

Signals, Part 4: Sigaction

jakebailey edited this page on Dec 16, 2014 · 9 revisions

How and why do I use sigaction ?

To change the "signal disposition" of a process - i.e. what happens when a signal is delivered to your process - use `sigaction`

You can use system call `sigaction` to set the current handler for a signal or read the current signal handler for a particular signal.

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

The sigaction struct includes two callback functions (we will only look at the 'handler' version), a signal mask and a flags field -

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

How do I convert a signal call into the equivalent sigaction call?

Suppose you installed a signal handler for the alarm signal,

```
signal(SIGALRM, myhandler);
```

The equivalent `sigaction` code is:

```
struct sigaction sa;
sa.sa_handler = myhandler;
sigemptyset(&sa.sa_mask);
sa.sa_flags = 0;
sigaction(SIGINT, &sa, NULL)
```

However, we typically may also set the mask and the flags field. The mask is a temporary signal mask used during the signal handler execution. The SA_RESTART flag will automatically restart some (but not all) system calls that otherwise would have returned early (with EINTR error). The latter means we can simplify the rest of code somewhat because a restart loop may no longer be required.

Edit

New Page

▼ Pages 51

[Home](#)

[#Example Markdown](#)

[#Informal Glossary](#)

[#Piazza: When And How to Ask For Help](#)

[C Programming, Part 1: Introduction](#)

[C Programming, Part 2: Text Input And Output](#)

[C Programming, Part 3: Common Gotchas](#)

[C Programming, Part 4: Debugging](#)

[Deadlock, Part 1: Resource Allocation Graph](#)

[Deadlock, Part 2: Deadlock Conditions](#)

[File System, Part 1: Introduction](#)

[File System, Part 2: Files are inodes \(everything else is just data...\)](#)


[File System, Part 3: Permissions](#)

[File System, Part 4: Working with directories](#)


[File System, Part 5: Virtual file systems](#)


Show 36 more pages...


Clone this wiki locally


 Clone in Desktop

<>









```
sigfillset(&sa.sa_mask);
sa.sa_flags = SA_RESTART; /* Restart functions if interrupted by handler */
```

How do I use sigwait?

Sigwait can be used to read one pending signal at a time. `sigwait` is used to synchronously wait for signals, rather than handle them in a callback. A typical use of sigwait in a multi-threaded program is shown below. Notice that the thread signal mask is set first (and will be inherited by new threads). This prevents signals from being *delivered* so they will remain in a pending state until sigwait is called. Also notice the same set sig_t variable is used by sigwait - except rather than setting the set of blocked signals it is being used as the set of signals that sigwait can catch and return.

One advantage of writing a custom signal handling thread (such as the example below) rather than a callback function is that you can now use many more C library and system functions that otherwise could not be safely used in a signal handler because they are not async signal-safe.

Based on

http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_sigmask.html

```
static sigset_t    signal_mask; /* signals to block          */

int main (int argc, char *argv[])
{
    pthread_t sig_thr_id;      /* signal handler thread ID */
    sigemptyset (&signal_mask);
    sigaddset (&signal_mask, SIGINT);
    sigaddset (&signal_mask, SIGTERM);
    pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);

    /* New threads will inherit this thread's mask */
    pthread_create (&sig_thr_id, NULL, signal_thread, NULL);

    /* APPLICATION CODE */
    ...
}

void *signal_thread (void *arg)
{
    int          sig_caught;    /* signal caught          */

    /* Use same mask as the set of signals that we'd like to know about! */
    sigwait(&signal_mask, &sig_caught);
    switch (sig_caught)
    {
        case SIGINT:          /* process SIGINT      */
            ...
            break;
        case SIGTERM:         /* process SIGTERM     */
            ...
            break;
        default:              /* should normally not happen */
            fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
            break;
    }
}
```

Legal and Licensing information: Unless otherwise specified, submitted content to the wiki must be original work (including text, java code, and media) and you provide this material under a [Creative Commons License](#). If you are not the copyright holder, please give proper attribution and credit to existing content and ensure that you have license to include the materials.

