Signal Management

Advanced Embedded Linux Development

with **Dan Walkes**

University of Colorado **Boulder**

Learning objectives:

Writing Signal Handlers

Signal Handlers and Reentrancy

Signal Management

```
SIGNAL(2)                                                Linux Programmer's Manual

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);

DESCRIPTION
       The behavior of signal() varies across UNIX versions, and has also varied historically across different versions of Linux.  Avoid its use: use sigaction(2) instead.
```

```
SIGACTION(2)

NAME
       sigaction, rt_sigaction - examine and change a signal action

SYNOPSIS
       #include <signal.h>

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

Linux System Programming Chapter 10

http://man7.org/linux/man-pages/man2/signal.2.html

http://man7.org/linux/man-pages/man2/sigaction.2.html

sigaction signal handling

- POSIX alternative (not part of the original std C library)
- Better signal management capabilities

```
NAME
       sigaction, rt_sigaction - examine and change a signal action

SYNOPSIS
       #include <signal.h>

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

The sigaction structure is defined as something like:

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

Linux System Programming Chapter 10
http://man7.org/linux/man-pages/man2/sigaction.2.html

sigaction signal handling

- Better signal management capabilities
  - Block new signals during handler
  - Retrieve state information
  - Only need to setup sa_handler in sigaction in simple cases

```
NAME
       sigaction, rt_sigaction - examine and change a signal action

SYNOPSIS
       #include <signal.h>

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

The sigaction structure is defined as something like:

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

Linux System Programming Chapter 10

http://man7.org/linux/man-pages/man2/sigaction.2.html

Waiting for signals

- pause() waits for a signal

```
PAUSE(2)                    Linux Programmer's Manual                    PAUSE(2)

NAME
       pause - wait for signal

SYNOPSIS
       #include <unistd.h>

       int pause(void);

DESCRIPTION
       pause() causes the calling process (or thread) to sleep until a signal is
       delivered that either terminates the process or causes the invocation  of
       a signal-catching function.
```

Sending signals

- kill() sends a signal to a process
  - Sends any signal, not just SIGKILL
- Available as system call or user command

```
KILL(1)

NAME
       kill - send a signal to a process

SYNOPSIS
       kill [options] <pid> [...]

 -<signal>
 -s <signal>
 --signal <signal>
       Specify the signal to be sent. The signal can be specified
       by using name or number. The behavior of signals is
       explained in signal(7) manual page.
```

```
KILL(2)

NAME
       kill - send signal to a process

SYNOPSIS
       #include <sys/types.h>
       #include <signal.h>

       int kill(pid_t pid, int sig);


NAME
       sigqueue - queue a signal and data to a process

SYNOPSIS
       #include <signal.h>

       int sigqueue(pid_t pid, int sig, const union sigval value);
```

Linux System Programming Chapter 10

http://man7.org/linux/man-pages/man2/kill.2.html

http://man7.org/linux/man-pages/man1/kill.1.html

http://man7.org/linux/man-pages/man3/sigqueue.3.html

sigaction signal handling

```c
int main ( int argc, char **argv )
{
    struct sigaction new_action;
    bool success = true;
    memset(&new_action,0,sizeof(struct sigaction));
    new_action.sa_handler=signal_handler;
    if( sigaction(SIGTERM, &new_action, NULL) != 0 ) {
        printf("Error %d (%s) registering for SIGTERM",errno,strerror(errno))
        success = false;
    }
    if( sigaction(SIGINT, &new_action, NULL) ) {
        printf("Error %d (%s) registering for SIGINT",errno,strerror(errno));
        success = false;
    }
```

```c
bool caught_sigint = false;
bool caught_sigterm = false;

static void signal_handler ( int signal_number )
{
    if ( signal_number == SIGINT ) {
        caught_sigint = true;
    } else if ( signal_number == SIGTERM ) {
        caught_sigterm = true;
    }
}
```

```c
if( success ) {
    printf("Waiting forever for a signal\n");
    pause();
    if( caught_sigint ) {
        printf("\nCaught SIGINT!\n");
    }
    if( caught_sigterm ) {
        printf("\nCaught SIGTERM!\n");
    }
}
```

Linux System Programming Chapter 10

https://github.com/cu-ecen-5013/aesd-lectures/blob/master/lecture9/signal_handler.c

signal handling

```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture9$ ./signal_handler
Waiting forever for a signal
^C
Caught SIGINT!
```

```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture9$ ./signal_handler
Waiting forever for a signal
```

```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture9$ ps aux | grep signal_handler
aesd       31236  103  0.0   4504   732 pts/1    R+   14:07   0:16 ./signal_handler
aesd       31246  0.0  0.0  21532   996 pts/8    S+   14:07   0:00 grep --color=auto signal_handler
```

```
aesd@aesd-VirtualBox:~/aesd-lectures/lecture9$ kill -TERM 31236
```

```
Caught SIGTERM!
aesd@aesd-VirtualBox:~/aesd-lectures/lecture9$
```

- Ctrl->C sends SIGINT
- Restart
- Find process PID
- Send TERM
- Process exits with TERM

Linux System Programming Chapter 10

https://github.com/cu-ecen-5013/aesd-lectures/blob/master/lecture9/signal_handler.c

Sending signals

- sigqueue() can send a payload with a signal
  - Rudimentary form of IPC

```
NAME
       sigqueue - queue a signal and data to a process

SYNOPSIS
       #include <signal.h>

       int sigqueue(pid_t pid, int sig, const union sigval value);

     union sigval {
         int   sival_int;
         void *sival_ptr;
     };

  If the receiving process has installed a handler for this signal using
  the SA_SIGINFO flag to sigaction(2), then it can obtain this data via the
  si_value field of the siginfo_t structure passed as the second argument
  to the handler.  Furthermore, the si_code field of that structure will be
  set to SI_QUEUE.
```

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

siginfo_t {
    int      si_signo;     /* Signal number */
    int      si_errno;     /* An errno value */
    int      si_code;      /* Signal code */

    clock_t  si_utime;     /* User time consumed */
    clock_t  si_stime;     /* System time consumed */
    sigval_t si_value;     /* Signal value */
    int      si_int;       /* POSIX.1b signal */
    void    *si_ptr;       /* POSIX.1b signal */
```

Linux System Programming Chapter 10

http://man7.org/linux/man-pages/man3/sigqueue.3.html

http://man7.org/linux/man-pages/man2/sigaction.2.html

Additional Signal Concepts

- Child inherits signal actions of the parent on fork
- Starting a processes with exec() resets all signals to default actions (other than those ignored by the parent).

Signal Handlers and Safety

- Signal handlers:
  - Suspend execution of a process
  - Jump to our signal handler function
  - Not a thread switch or new thread - re-uses the existing thread

Linux System Programming Chapter 10
Advanced Programming in the UNIX environment section 10.6
IEEE Std 1003.1-2017 https://pubs.opengroup.org/onlinepubs/9699919799/

Signal Handlers and Safety

- What does this mean about the functions you can call in the signal handler?
  - Must be “async-signal-safe” in Single Unix Specification parlance.
  - Must be reentrant *and* block signals during any points which could cause inconsistencies

Linux System Programming Chapter 10
Advanced Programming in the UNIX environment section 10.6
IEEE Std 1003.1-2017 https://pubs.opengroup.org/onlinepubs/9699919799/

Signal Handlers and Safety

- How do I know if a POSIX function I'm calling is signal safe?
  - Table of allowed functions - one in your book, more complete in link below
  - malloc, free, (or any function which may call these) stdio library functions (printf for instance) are generally not safe to call.

Linux System Programming Chapter 10
Advanced Programming in the UNIX environment section 10.6
IEEE Std 1003.1-2017 https://pubs.opengroup.org/onlinepubs/9699919799/functions/V2_chap02.html#tag_15_04_03

Signal Handlers and Reentrancy

- Must not manipulate static data also used outside the signal handler (strsignal() or similar functions)
- Should manipulate only stack-allocated data or data provided by the caller
  - Setting global flag is acceptable
- Must not invoke any non- async-signal-safe function

Signal Handler Best practices

- Minimize global data access
  - Confirm any access is async signal safe
- Save/restore errno
- Call an absolute minimum set of functions,
  - Call only functions on an approved list of signal handler functions.

Linux System Programming Chapter 10
Advanced Programming in the UNIX environment section 10.6
IEEE Std 1003.1-2017 https://pubs.opengroup.org/onlinepubs/9699919799/

Why Use/Not Use Signals?

- “old, antiquated mechanism for kernel to user communication”
- However, we are forced to use them for many important notifications (especially termination).
- Signal safety problems are easy to introduce and difficult to track down.
  - Handlers must be implemented carefully.