Lab 11: 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. There may be considerable time between signal generation and signal delivery. The process must be running on a processor at the time of signal delivery.

A process catches a signal if it executes a signal handler when the signal is delivered. A program installs a signal handler by calling `sigaction` with the name of a user-written function. The `sigaction` function may also be called with `SIG_DFL` or `SIG_IGN` instead of a handler. The `SIG_DFL` means take the default action, and `SIG_IGN` means ignore the signal. Neither of these actions is considered to be "catching" the signal. If the process is set to ignore a signal, that signal is thrown away when delivered and has no effect on the process.

The action taken when a signal is generated depends on the current signal handler for that signal and on the process signal mask. The signal mask contains a list of currently blocked signals. It is easy to confuse blocking a signal with ignoring a signal. Blocked signals are not thrown away as ignored signals are. If a pending signal is blocked, it is delivered when the process unblocks that signal. A program blocks a signal by changing its process signal mask, using `sigprocmask`. A program ignores a signal by setting the signal handler to `SIG_IGN`, using `sigaction`.

Manipulating Signal Masks and Signal Sets

A process can temporarily prevent a signal from being delivered by blocking it. Blocked signals do not affect the behavior of the process until they are delivered. The process signal mask gives the set of signals that are currently blocked. The signal mask is of type `sigset_t`.

Blocking a signal is different from ignoring a signal. When a process blocks a signal, the operating system does not deliver the signal until the process unblocks the signal. A process blocks a signal by modifying its signal mask with `sigprocmask`. When a process ignores a signal, the signal is delivered and the process handles it by throwing it away. The process sets a signal to be ignored by calling `sigaction` with a handler of `SIG_IGN`.

Specify operations (such as blocking or unblocking) on groups of signals by using signal sets of type `sigset_t`. Signal sets are manipulated by the five functions listed in the following synopsis box. The first parameter for each function is a pointer to a `sigset_t`. The `sigaddset` adds `signo` to the signal set, and the `sigdelset` removes `signo` from the signal set. The `sigemptyset` function initializes a `sigset_t` to contain no signals;

`sigfillset` initializes a `sigset_t` to contain all signals. Initialize a signal set by calling either `sigemptyset` or `sigfillset` before using it. The `sigismember` reports whether `signo` is in a `sigset_t`.

```
SYNOPSIS

  #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);
```

The `sigismember` function returns 1 if `signo` is in `*set` and 0 if `signo` is not in `*set`. If successful, the other functions return 0. If unsuccessful, these other functions return –1 and set `errno`. POSIX does not define any mandatory errors for these functions.

A process can examine or modify its process signal mask with the `sigprocmask` function. The `how` parameter is an integer specifying the manner in which the signal mask is to be modified. The `set` parameter is a pointer to a signal set to be used in the modification. If `set` is `NULL`, no modification is made. If `oset` is not `NULL`, the `sigprocmask` returns in `*oset` the signal set before the modification.

```
SYNOPSIS

  #include <signal.h>

  int sigprocmask(int how, const sigset_t *restrict set,
                  sigset_t *restrict oset);

POSIX:CX
```

If successful, `sigprocmask` returns 0. If unsuccessful, `sigprocmask` returns –1 and sets `errno`. The `sigprocmask` function sets `errno` to `EINVAL` if `how` is invalid. The `sigprocmask` function should only be used by a process with a single thread. When multiple threads exist, the `pthread_sigmask` function (page 474) should be used.

The `how` parameter, which specifies the manner in which the signal mask is to be modified, can take on one of the following three values.

`SIG_BLOCK`: add a collection of signals to those currently blocked

`SIG_UNBLOCK`: delete a collection of signals from those currently blocked

`SIG_SETMASK`: set the collection of signals being blocked to the specified set

Keep in mind that some signals, such as `SIGSTOP` and `SIGKILL`, cannot be blocked. If an attempt is made to block these signals, the system ignores the request without reporting an error.

Catching and Ignoring Signals—sigaction

The sigaction function allows the caller to examine or specify the action associated with a specific signal. The sig parameter of sigaction specifies the signal number for the action. The act parameter is a pointer to a struct sigaction structure that specifies the action to be taken. The oact parameter is a pointer to a struct sigaction structure that receives the previous action associated with the signal. If act is NULL, the call to sigaction does not change the action associated with the signal. If oact is NULL, the call to sigaction does not return the previous action associated with the signal.

```
SYNOPSIS

  #include <signal.h>

  int sigaction(int sig, const struct sigaction *restrict act,
                struct sigaction *restrict oact);

POSIX:CX
```

If successful, sigaction returns 0. If unsuccessful, sigaction returns –1 and sets errno. The following table lists the mandatory errors for sigaction.

Waiting for Signals—pause, sigsuspend and sigwait

Signals provide a method for waiting for an event without busy waiting. Busy waiting means continually using CPU cycles to test for the occurrence of an event. Typically, a program does this testing by checking the value of a variable in a loop. A more efficient approach is to suspend the process until the waited-for event occurs; that way, other processes can use the CPU productively. The POSIX pause, sigsuspend and sigwait functions provide three mechanisms for suspending a process until a signal occurs.

The pause function

The pause function suspends the calling thread until the delivery of a signal whose action is either to execute a user-defined handler or to terminate the process. If the action is to terminate, pause does not return. If a signal is caught by the process, pause returns after the signal handler returns.

```
SYNOPSIS

  #include <unistd.h>

  int pause(void);
```

The pause function always returns –1. If interrupted by a signal, pause sets errno to EINTR.

To wait for a particular signal by using pause, you must determine which signal caused pause to return. This information is not directly available, so the signal handler must set a flag for the program to check after pause returns.

The sigsuspend function

The delivery of a signal before pause was one of the major problems with the original UNIX signals, and there was no simple, reliable way to get around the problem. The program must do two operations "at once"—unblock the signal and start pause. Another way of saying this is that the two operations together should be atomic (i.e., the program cannot be logically interrupted between execution of the two operations). The sigsuspend function provides a method of achieving this.

The sigsuspend function sets the signal mask to the one pointed to by sigmask and suspends the process until a signal is caught by the process. The sigsuspend function returns when the signal handler of the caught signal returns. The sigmask parameter can be used to unblock the signal the program is looking for. When sigsuspend returns, the signal mask is reset to the value it had before the sigsuspend function was called.

```
SYNOPSIS
  #include <signal.h>

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

The sigwait function

The sigwait function blocks until any of the signals specified by *sigmask is pending and then removes that signal from the set of pending signals and unblocks. When sigwait returns, the number of the signal that was removed from the pending signals is stored in the location pointed to by signo.

```
SYNOPSIS
  #include <signal.h>

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

If successful, sigwait returns 0. If unsuccessful, sigwait returns –1 and sets errno. No mandatory errors are defined for sigwait.

Note the differences between sigwait and sigsuspend. Both functions have a first parameter that is a pointer to a signal set (sigset_t *). For sigsuspend, this set holds the new signal mask and so the signals that are not in the set are the ones that can cause sigsuspend to return. For sigwait, this parameter holds the set of signals to be waited for, so the signals in the set are the ones that can cause the sigwait to return. Unlike sigsuspend, sigwait does not change the process signal mask. The signals in sigmask should be blocked before sigwait is called.

Task: Implement wait ( ) function

a. By changing the default behavior of SIGCHLD (without using pause or sigsuspend or sigwait)

b. Using pause () function

c. Using signal suspend option

d. Using sigwait