Signals

- Signals interrupt processes, causing them to do something other than normal:

Signals

- Signals interrupt processes, causing them to do something other than normal:
  - die (ie. terminate, and usually, core dump)

Signals

- Signals interrupt processes, causing them to do something other than normal:
  - die (ie. terminate, and usually, core dump)
  - wake up; break out of a blocked state (such as when blocked on a read)

Signals

- Signals interrupt processes, causing them to do something other than normal:
  - die (ie. terminate, and usually, core dump)
  - wake up; break out of a blocked state (such as when blocked on a read)
  - receive an alarm

Signals

- Signals interrupt processes, causing them to do something other than normal:
  - die (ie. terminate, and usually, core dump)
  - wake up; break out of a blocked state (such as when blocked on a read)
  - receive an alarm
- The signal.h file contains definitions for signals (ie. #define SIGFPE 8) (actually, signal.h includes other files with those definitions)

Signals

- Signals interrupt processes, causing them to do something other than normal:
  - die (ie. terminate, and usually, core dump)
  - wake up; break out of a blocked state (such as when blocked on a read)
  - receive an alarm
- The signal.h file contains definitions for signals (ie. #define SIGFPE 8) (actually, signal.h includes other files with those definitions)
- Essentially, signals are integers

List of Signals #1

SIGALRM alarm timer

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered
SIGINT	user hit ctrl-c

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered
SIGINT	user hit ctrl-c
SIGKILL	sure-kill. Cannot be stopped. Process *will* die!

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered
SIGINT	user hit ctrl-c
SIGKILL	sure-kill. Cannot be stopped. Process *will* die!
SIGPIPE	illegal write to a pipe (no reader)

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered
SIGINT	user hit ctrl-c
SIGKILL	sure-kill. Cannot be stopped. Process *will* die!
SIGPIPE	illegal write to a pipe (no reader)
SIGQUIT	process quit signal (ctrl-backslash)

List of Signals #1

SIGALRM	alarm timer
SIGABRT	process abort (see abort())
SIGFPE	floating point error
SIGHUP	controlling terminal exited
SIGILL	illegal machine instruction encountered
SIGINT	user hit ctrl-c
SIGKILL	sure-kill. Cannot be stopped. Process *will* die!
SIGPIPE	illegal write to a pipe (no reader)
SIGQUIT	process quit signal (ctrl-backslash)
SIGSEGV	segmentation fault: attempt to read/write to memory the process doesn't own

List of Signals #2

SIGTERM process termination (kill pid)

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated
SIGCONT	(job control) resume execution of a stopped process

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated
SIGCONT	(job control) resume execution of a stopped process
SIGSTOP	(job control) stop a process' execution

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated
SIGCONT	(job control) resume execution of a stopped process
SIGSTOP	(job control) stop a process' execution
SIGTTIN	stop a background process' attempt to read from its terminal

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated
SIGCONT	(job control) resume execution of a stopped process
SIGSTOP	(job control) stop a process' execution
SIGTTIN	stop a background process' attempt to read from its terminal
SIGTSTP	stop a process' execution via a ctrl-z

List of Signals #2

SIGTERM	process termination (kill pid)
SIGUSR1	programmer defined meaning
SIGUSR2	programmer defined meaning
SIGCHLD	child process has terminated
SIGCLD	(not Posix) child process has terminated
SIGCONT	(job control) resume execution of a stopped process
SIGSTOP	(job control) stop a process' execution
SIGTTIN	stop a background process' attempt to read from its terminal
SIGTSTP	stop a process' execution via a ctrl-z
SIGTTOU	stop a background process' attempt to write to its terminal

Signal Definitions

pending A signal is *pending* when the process must handle it, but hasn’t done so already.

Signal Definitions

pending A signal is *pending* when the process must handle it, but hasn't done so already.

A process may, at the programmer's discretion, do one of three actions upon receipt of a signal (except for SIGKILL):

Signal Definitions

pending A signal is *pending* when the process must handle it, but hasn't done so already.

A process may, at the programmer's discretion, do one of three actions upon receipt of a signal (except for SIGKILL):

accept the default action (terminate, core dump, etc)

Signal Definitions

pending A signal is *pending* when the process must handle it, but hasn't done so already.

A process may, at the programmer's discretion, do one of three actions upon receipt of a signal (except for SIGKILL):

accept the default action (terminate, core dump, etc)

ignore signal. Discards the signal.

Signal Definitions

pending A signal is *pending* when the process must handle it, but hasn't done so already.

A process may, at the programmer's discretion, do one of three actions upon receipt of a signal (except for SIGKILL):

accept the default action (terminate, core dump, etc)

ignore signal. Discards the signal.

invoke function Programmer may have a custom action tailored to his/her needs in the form of a function used to "catch" the signal. Typically used with:

SIGALRM, *SIGUSR1*, *SIGUSR2*, *SIGINT*

Notes on Signals

- Signals are **asynchronous**; your program will stop whatever it is doing (immediately!) and execute the action demanded by the signal.

  asynchronous: (*adjective*) activity which happens irregardless of current operation

Notes on Signals

- Signals are **asynchronous**; your program will stop whatever it is doing (immediately!) and execute the action demanded by the signal.

  asynchronous: (*adjective*) activity which happens irregardless of current operation

- *Bad Idea*: putting system function calls inside signal handling functions, especially those that access a resource (ie. printf, scanf, etc)

Notes on Signals

- Signals are **asynchronous**; your program will stop whatever it is doing (immediately!) and execute the action demanded by the signal.

  asynchronous: (*adjective*) activity which happens irregardless of current operation

- *Bad Idea*: putting system function calls inside signal handling functions, especially those that access a resource (ie. printf, scanf, etc)

- Exception: those signal handlers which are merely doing a little cleanup (freeing memory, resources, etc) immediately followed by calling exit().

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

  0 use default action

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

  0 use default action

  1 ignore signal

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

  0 use default action

  1 ignore signal

  + pointer to function to be executed

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

  0 use default action

  1 ignore signal

  + pointer to function to be executed

- Multiple pending signals are processed in whatever order is convenient. (ie. *not* necessarily in the order received) *No "signal priority" scheme is used.*

Unix Kernel Support

- The process table has an array of signal flags (one for each type of signal)
- Receiving a signal causes the flag to be set
- Sleeping processes (wait(), pause(), etc) are scheduled to run
- Process' u-area holds signal handling specifications

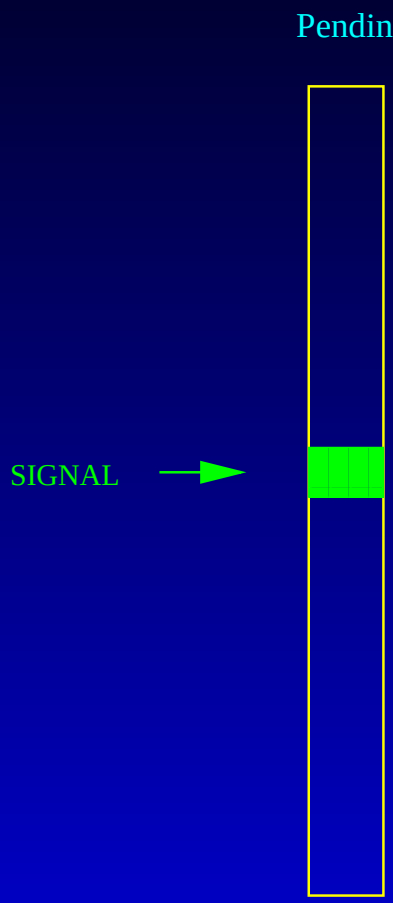
  0 use default action

  1 ignore signal

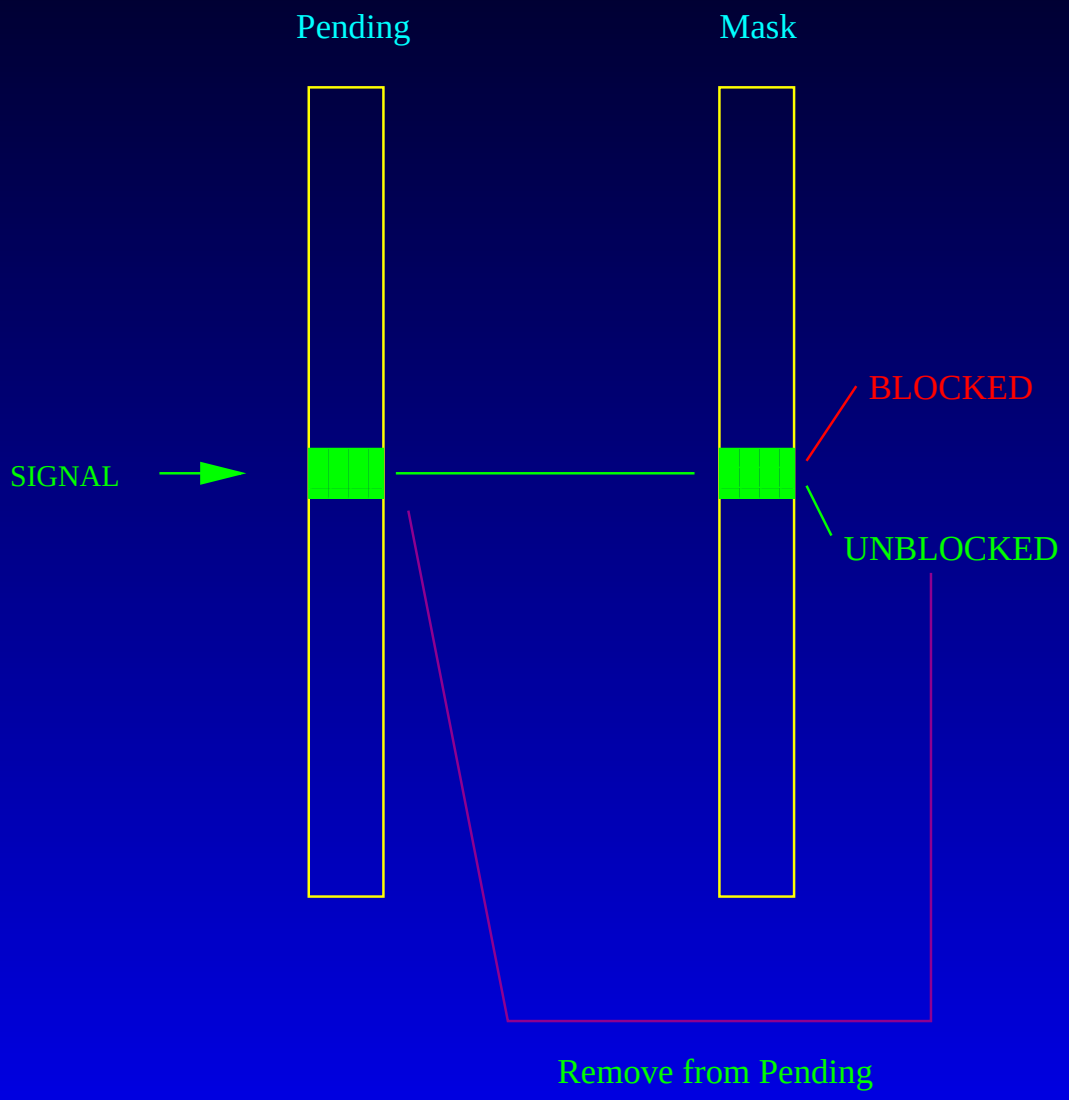
  + pointer to function to be executed

- Multiple pending signals are processed in whatever order is convenient. (ie. *not* necessarily in the order received) *No "signal priority" scheme is used.*
- Multiple signals of the same type received while signals are pending are *not* counted; only one such signal will be processed

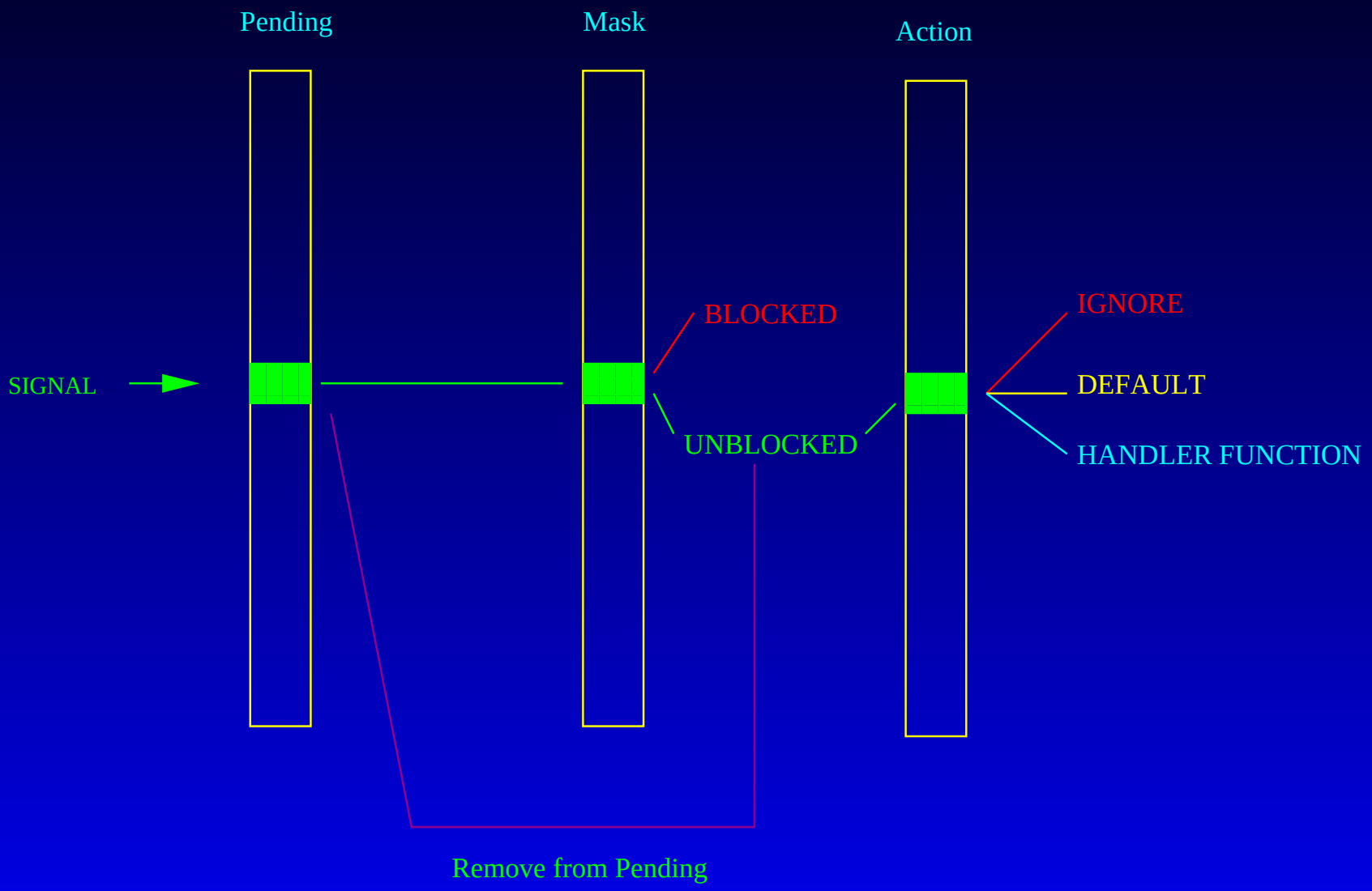
Pending-Mask-Action



Pending-Mask-Action



Pending-Mask-Action



signal()

#include <signal.h> (ANSI, not POSIX)

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

signum SIGALRM, SIGUSR1, etc: use one of the signals defined via <signal.h>

signal()

#include <signal.h> (ANSI, not POSIX)

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

signum SIGALRM, SIGUSR1, etc: use one of the signals defined via <signal.h>

sighandler The programmer's signal handling function, or...

signal()

#include <signal.h> (ANSI, not POSIX)

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

signum SIGALRM, SIGUSR1, etc: use one of the signals defined via <signal.h>

sighandler The programmer's signal handling function, or...

SIG_DFL (void (*)(int)) 0: default action

signal()

#include <signal.h> (ANSI, not POSIX)

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

signum SIGALRM, SIGUSR1, etc: use one of the signals defined via <signal.h>

sighandler The programmer's signal handling function, or...

SIG_DFL (void (*)(int)) 0: default action

SIG_IGN (void (*)(int)) 1: ignore signal

signal()

#include <signal.h> (ANSI, not POSIX)

typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

signum SIGALRM, SIGUSR1, etc: use one of the signals defined via <signal.h>

sighandler The programmer's signal handling function, or...

SIG_DFL (void (*)(int)) 0: default action

SIG_IGN (void (*)(int)) 1: ignore signal

typical signal handling function :

```
void MySignalHandler(int signum)
{
        signal(signum,MySignalHandler);     // re-install signal handler!
        ...
}
```

Discussion on signal()

- signal() is unreliable because...

Discussion on signal()

- signal() is unreliable because...
- The signal handling is reset to SIG_DFL; consequently, the signal handling function had to re-install itself (usually as the very first thing the function does)

Discussion on signal()

- signal() is unreliable because...
- The signal handling is reset to SIG_DFL; consequently, the signal handling function had to re-install itself (usually as the very first thing the function does)
- It is unreliable because there's a chance that another signal of the same type will arrive prior to the signal function having re-installed itself, and so the default action would then be taken.

Discussion on signal()

- signal() is unreliable because...
- The signal handling is reset to SIG_DFL; consequently, the signal handling function had to re-install itself (usually as the very first thing the function does)
- It is unreliable because there's a chance that another signal of the same type will arrive prior to the signal function having re-installed itself, and so the default action would then be taken.
- sigset() is reliable!

Discussion on signal()

- signal() is unreliable because...
- The signal handling is reset to SIG_DFL; consequently, the signal handling function had to re-install itself (usually as the very first thing the function does)
- It is unreliable because there's a chance that another signal of the same type will arrive prior to the signal function having re-installed itself, and so the default action would then be taken.
- sigset() is reliable!
- sigaction() is reliable!

sigset()

(Sys V3,4 signal handling)

#define _XOPEN_SOURCE 500 #include <signal.h>	needed for SysV handling
typedef void (*sighandler_t)(int)	handler function
sighandler_t sigset(int sig, sighandler_t sighandler)	see next slide

- The signal handler does not get reset to SIG_DFL when invoked

sigset()

(Sys V3,4 signal handling)

#define _XOPEN_SOURCE 500 #include <signal.h>	needed for SysV handling
typedef void (*sighandler_t)(int)	handler function
sighandler_t sigset(int sig, sighandler_t sighandler)	see next slide

- The signal handler does not get reset to SIG_DFL when invoked
- When a signal is received, other signals of that type are blocked

sigset()

(Sys V3,4 signal handling)

#define _XOPEN_SOURCE 500 #include <signal.h>	needed for SysV handling
typedef void (*sighandler_t)(int)	handler function
sighandler_t sigset(int sig, sighandler_t sighandler)	see next slide

- The signal handler does not get reset to SIG_DFL when invoked
- When a signal is received, other signals of that type are blocked
- Developed by AT&T (Sys V3,4)

sigset()

(Sys V3,4 signal handling)

```
#define _XOPEN_SOURCE 500                                  needed for
#include <signal.h>                                        SysV handling

typedef void (*sighandler_t)(int)                          handler function

sighandler_t sigset(int sig, sighandler_t sighandler)      see next slide
```

- The signal handler does not get reset to SIG_DFL when invoked
- When a signal is received, other signals of that type are blocked
- Developed by AT&T (Sys V3,4)
- These functions are provided under Linux, too

sigset()

(Sys V3,4 signal handling)

#define _XOPEN_SOURCE 500 #include <signal.h>	needed for SysV handling
typedef void (*sighandler_t)(int)	handler function
sighandler_t sigset(int sig, sighandler_t sighandler)	see next slide

- The signal handler does not get reset to SIG_DFL when invoked
- When a signal is received, other signals of that type are blocked
- Developed by AT&T (Sys V3,4)
- These functions are provided under Linux, too
- Posix has marked sigset() as obsolete (it encourages use of sigaction() instead)

Signal Disposition

(Sys V3,4 signal handling)

Values the sighandler can have with sigset():

SIG_DFL reset the disposition of the sig to the default

Signal Disposition

(Sys V3,4 signal handling)

Values the sighandler can have with sigset():

SIG_DFL reset the disposition of the sig to the default

SIG_IGN ignore the sig

Signal Disposition

(Sys V3,4 signal handling)

Values the sighandler can have with sigset():

SIG_DFL reset the disposition of the sig to the default

SIG_IGN ignore the sig

SIG_HOLD add sig to the process' signal mask, but leave the disposition of sig unchanged

Signal Disposition

(Sys V3,4 signal handling)

Values the sighandler can have with sigset():

SIG_DFL reset the disposition of the sig to the default

SIG_IGN ignore the sig

SIG_HOLD add sig to the process' signal mask, but leave the disposition of sig unchanged

function one may specify a signal handling function. Its prototype is given by sighandler_t *sighandler*(int signum)

Signal Disposition

(Sys V3,4 signal handling)

Values the sighandler can have with sigset():

SIG_DFL reset the disposition of the sig to the default

SIG_IGN ignore the sig

SIG_HOLD add sig to the process' signal mask, but leave the disposition of sig unchanged

function one may specify a signal handling function. Its prototype is given by sighandler_t *sighandler*(int signum)

- SIGKILL and SIGSTOP dispositions cannot be changed.

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig) sets action to SIG_DFL (default)

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig)	sets action to SIG_DFL (default)
int sigrelse(int sig)	releases (removes) signal from mask

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig)	sets action to SIG_DFL (default)
int sigrelse(int sig)	releases (removes) signal from mask
int sigignore(int sig)	removes signal from mask AND suspends process until it is interrupted by a signal

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig)	sets action to SIG_DFL (default)
int sigrelse(int sig)	releases (removes) signal from mask
int sigignore(int sig)	removes signal from mask AND suspends process until it is interrupted by a signal

- These functions modify the disposition of the signal

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig)	sets action to SIG_DFL (default)
int sigrelse(int sig)	releases (removes) signal from mask
int sigignore(int sig)	removes signal from mask AND suspends process until it is interrupted by a signal

- These functions modify the disposition of the signal
- They may be used to override sigset()-specified disposition

Hold/Release/Ignore

(Sys V3,4 signal handling)

int sighold(int sig)	sets action to SIG_DFL (default)
int sigrelse(int sig)	releases (removes) signal from mask
int sigignore(int sig)	removes signal from mask AND suspends process until it is interrupted by a signal

- These functions modify the disposition of the signal
- They may be used to override sigset()-specified disposition
- Posix has marked these as obsolete functions

  (encourages use of sigaction() instead)

SIGCLD/SIGCHLD

SIGCLD SysV name for the signal received when a child terminates

SIGCLD/SIGCHLD

SIGCLD SysV name for the signal received when a child terminates

SIGCHLD Posix, BSD, Linux name for the signal received when a child terminates

(Sys V)

SIGCLD/SIGCHLD

SIGCLD SysV name for the signal received when a child terminates

SIGCHLD Posix, BSD, Linux name for the signal received when a child terminates

(Sys V)

SIGCLD is handled a bit idiosyncratically by SysV when using signal() or sigset():

SIGCLD/SIGCHLD

SIGCLD SysV name for the signal received when a child terminates

SIGCHLD Posix, BSD, Linux name for the signal received when a child terminates

(Sys V)

SIGCLD is handled a bit idiosyncratically by SysV when using signal() or sigset():

1. If disposition is SIG_IGN: children will not generate zombies.
   If disposition is SIG_DFL, will be ignored (child status is discarded)
   If parent uses a wait() style function, it will block until all children terminate and return -1s with errno having EINTR.

SIGCLD/SIGCHLD

SIGCLD SysV name for the signal received when a child terminates

SIGCHLD Posix, BSD, Linux name for the signal received when a child terminates

(Sys V)

SIGCLD is handled a bit idiosyncratically by SysV when using signal() or sigset():

1. If disposition is SIG_IGN: children will not generate zombies.
   If disposition is SIG_DFL, will be ignored (child status is discarded)
   If parent uses a wait() style function, it will block until all children terminate and return -1s with errno having EINTR.
2. If SIGCLD is to be caught, the kernel immediately checks if any child processes are ready to be waited for and, if so, calls the SIGCLD handler.
   This (weird) behavior can cause an infinite regression of signal handler calls if the handler attempts to re-install itself (as is the usual practice with signal())

sigaction()

(Posix signal handling)

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- The latest fashion in signal handling; supercedes both signal() and sigset()

sigaction()

(Posix signal handling)

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- The latest fashion in signal handling; supercedes both signal() and sigset()
- If act is not NULL, then a new action is installed to handle signal signum

sigaction()

(Posix signal handling)

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- The latest fashion in signal handling; supercedes both signal() and sigset()
- If act is not NULL, then a new action is installed to handle signal signum
- If oldact is not NULL, then the current action is saved in oldact

struct sigaction

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- If sa_flags has SA_SIGINFO, use sa_sigaction() rather than sa_handler() for signal handling

struct sigaction

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- If sa_flags has SA_SIGINFO, use sa_sigaction() rather than sa_handler() for signal handling
- sa_sigaction() takes three arguments: the signal number, a pointer to a siginfo_t (in two slides), and a pointer to a ucontext_t cast as a (void *). (see man ucontext.h)

struct sigaction

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- If sa_flags has SA_SIGINFO, use sa_sigaction() rather than sa_handler() for signal handling
- sa_sigaction() takes three arguments: the signal number, a pointer to a siginfo_t (in two slides), and a pointer to a ucontext_t cast as a (void *). (see man ucontext.h)
- sa_mask specifies a mask of signals which should be blocked during execution of the signal handler.

struct sigaction

```
struct sigaction {
  void (*sa_handler)(int);                          per-signal
  void (*sa_sigaction)(int, siginfo_t *, void *);   corresponds to sighandler
  sigset_t sa_mask;                                 corresponds to sighandler
  int sa_flags;                                     can be used to block more
  void (*sa_restorer)(void);                        signals when handling this one
  }                                                 obsolete, don't use
```

- If sa_flags has SA_SIGINFO, use sa_sigaction() rather than sa_handler() for signal handling
- sa_sigaction() takes three arguments: the signal number, a pointer to a siginfo_t (in two slides), and a pointer to a ucontext_t cast as a (void *). (see man ucontext.h)
- sa_mask specifies a mask of signals which should be blocked during execution of the signal handler.
- In addition, the triggering signal will also be blocked (unless SA_NODEFER is in the sa_flags)

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

SA_NOCLDSTOP block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

SA_NOCLDSTOP block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).

SA_NOCLDWAIT do not transform child process into zombie when they terminate

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

SA_NOCLDSTOP block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).

SA_NOCLDWAIT do not transform child process into zombie when they terminate

SA_NODEFER do not prevent the signal from being received from within its own signal handler

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

SA_NOCLDSTOP block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).

SA_NOCLDWAIT do not transform child process into zombie when they terminate

SA_NODEFER do not prevent the signal from being received from within its own signal handler

SA_ONSTACK call the signal handler on an alternate stack. (see sigaltstack())

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

- **SA_NOCLDSTOP** block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).
- **SA_NOCLDWAIT** do not transform child process into zombie when they terminate
- **SA_NODEFER** do not prevent the signal from being received from within its own signal handler
- **SA_ONSTACK** call the signal handler on an alternate stack. (see sigaltstack())
- **SA_RESETHAND** restore the signal action to default state after signal handler is called

sigaction(): sa_flags

sa_flags (0 if none of the following is desired)

SA_NOCLDSTOP block notification when child processes stop (as opposed to terminate). (SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU).

SA_NOCLDWAIT do not transform child process into zombie when they terminate

SA_NODEFER do not prevent the signal from being received from within its own signal handler

SA_ONSTACK call the signal handler on an alternate stack. (see sigaltstack())

SA_RESETHAND restore the signal action to default state after signal handler is called

SA_RESTART some system calls will be restarted after signal handling. If not set, the system call will be aborted and will return -1 with errno set to EINTR.