Unix Signals

Overview A Unix **signal**, also known as *software interrupts*, is a notification to a process that an event has occurred. Signals can be sent

- by one process to another process (or to itself),
- by the kernel to the process

Signal action Every signal has a **action**, which is also called the **disposition** associated with the signal. The action of the signal can be set by `sigaction()` function. There are three choices for the action:

1. We can set up a functino to be called whenever a specific signal occurs. This function is called a **signal handler** and this action is called **catching** a signal. Two signals `SIGKILL` and `SIGSTOP` cannot be called. Function prototype of signal handler is

   ```
   void handler(int signo);
   ```

2. We can **ignore** a signal by setting its disposition to `SIG_IGN`. The two signals `SIGKILL` and `SIGSTOP` cannot be ignored.
3. We can set the **default** disposition for a signal by setting its disposition to `SIG_DFL`.

The `sigaction()` function

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

`signum` can be any signal except for `SIGKILL` and `SIGSTOP`. `act` has the following structure type:

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

The `sa_handler` can be one of `SIG_DFL`, `SIG_IGN` or a pointer to `void signal_handler(int)` function.

The `sa_mask` specifies a mask of signals which should be blocked (i.e. added to the signal mask of the thread in which the signal handler is blocked) during the execution of the signal handler. .