The sigaction structure is defined in the man pages 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);
};
```

Parameter `sa_mask` specifies a mask of signals which will be blocked (will not make it though) during execution of the signal handler.

To create a mask of signals to pass to `sigaction`, you should:

- Initialize an empty signal mask using `int sigemptyset(sigset_t *set);`
- Add a signal to the mask using `int sigaddset(sigset_t *set, int signum);`

In addition to the mask supplied, the signal which triggered the handler will be blocked, added to the mask – this is to avoid signal being sent repetitevely while it is being handled. (?)

If `SA_SIGINFO` is included in `sa_flags` mask, then `sa_sigaction` (instead of `sa_handler` by default) specifies the signal-handling function for `signum`.

To set flags:

```
flag = FLAG1 | FLAG2 | FLAG3
```

Signature of the function passed via `sa_sigaction` (which registeres a more sophisticated handler) looks like:

```
void handler(int sig, siginfo_t *info, void *ucontext);
```

This handler takes 3 arguments follows:

- `sig` The number of the signal that caused invocation of the handler.
- `info` A pointer to a `siginfo_t`, which is a structure containing further information about the signal, as described below.
- `ucontext` This is a pointer to a `ucontext_t` structure, cast to `(void *)`. The structure pointed to by field contains some more information about the signal. Commonly this is not used.

Each signal is thus supplememted with a `siginfo_t` structure with the following fields:

```
siginfo_t {
   int      si_signo;     /* Signal number */
   int      si_errno;     /* An errno value */
   int      si_code;      /* Signal code */
   int      si_trapno;    /* Trap number that caused
                             hardware-generated signal
                             (unused on most architectures) */
   pid_t    si_pid;       /* Sending process ID */
   uid_t    si_uid;       /* Real user ID of sending process */
   int      si_status;    /* Exit value or signal */
   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 */
   int      si_overrun;   /* Timer overrun count;
                             POSIX.1b timers */
   int      si_timerid;   /* Timer ID; POSIX.1b timers */
   void    *si_addr;      /* Memory location which caused fault */
   long     si_band;      /* Band event (was int in
                             glibc 2.3.2 and earlier) */
   int      si_fd;        /* File descriptor */
   short    si_addr_lsb;  /* Least significant bit of address
                             (since Linux 2.6.32) */
   void    *si_lower;     /* Lower bound when address violation
                             occurred (since Linux 3.19) */
   void    *si_upper;     /* Upper bound when address violation
                             occurred (since Linux 3.19) */
   int      si_pkey;      /* Protection key on PTE that caused
                             fault (since Linux 4.6) */
   void    *si_call_addr; /* Address of system call instruction
                             (since Linux 3.5) */
   int      si_syscall;   /* Number of attempted system call
                             (since Linux 3.5) */
   unsigned int si_arch;  /* Architecture of attempted system call
                             (since Linux 3.5) */
}
```

Fields we care about for handling `SIGCHLD`

- `SIGCHLD` fills in `si_pid`, `si_uid`, `si_status`, `si_utime`, and `si_stime`, providing information about the child.

- The `si_status` field contains the exit status of the child or the signal number that caused the process to change state.
- The `si_utime` and `si_stime` contain the user and system CPU time used by the child process.
- AND IMPORTANTLY, `si_code`:

The following values can be placed in `si_code` for a `SIGCHLD` signal:

```
CLD_EXITED
       Child has exited.

CLD_KILLED
       Child was killed.

CLD_DUMPED
       Child terminated abnormally.

CLD_TRAPPED
       Traced child has trapped.

CLD_STOPPED
       Child has stopped.

CLD_CONTINUED (since Linux 2.6.9)
       Stopped child has continued.
```