Signal

Signal Concepts

- signal
  - software interrupts
  - provides a way of handling asynchronous events
  - the kernel may send signals to process or processes may send each other signals with kill( ) system call

Signals

- signals for terminating processes
  - SIGHUP
    - This signal is sent to the controlling process associated with a controlling terminal if a disconnect is detected by the terminal interface.
    - This signal is also generated if the session leader terminates. In this case the signal is sent to each process in the foreground process group.

Signals

- signals for terminating processes
  - SIGHUP (cont'd)
    - This signal is commonly used to notify daemon processes to reread their configuration files. The reason SIGHUP is chosen for this is because a daemon should not have a controlling terminal and would normally never receive this signal.
    - default action: termination

Signals

- signals for terminating processes
  - SIGINT
    - sending of the interrupt character from the terminal [CTRL-C]
    - termination
  - SIGQUIT
    - sending of the quit character from the terminal [CTRL-\]
    - termination with core

Signals

- SIGABRT, SIGIOT
  - abnormal termination (abort( )), hardware fault
  - termination
- SIGKILL
  - irrevocable termination signal
  - termination
- SIGTERM
  - default signal sent out by the kill command
  - termination

Signals

- signals triggered by a particular physical circumstance
  - SIGILL
    - illegal instruction
    - termination
  - SIGTRAP
    - breakpoint in a program (system call ptrace)
    - termination with core
  - SIGBUS
    - bus error
    - termination

Signals

- SIGFPE
  - arithmetic error (floating point exception)
  - termination
- SIGSEGV
  - memory address invalid
  - termination with core
- SIGSTKFLT
  - maths co-processor stack overflow (only in the Intel architecture)
  - termination

Signals

- signals available for use by the programmer
  - SIGUSR1, SIGUSR2
    - termination
- signal generated when a pipe is closed
  - SIGPIPE
    - pipe without reader
    - termination

Signals

- suspending or resuming
  - SIGCHLD, SIGCLD
    - termination of child process
    - ignore
  - SIGCONT
    - repetition in the foreground or background of the process
    - resume execution if it is stopped
  - SIGSTOP
    - suspension of process
    - suspension (non-changeable)

Signals

- SIGTSTP
  - sending out of suspension character from the terminal [CTRL-Z]
  - suspension
- SIGTTIN
  - terminal read for a background process
  - suspension
- SIGTTOU
  - writing to a terminal for a background process
  - suspension

Signals

- linked to resources
  - SIGXCPU
    - CPU time limit exceeded
    - termination
  - SIGXFSZ
    - file size limit exceeded
    - termination
- management of alarms
  - SIGALRM
    - end of timer ITIMER_REAL or alarm()
    - termination

Signals

- SIGVTALRM
  - end of timer ITIMER_VIRTUAL
  - termination
- SIGPROF
  - end of timer ITIMER_PROF
  - termination

- management of inputs and outputs
  - SIGWINCH
    - change of window size (used by X11)
    - ignore

Signals

- SIGIO, SIGPOLL
  - data available for an input/output
  - termination
- SIGURG
  - urgent data for sockets
  - ignore

- fault with the power supply
  - SIGPWR, SIGINFO
    - power fault
    - termination

Handling Singal

- default action (SIG_DFL)
- ignore the signal (SIG_IGN)
- catch the signal
  - register signal handler
  - SIGKILL and SIGSTOP cannot be caught or ignored

signal( )

- synopsis
  - #include <signal.h>
  - void (*signal(int signum, void (*handler)(int)))(int);
- description
  - installs a new signal handler for the signal with number signum
  - the signal handler is set to handler which may be a user specified function, or one of the following:

signal( )

  - SIG_IGN： Ignore the signal.
  - SIG_DFL： Reset the signal to its default behavior
- The integer argument that is handed over to the signal handler routine is the signal number
- it possible to use one signal handler for several signals
- return value
  - the previous value of the signal handler, or SIG_ERR on error

signal( )

- Example

```
#include      <signal.h>
void myhandler(int signo)          /* argument is signal number */
{
        switch (signo) {
        case SIGINT : printf("SIGINT(%d) is caught\n",SIGINT);
                break;
        case SIGQUIT : printf("SIGQUIT(%d) is caught\n",SIGQUIT);
                break;
        case SIGTSTP : printf("SIGTSTP(%d) is caught\n",SIGTSTP);
                break;
        case SIGTERM : printf("SIGTERM(%d) is caught\n",SIGTERM);
                break;
        case SIGUSR1 : printf("SIGUSR1(%d) is caught\n",SIGUSR1);
                break;
        default: printf("other singal\n");
        }
        return;
}
```

signal( )

```
int main(void)
{
        signal(SIGINT, myhandler);
    //  signal(SIGINT, SIG_IGN);
        signal(SIGQUIT, myhandler);
        signal(SIGTSTP, myhandler);
     // signal(SIGTERM, SIG_DFL);
        signal(SIGTERM, myhandler);
        signal(SIGUSR1, myhandler);
        for (;;)
                pause();
}
```

signal( )

```
#include      <sys/types.h>
#include      <signal.h>
#include      <stdio.h>
void    sig_cld();
int main()
{
        pid_t   pid;

        if (signal(SIGCLD, sig_cld) == -1)
                perror("signal error");

        if ( (pid = fork()) < 0)
                perror("fork error");
        else if (pid == 0) {            /* child */
                sleep(2);
                _exit(0);
        }
        pause();        /* parent */
        exit(0);
}
```

```
void sig_cld()
{
        pid_t   pid;
        int     status;
        printf("SIGCLD received\n");
        if (signal(SIGCLD, sig_cld) == -1)
      /* reestablish handler */
                perror("signal error");
        if ( (pid = wait(&status)) < 0)
      /* fetch child status */
                perror("wait error");
        printf("pid = %d\n", pid);
        return;         /* interrupts pause() */
}
```

Interrupted System Calls

- characteristic of earlier UNIX systems
  - if a process caught a signal while the process was blocked in a “slow” system call, the system call was interrupted.
    - errno == EINTR
  - slow system calls
    - read from pipe, terminal, network devices
    - write to pipe, terminal, network devices
    - pause(), wait()
    - certain ioctl() operations
    - some of IPC functions

Interrupted System Calls

- automatic restarting
  - sometimes we don’t know that the input or output device is a slow device
  - ioctl, read, readv, write, writev, wait, waitpid

Reentrant Functions

_exit				
abort	fork	mkdir	sigaddset	time
access	fstat	mkfifo	sigdelset	times
alarm	getegid	open	sigemptyset	umask
chdir	geteuid	pathconf	sigfillset	uname
chmod	getgid	pause	sigismember	unlink
chown	getgroups	pipe	signal	utime
close	getpgrp	read	sigpending	wait
creat	getpid	rename	sigprocmask	waitpid
dup	getppid	rmdir	sigsuspend	write
dup2	getuid	setgid	sleep	
execve	kill	setpgid	stat	
exit	link	setsid	sysconf	
fcntl	longjmp	setuid	tcgetpgrp	
	lseek	sigaction	tcsetpgrp	

SIGCLD (SIGCHLD) Semantics

- SIG_IGN
  - children of the calling process will not generate zombie processes.
  - on termination the status of child processes is just discarded.
  - wait() returns –1, errno == ECHILD
- catch
  - the kernel immediately checks if there are any child process ready to be waited and, if so, calls the SIGCHLD handler.

kill( )

- synopsis
  - #include <sys/types.h>
  - #include <signal.h>
  - int kill(pid_t pid, int sig);
- description
  - used to send any signal to any process group or process

kill( )

- pid > 0
  - signal sig is sent to pid
- pid == 0
  - sig is sent to every process in the process group of the current process
- pid == –1 (broadcast)
  - sig is sent to every process except for the first one (init) from higher numbers in the process table to lower
- pid < -1
  - sig is sent to every process in the process group -pid

kill( )

- sig is 0
  - no signal is sent, but error checking is still performed
- return value
  - On success, zero is returned
  - On error, -1 is returned

raise( )

- synopsis
  - #include <signal.h>
  - int raise (int sig);
- description
  - sends a signal to the current process
  - kill(getpid( ),sig)
  - return value
    - 0 on success, nonzero for failure

alarm( )

- synopsis
  - #include <unistd.h>
  - unsigned int alarm(unsigned int seconds);

- description
  - arranges for a SIGALRM signal to be delivered to the process in seconds seconds.
  - if seconds is zero, no new alarm is scheduled
  - in any event any previously set alarm is cancelled

alarm( )

- return value
  - the number of seconds remaining until any previously scheduled alarm was due to be delivered
  - zero if there was no previously scheduled alarm

Interval Timer

- synopsis
  - #include <sys/time.h>
  - int getitimer(int which, struct itimerval *value);
  - int setitimer(int which, const struct itimerval *value, struct itimerval *ovalue);
- description
  - the system provides each process with three interval timers, each decrementing in a distinct time domain.
  - when any timer expires, a signal is sent to the process, and the timer (potentially) restarts.

Interval Timer

- time domain (which)
  - ITIMER_REAL
    - decrements in real time (SIGALRM)
  - ITIMER_VIRTUAL
    - decrements only when the process is executing (SIGVTALRM)
  - ITIMER_PROF
    - decrement both when the process executes and when the system is executing on behalf of the process (SIGPROF)

Interval Timer

- time value (value)

```
struct itimerval {
        struct timeval it_interval;  /* next value */
        struct timeval it_value;  /* current value */
}
struct timeval {
        long tv_sec;  /* seconds */
        long tv_usec;  /* microseconds */
}
```

  - timer decrement from it_value to zero, generate a signal, and reset to it_interval.
  - a timer which is set to zero stops
  - timer resolution is 10ms

pause( )

- synopsis
  - #include <unistd.h>
  - int pause(void);
- description
  - causes the invoking process to sleep until a signal is received
  - return value
    - always returns -1

abort( )

- synopsis
  - #include <stdlib.h>
  - void abort(void);
- description
  - causes abnormal program termination unless the signal SIGABRT is caught and the signal handler does not return
  - if the abort( ) function causes program termination, all open streams are closed and flushed

abort( )

- If the SIGABRT signal is blocked or ignored, the abort( ) function will still override it.
- return value
  - The abort() function never returns.

sleep( )

- synopsis
  - #include <unistd.h>
  - unsigned int sleep(unsigned int seconds);
- description
  - makes the current process sleep until seconds seconds have elapsed or a signal arrives which is not ignored
  - return value
    - zero if the requested time has elapsed, or the number of seconds left to sleep

POSIX signal functions

- signal set
  - #include <signal.h>

  - int sigemptyset(sigset_t *set);
  - int sigfillset(sigset_t *set);
  - int sigaddset(sigset_t *set, int signum);
  - int sigdelset(sigset_t *set, int signum);
  - int sigismember(const sigset_t *set, int signum);

POSIX signal functions

- description
  - sigemptyset( )
    - initializes the signal set given by set to empty, with all signals excluded from the set.
  - sigfillset( )
    - initializes set to full, including all signals.
  - sigaddset( ) and sigdelset( )
    - add and delete respectively signal signum from set.
  - sigismember( )
    - tests whether signum is a member of set

POSIX signal functions

- return value
  - sigemptyset, sigfillset, sigaddset and sigdelset return 0 on success and -1 on error
  - sigismember returns 1 if signum is a member of set, 0 if signum is not a member, and -1 on error

POSIX signal functions

- synopsis
  - #include <signal.h>
  - int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
  - int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
  - int sigpending(sigset_t *set);
  - int sigsuspend(const sigset_t *mask);

POSIX signal functions

- description
  - sigaction( )
    - change the action taken by a process on receipt of a specific signal
    - signum
      - specifies the signal and can be any valid signal except SIGKILL and SIGSTOP
    - If act is non-null, the new action for signal signum is installed from act.
    - If oldact is non-null, the previous action is saved in oldact

POSIX signal functions

- sigaction structure

```
struct sigaction {
        void (*sa_handler)(int);
        sigset_t sa_mask;
        int sa_flags;
        void (*sa_restorer)(void);
}
```

- sa_handler
  - specifies the action to be associated with signum and may be SIG_DFL for the default action, SIG_IGN to ignore this signal, or a pointer to a signal handling function

POSIX signal functions

- sa_mask
  - gives a mask of signals which should be blocked during execution of the signal handler
  - In addition, the signal which triggered the handler will be blocked, unless the SA_NODEFER or SA_NOMASK flags are used
- sa_flags
  - specifies a set of flags which modify the behavior of the signal handling process
  - It is formed by the bitwise OR of zero or more of the following:

POSIX signal functions

- SA_NOCLDSTOP
  - If signum is SIGCHLD, do not receive notification when child processes stop
- SA_ONESHOT or SA_RESETHAND
  - Restore the signal action to the default state once the signal handler has been called.
- SA_RESTART
  - Provide behavior compatible with BSD signal semantics by making certain system calls restartable across signals.

POSIX signal functions

        - SA_NOMASK or SA_NODEFER
            - Do not prevent the signal from being received from within its own signal handler
    - sa_restorer
        - is obsolete and should not be used.
- sigprocmask( )
    - change the list of currently blocked signals
    - how
        - SIG_BLOCK
            - The set of blocked signals is the union of the current set and the set argument.

POSIX signal functions

  - SIG_UNBLOCK
    - The signals in set are removed from the current set of blocked signals
    - It is legal to attempt to unblock a signal which is not blocked
  - SIG_SETMASK
    - The set of blocked signals is set to the argument set.
- oldset
  - if non-null, the previous value of the signal mask is stored in oldset
- It is not possible to block SIGKILL or SIGSTOP with the sigprocmask call

POSIX signal functions

- sigpending( )
  - examine the pending signals
  - the signal mask of pending signals is stored in set
- sigsuspend( )
  - temporarily replaces the signal mask for the process with that given by mask and then suspends the process until a signal is received
- return value
  - sigaction, sigprocmask, sigpending and sigsuspend return 0 on success and -1 on error.

sigsetjmp, siglongjmp

- synopsis
  - #include <setjmp.h>
  - int sigsetjmp(sigjmp_buf env, int savemask);
  - void siglongjmp(sigjmp_buf env, int val);
- description
  - if savemask is nonzero then sigsetjmp() also saves the current signal mask in env.
  - when siglongjmp() is called, if the env was saved, then siglongjmp() restores the saved signal mask.