Signal Functions

1. signal

Purpose: Sets a function as a signal handler for a specific signal.

Prototype:

```c
void (*signal(int signum, void (*handler)(int)))(int);
```

Parameters:

signum: The signal number (e.g., SIGINT, SIGALRM) for which you want to set the handler.

handler: A pointer to the signal handler function that should be executed when the signal is received.

Description: This function tells the operating system to invoke the handler function whenever the process receives the specified signal.

Example:

```c
#include <signal.h>
#include <stdio.h>

void handler(int signum) {
    printf("Caught signal %d\n", signum);
}

int main() {
    signal(SIGINT, handler);  // Catch Ctrl+C
    while (1) {
        // Infinite loop to wait for signal
    }
    return 0;
}
```

2. sigaction

Purpose: Allows finer control over signal handling compared to signal(). It can modify signal behaviors and retrieve previous handlers.

Prototype:

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

Parameters:
signum: The signal number (e.g., SIGUSR1, SIGALRM) to handle.
act: Pointer to a struct sigaction that describes the new action for the signal.
oldact: Pointer to store the previous action (can be NULL if not needed).

Description: Allows a process to specify how it handles a particular signal (whether to catch it, ignore it, or handle it in some custom way).

Example:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>

void handler(int signum) {
    printf("Handled signal %d\n", signum);
}

int main() {
    struct sigaction act;
    act.sa_handler = handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    sigaction(SIGUSR1, &act, NULL);
    kill(getpid(), SIGUSR1);
    return 0;
}

3. kill
Purpose: Sends a signal to a process or a group of
processes.

Prototype:
int kill(pid_t pid, int sig);

Parameters:
pid: The process ID to which the signal should be sent.
Special values like -1 can be used to target all processes.
sig: The signal number to send (e.g., SIGTERM, SIGKILL).

Description: This function allows a process to send a signal
to another process (or itself).

Example:
#include <signal.h>
#include <stdio.h>

int main() {
    pid_t pid = getpid();
    printf("Sending SIGUSR1 to process %d\n", pid);
    kill(pid, SIGUSR1);  // Sends SIGUSR1 to the current
process
    return 0;
}

4. sigqueue
Purpose: Sends a signal to a process with an accompanying
value.

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

```
Parameters:
pid: The process ID to send the signal to.
sig: The signal number to send.
value: A union sigval value to send alongside the signal.
Description: Extends kill() by allowing the sending of
additional data with the signal.

Example:
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

void handler(int sig, siginfo_t *si, void *unused) {
    printf("Received signal %d with value %d\n", sig, si-
>si_value.sival_int);
}

int main() {
    struct sigaction act;
    act.sa_sigaction = handler;
    act.sa_flags = SA_SIGINFO;
    sigaction(SIGUSR1, &act, NULL);

    union sigval value;
    value.sival_int = 42;
    sigqueue(getpid(), SIGUSR1, value);
    return 0;
}


5. sigwait
Purpose: Waits for one of the signals in the provided set.

Prototype:
int sigwait(const sigset_t *set, int *sig);

Parameters:
set: A signal set that specifies the signals to wait for.
```

```
sig: A pointer to store the received signal number.
Description: Blocks until one of the signals in set is
delivered.

Example:
#include <signal.h>
#include <stdio.h>

int main() {
    sigset_t set;
    int sig;

    sigemptyset(&set);
    sigaddset(&set, SIGINT);

    printf("Waiting for SIGINT (Ctrl+C)...\n");
    sigwait(&set, &sig);
    printf("Received signal: %d\n", sig);
    return 0;
}


6. sigsuspend

Purpose: Temporarily replaces the signal mask and suspends
the process until a signal is caught.

Prototype:
int sigsuspend(const sigset_t *mask);

Parameters:
mask: The new mask to apply temporarily while waiting for a
signal.

Description: The process is suspended until a signal
arrives, which is not blocked by the mask.
```

```
Example:
#include <signal.h>
#include <stdio.h>

int main() {
    sigset_t set;
    sigemptyset(&set);
    sigaddset(&set, SIGINT);

    printf("Press Ctrl+C to continue...\n");
    sigsuspend(&set);  // Temporarily blocks SIGINT and
waits for any signal
    return 0;
}


7. waitpid
Purpose: Waits for a specific child process to change its
state.

Prototype:
pid_t waitpid(pid_t pid, int *status, int options);

Parameters:
pid: The PID of the child process to wait for.
status: A pointer to store the status information of the
child process.
options: Flags to modify the behavior of waitpid() (e.g.,
WNOHANG).

Description: This function blocks until the specified child
process changes state (e.g., terminates).

Example:
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
```

```
int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child process
        printf("Child process\n");
        sleep(2);
    } else {
        // Parent process
        int status;
        waitpid(pid, &status, 0);
        printf("Child exited with status %d\n", status);
    }

    return 0;
}
```

8. sigemptyset

Purpose: Initializes a signal set to exclude all signals.

Prototype:

```
int sigemptyset(sigset_t *set);
```

Parameters:

set: A pointer to the signal set to initialize.

Description: Creates an empty signal set (i.e., no signals are in the set).

Example:

```
sigset_t set;
sigemptyset(&set);  // Creates an empty signal set
```

```
9. sigaddset

Purpose: Adds a signal to a signal set.

Prototype:
int sigaddset(sigset_t *set, int signum);

Parameters:
set: The signal set to modify.
signum: The signal to add to the set.

Description: Adds a specific signal to the signal set.

Example:
sigaddset(&set, SIGINT);  // Adds SIGINT to the set


10. sigprocmask

Purpose: Examines or changes the signal mask.

Prototype:
int sigprocmask(int how, const sigset_t *set, sigset_t
*oldset);

Parameters:
how: Specifies how the mask should be modified (e.g.,
SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK).
set: The new signal mask to apply.
oldset: A pointer to store the previous signal mask.
Description: Modifies the signal mask for the calling
process.

Example:
sigprocmask(SIG_BLOCK, &set, NULL);  // Blocks signals in
set
```