagnostics. Also user specified communication can take place in this way.

Unreliable signals

In earlier versions of UNIX signals are unreliable that means a signal may get lost. Also a process has a little control over a signal;a process could catch the signal or ignore it.

Problem with this is that: the action for a signal was reset to its default, each time the signal occurred. Another problem is that the process was unable to turn a signal off when it did not want to signal to occur.

To catch a signal and a flag to indicate the process about the signal occurrences

```
int sig_int_flag;main()
{    int sing_int();
 ...
 singla(SIGINT,sing_int);
    ....
 while(sing_intflag)==0)
   pause();}
sing_int()
{
 signal(SIGINT, sig_int);
  sing_int_flag=1;
}
```

Here the process is calling the pause function to put it to sleep until the signal is caught. Whtn the signal is caught the signal handler just sets the flag sig_int flag to a nonzero value
The process is automatically awakened by the kernel after the signal handler returns, noticess that the flag is non zero and does whatever it needs to do

Interrupted system calls

To differentiate between the system call and function it is the system call within the kernel that is interrupted when a signal is caught. The system calls are divided into two categories:

i) slow system calls

ii) other system calls

The slow system calls are those that can block forever, the system calls included in this category are:

A) Reads that can block the caller forever if data is not available with certain file types such as pipes terminal devices and network devices.

B) Writes that can block the caller for every if the data can't be accepted immediately by these same file types.

C) The pause function and the wait function
D) certain ioctl operations

E) some of the interprocess communication functions

F) Opens that block until some condition occurs on certain file types

The exception to these allow system calls is anything related to disk I/O, in interrupted system calls it is required to handle errors explicitly.

How interrupted system callas are differ from other system calls?

System Call	System Interrupt
A procedural method through which a computer software requests the operating system's kernel for assistance	An interrupt is an event in which the CPU is asked to do a specific action by outside components in an operating system
Enables an application to interact with the kernel in order to access resources, such as memory or hardware devices	Informs the CPU to pause running the currently executing programs in order to perform some urgent actions
A system call is initiated by the program calling the kernel by executing a special instruction	A system interrupt is initiated by hardware or software
Requires the kernel's attention	Don't require the kernel's attention

Reentrant Functions

When a process catches a signal the normal sequence of instructions being executed by the process, is temporarily interrupted by the signal handler.

The process then continues executing but the instructions in the signal handler are now executed.

If the signal handler returns then the normal sequence of instructions that the process executing when the signal was caught continues executing.

The signal UNIX specification specifies the functions that are guaranteed to be reentrant.

Reentrant functions(some functions)

accept	fchmofcnld	lseek	sendto	stat
access	fchown	lstat	sertid	symlink
alarm	fcntl	mkdir	setsid	tcflow
bind	fork	open	signal	time
chdir	fstat	pause	sigset	times
chmod	fsync	pipe	sleep	umask
close	ftruncate	poll	socket	uname
connect	getgid	read	seruid	unlink
dup	kill	rmdir	sigqueue	Utime
Dup 2	link	select	shutdown	wait

SIGCLD semantics

When a signal handler is established for SIGCHLD and there exists a terminated child we have not yet waited for, it is unspecified whether the signal is generated.

While processing a SIGCHLD signal the kernel checks whether a child needs to be waited for so it generates another call to signal handler.

The signal handler calls signal and whole process starts again.

kill() and raise() functions

The kill function sends a signal to a process to a group of processes where as the raise function allows a process to send a signal to itself.

#include<signal.h>

int kill(pid pid, int signo);

int raise(int signo);

The call raise(signo) in equivalent to the call kill(getpid(),signo) there are 4 different conditions for the pid argument to kill.

1	pid>0	The signal is sent to the process whose process ID is pid
2	pid ==0	The signal is sent to all processes whose group ID equals the process group ID of the sender and foe which the sender has permission to send signal
3	pid <0	The signal in sent to all processes whose process group ID equals the absolute value of pid and for which the sender has permission to send the signal.
4	pid ==-1	The signal is sent to all or processes on the system for which the sender has the permission to send the signal as before the set of processes excluders certain system processes.

alarm() and pause() functions

The alarm function allows us to set a timer that will expire that specified time in the future. When the timer expires, the SIGALRM signal is generated.

#include<unistd.h>

unsigned int alarm(unsigned int seconds);

The seconds value is the number of clock seconds in the future when the signal could be generated. There is only one of these alarm clocks per processes although the default action for SIGALRM is to terminate the process, most processes that use an alarm clock catch this signal. If the process then wants terminate it can perform whatever clean

up is required before terminating.

The pause() function suspends the calling process until a signal is caught,

include<unistd.h>

int pause(void)

The only time pause returns is if a signal handler is executed and that handler returns. In that case pause() returns -1 with errno ser too EINTR.

Signal sets

Signal set is a data type used to represent multiple signals. The function sigfillset initializes the signal ser so that all signals are included.

The signal ser operations manage these signal sets:

i) sigempty set() It initializes the signal ser given by set, marking it empty(all signals excluded from set)

ii)sigfillset(): It initializes the signal set, given by the ser marking it full (all signals included in the set)

iii) Sigaddset(): It adds signo to the signal ser given by the set.

iv)sigdelset(): It removes signo from the signal ser given by set.

v)sigismember(): returns -1 if signo is in the signal set given by the ser, 0 if it is not and -1 on error.

```
#include<signal.h>
int sigemptyset(sigset_t,*set);
int sigfillset(sigset_t,*set);
int sigaddset(segset_t *set, int signo);
int sigselset(sigset_t *set, int signo);
int sigismember(const sigset_t *set, int signo);
```

Retrieving Pending signals

The sigpending() function returns the set of set of signals that are blocked from delivery and currently pending for the calling process. The set of signals is returned through the set argument.

#include<signal.h?

int sigpending(sigset_t *set)

When a pending signal is unblocked, the kernel then passes signal off to the process to handle.

sigaction() function

It allows to modify or examiner the action associated with a particular signal.

#include<signal.h>

int sigaction(int signo, const struct sigaction * restrict act, struct sigaction * restrict oact);

POSIX standardizes the sigaction() system call which provides much greater signal management capabilities.

A call to sigaction() changes the behaviour of the signal identified by signo which can be any value except these associated with SIGKILL and SIGSTOP.

The argument signo is the signal number whose action we are examining or modifying , If the oact pointer is non null we are modifying the action.

If the oact pointer is non numm the system returns the previous action for the signal through the oact pointer.

Nonlocal Branching

sigsetjmp() and siglongjmp() functions are used for non local branching. The siglongjump() is often called from a signal handler to return to the main loop of program instead of returned from the handler.

#include<setjmp.h>

int sigsetjmp(sigjmp_buf env, int savemask);

void siglongjmp(sigjmp_buf env, int val);

Some more Functions

i)sigsuspend() function: Sigsuspend resets the signal mask as well as puts the process to sleep in a single atomic operation.

#include<signal.h>

int sigsuspend(const sigset_t *sigmask)

The signal mask of the process I set to the value pointer to by sigamsk, Then the process is suspended until a signal is caught or until a signal occurs that terminated the process.

If a signal is caught and if the signal handler returns then sigsuspend returns and the signal mask of the process is set to its value before the call to sigsuspend.

Some more Functions

i)abort() function: It results in abnormal program termination. This function sends the SIGABRT signal to the caller processes. When the process catches the SIGABRT signal, It allows the SIGABRT to perform the cleanup operation before the process terminates

#include<stdlib.h>

void abort(void);

ii) system() function: The purpose of a system function is to handle signals. When it is required to include a system function in a program It is required to interpret the correct return value as it is the termination status of the shell which is not always the termination status of the command string.

Some more Functions

i)sleep() function: sleep function causes the calling process to be suspended until either the amount of awall clock time specified by seconds has elapsed or a signal is caught by the process and the signal handler returns.

Vii) Job control signals: there are various job control signals such as:

A. SIGCHLS: child process has stopped/terminated

B. SIGCONT: Continue process if stopped.

C. SIGSTOP: stop signal (cannot be caught/ignored)

D. SIGSTP: interactive stop signal

E. SIGTTIN: read from controlling terminal by member of a background process group

F. SIGTTOU: write to controlling terminal by member of a background process group.

Advanced Signal Management

The signal function is very basic. Because it is part of the standard C library and therefore has to reflect minimal assumptions about the capabilities of the operating system in which it runs, it can offer only a lowest common denominator to signal management.

POSIX standardizes the sigaction() system call, which provides much greater signal management capabilities, Among other things it is used to block the reception of specified signal while your handler runs, and to retrieve a wide range of data bout the system and process state at the movement a signal was raised.

#inclide<signal.h>

Int sigaction(int signo, const struct sigaction *act, struct sigaction *oldact)

A call to sigaction() changes the behaviour of th signal identified by signo, which can be any value except those associated with SIGKILL and SIGSTOP. If act is not NULL the system call changes the current behaviour of the signal as specified by ac.t If oldact is not NULL the call stores the previous behaviour of the given signal there.

The sa_handler field dictates the action to take upon receiving the signal As with signal this field may be SIF_DFL signifying default action SIG_IGN instructing the kernel to ignore the signal for process, or a pointer toa signal handling function. This function has the same prototype as a signal handler installed by signal.

void my_handler(int signo)

Sa flag values

SA_NOCLDSTO: instructs the system to not provide notification when a child process stops or resumes.

SA_NOCLDWAIT: this flag enables automatic child reaping: children are not converted to zombies on termination and the parent need not call wait()

SA_NOMASK: this flag an obsolete non-POSIX equivalent to SA_NODEFER Use NODEFER instead of this flag but be prepared to see this value turn up older code.

SA_ONESHOT: This flag is an obsolete non-POSIX equivalent to SA_RESETHAND use SA_RESETHAN instead of this flag.

SA_ONSTACK: This flag instructus the system to invoke the given signal handler on an alternative sognal stack as provided by sinalstack(). Alternative signal stacks are rare although they are useful in some pthreads applications with smaller thread stacks that might be overrun by some signal handler usage.

SA_RESTART: This flag enables BSD-style restarting of system calls that are interrupted signals.

SA_RESETHAND: This flag enables “one shot” mode. The behaviour of the given signal is reset to the default once the signal handler returns. Sigaction() returns 0 on success. On failure the call returns -1 an sets errno to one of the following error codes:

EFAULT: act or invalid pointer

EINVAL: signo is an invalid signal SIGKILL or SIGSTOP.

Sending a signal with a payload

Signal handlers registered with eh SA_SIGINFO flag are passes a siginfo_t parameter, This structure contains a field named si-value which is an optional payload passed form the signal generator to the signal receiver.

A function known as sigqueue() defined by PPSIX allows a process to send a signal with this payload.

#include<signal.h>

int sigqueue(pid_t pid, int signo, const union signal value)

Here sigqueue function works same as kill. The signals payload is given by value which is an union of an integer value and a void pointer described as follows.

union sigval

{ int sival_int;

coid (sival_ptr;

};