Signals

- A primitive way of doing IPC.
  - Are used to inform processes of asynchronous events.
  - An asynchronous event either terminates a process or is simply being ignored.
  - Arrangements can be made to trap signals.
Signal Generation

A signal is generated when (not a complete list):

- A hardware exception occurs.
- Interrupt or quit from control terminal.
- An alarm timer expires.
- A call to kill().
- Termination of a child process.
Signals Are Software Interrupts

- Each signal has a name.
  - A signal is identified by a named constant (symbolic constant).
  - A set of predefined numbers: 1~MAXSIG.
  - Details in signal.h.

- Signals may not be queued.
  - Implementation-dependent to recognize multiple instances of a signal.

- Order of service is not defined when different signals are pending on a process.
Use of Signals

- Intraprocess
- Interprocesses
  - With the same UID.
- Between kernel to any process.
Signal Status

A signal is said to be:

- generated when the event that causes the signal occurs.
- delivered when the action for a signal is taken.
- pending during the time between the generation of the signal and its delivery.
- blocked if unable to deliver due to a signal mask bit being set for the signal.
Sending Signals

- Signals can be sent to processes at any time.
  - By the kernel, or
  - A call to `kill()` by the user.

- However, signals are checked only when a process is about to return from the kernel mode to the user mode.
Signal Disposition

- Response to a signal, known as the disposition of the signal, can be one of the following:
  - Default action (SIG_DFL)
    - Termination in general.
  - Ignored (SIG_IGN)
    - Never posted to the process.
  - User-defined action
    - Needs a user-defined signal handler, or signal-catching function.
    - Most signals can be caught, or ignored except SIGKILL and SIGSTOP.
The signal() System Call

- **Syntax**
  ```c
  void (*signal(int signo, void (*handler)(int)))(int)
  ```

- **Return value**
  - The previous signal handler if OK, SIG_ERR on error.
  - signal() is said to be unreliable.
    - Signals can get lost.
    - The signal handler is reset to default each time the signal occurs.
    - A process is unable to block signals.
Newer Versions of `signal()`

- `signal()` is replaced by `sigset()` in newer versions of UNIX SV for reliability.
  - Syntax
    ```c
    void (*sigset(int signo, void (*handler)(int)))(int)
    ```
  - Has been further superseded by `sigaction()` in the latest implementations of various versions of UNIX systems.
Signal Masks

- A signal mask is used to block signal delivery.
  - A blocked signal depends on the recipient process to unblock and handle it accordingly.

- A signal mask may be implemented using an integer.
  - Positional – each bit corresponds to one signal.
  - Bit 1’s – the corresponding signals are being blocked.
  - One problem – the number of different signals can exceed the number of bits in an integer.

- A process may query or change its signal mask by a call to sigprocmask().
Signal Sets

- Are used to represent multiple signals the number of which may exceed the number of bits in an integer.

- To manipulate signal sets, a new data type known as \texttt{sigset_t} with the following five predefined functions is specified in POSIX.1:
  - \texttt{sigemptyset()}
  - \texttt{sigfillset()}
  - \texttt{sigaddset()}
  - \texttt{sigdelset()}
  - \texttt{sigismember()}
Some Relevant System Calls/Functions (1/2)

- **kill()**/**raise()**
  - **kill()** sends a signal to a process or a group of process.
  - **raise()** sends a signal to the calling process itself.

- **alarm()**
  - Is used to set a timer that will expire at a specified time in the future.

- **pause()**
  - Is used to suspend the calling process until a signal is received.
Some Relevant System Calls/Functions (2/2)

- **sigpending()**
  - Returns the set of signals that are blocked from delivery and currently pending for the calling process.

- **sigsuspend()**
- **sigsetjmp()**/**siglongjmp()**
More on Signals (1/2)

- Signal mask and dispositions are inherited by the child after fork(). However,
  - Pending alarms are cleared for the child.
  - The set of pending signals for the child is set to the empty set.
More on Signals (2/2)

- All signals are reset to default upon exec() unless ignored.
- The following are inherited by the new program from the calling process:
  - Process signal mask.
  - Pending signals.
- Keyboard interrupts are ignored in background processes.