Signals

Signals

- A **signal** is a software notification to a process of an event.
- A signal is **generated** when the event that causes the signal occurs.
- A signal is **delivered** when the process takes action based on that signal.
- The **lifetime** of a signal is the interval between its generation and its delivery.
- A signal that has been generated but not yet delivered is said to be **pending**.

Generating Signals

- Signals can be generated from the command line with the `kill` command.

```
kill -s signal_name pid...
kill -l [exit_status]
kill [-signal_name] pid...
kill [-signal_number] pid...
```

Generating Signals

- Signals can be generated in a process with the `kill` function to send a signal to another process:

```
#include<signal.h>
int kill(pid_t, int sig);
```

- Signals can be generated in a process with the `raise` function to send a signal to itself:

```
#include<signal.h>
int raise(int sig);
```

Signal Masks

- A process can temporarily prevent a signal from being delivered by blocking it.
- The **signal mask** gives the set of signals that are currently blocked.
- Note that blocking a signal is different than ignoring a signal; a pending blocked signal will be delivered once the process unblocks the signal.

Signal Sets

- Signal sets are groups of signals that can be operated on and have type `sigset_t`
- The following functions are used for manipulating signal sets

```
#include<signal.h>

int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigismember(const sigset_t *set, int signo);
```

Catching Signals

- A process catches a signal if it executes a **signal handler** when the signal is delivered.
- A program installs a **signal handler** by calling by calling `sigaction` with the name of a user-written function.

```
#include<signal.h>
int sigaction(int sig,
              const stuct sigaction *restrict act,
              strcut sigaction *restrict oact);
struct sigaction {
    void (*sa_handler) (int);
    sigset_t sa_mask;
    int sa_flags;
    void(*sa_sigaction) (int, siginfo *, void *);
}
```

Waiting for Signals

- The `pause` function suspends the calling thread until the delivery of a signal.

```
int pause(void);
```

- The `sigsuspend` function sets a signal mask and suspends the process until a signal is caught by the process.

```
int sigsuspend(const sigset_t *sigmask);
```

- The `sigwait` function blocks until any of the specified signals is pending then removes that signal from the set of pending signals and unblocks.

```
int sigwait(const sigset_t *restrict sigmask,
            int *restrict signo);
```

Errors and Async-signal Safety

- Difficulties when signals interact with function calls
  - Restarting a function that is interrupted by a signal
  - Signal handlers calling non-reentrant functions
  - Handling errors that use `errno`