Signals and Signal Handling

Ref: Chapter 6 of [HGS].

What is a signal?

- A "one-word" message.
- Examples: Green light, stop sign, a referee's gestures in a game.
- A primitive form of inter-process communication in Unix.
- The signal is itself the message.

Signals at the shell command level:

- CTRL-C to interrupt a command. (This sends a signal to the command that is running as well as the shell process.)
- CTRL-Z to stop a process.
- The `kill` command.

Where can signals come from?

- User.
- Kernel: Sends a signal to a process
  (a) if the process does something "bad" (e.g. bus error, floating point exception, segmentation fault) or
  (b) to notify the process of certain events (e.g. an alarm, resizing a window) or
  (c) when the system is being shut down.
- Other processes: A process may send the "kill" signal to another.

Terminology:

- Synchronous signal: Caused by something done by a process (bus error, etc.)
- Asynchronous signal: Caused by events outside the process (e.g. CTRL-C).

What can a process do with a signal?

- Accept the default action (usually abnormal termination).
- Ignore the signal (i.e., wear a "signal proof vest").
- "Catch" the signal (i.e., execute a signal handler).

Some signals:

- Each signal is a symbolic constant denoting an integer value.
- Complete list on pages 127–130 of [HGS].

```
#define  SIGHUP    1
#define  SIGINT    2
#define  SIGQUIT   3
#define  SIGILL    4
#define  SIGFPE    8
#define  SIGKILL   9
#define  SIGSEGV  11
#define  SIGTERM  15
```


Notes:

- For some signals (e.g. `SIGUSR1` and `SIGUSR2`), the default action is to ignore the signal.
- Signals `SIGKILL` and `SIGSTOP` cannot be ignored or caught.

Signal sets:

(a) Data structure:

- `sigset_t` defined in `<signal.h>`. (This header is needed for all signal-related functions.)
- Guaranteed to be large enough to hold all the system defined signals.
- A signal set is used to specify which signals should be blocked from delivery to a process.

(b) Permitted operations:

- Create an empty signal set.

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

- Create a signal set containing all the signals.

```
int  sigfillset  (sigset_t  *sg);
```

- Add a signal to a signal set.

```
int  sigaddset (sigset_t  *sg,
                int  signo);
```

- Delete a signal from a signal set.

```
int  sigdelset (sigset_t  *sg,
                int  signo);
```

- All the four functions above return 0 if no error occurred and −1 otherwise.
- Check membership of a signal in a signal set. (Returns 1 if true and 0 if false.)

```
int sigismember(const sigset_t *sg,
                int  signo);
```


Sample code segment:

```
#include  <signal.h>
sigset_t  set1, set2;

sigemptyset(&set1);
sigaddset(&set1, SIGINT);
sigaddset(&set1, SIGILL);

sigfillset(&set2);
sigdelset(&set2, SIGHUP);
sigdelset(&set2, SIGSEGV);

if (sigismember(&set2, SIGSEGV))
   printf("Yes\n");
else
   printf("NO\n");
```

Blocking a signal:

- Putting a signal on hold while another signal is being handled.
- Signals can't be "stacked" (i.e., only one signal of each type can be outstanding at any time).
- Signals may be lost.

System call `sigaction`:

```
int  sigaction (int signo,
         const struct sigaction *act,
         struct sigaction *oact);
```

- Header: `<signal.h>`.
- Parameter `act` specifies how the signal given by `signo` is handled.
- Fills in the structure pointed to by `oact` with the current setting for the signal.
- Returns 0 if successful and `-1` otherwise.

Structure `sigaction`:

```
struct sigaction {
 void (*sa_handler) (int);
 void (*sa_sigaction) (int,
          siginfo_t *, void *);
 sigset_t  sa_mask;
 int  sa_flags;
};
```

(a) Data member `sa_handler`:

- Function pointer – specifies action to be taken for a signal.
- Possible values: `SIG_DEF`, `SIG_IGN` or the name of a user-defined function. (This function will be executed when the specified signal is received.)
- `SIG_IGN` cannot be used for `SIGSTOP` or for `SIGKILL`.
- The user-defined function gets the signal value as the value parameter.

(b) Data member `sa_sigaction`:

- Also a function pointer. (The corresponding function gets additional inputs.)
- Only one of `sa_handler` and `sa_sigaction` may be used. (The data member `sa_flags` can be used to specify which one is used.)
- Not commonly supported; it is best to avoid using this data member.

(c) Data member `sa_mask`:

- Specifies the set of signals to be blocked when the current signal is handled.
- Normally, when the signal handler is entered, the current signal is also added to the set.

(d) Data member `sa_flags`:

- Some possible values are:
  (i) `SA_RESETHAND`: Reset the handler to `SIG_DFL` upon return from the handler.

  (ii) `SA_NODEFER`: Turn off automatic blocking of the signal being handled.
  (iii) `SA_RESTART`: Restart system calls upon return from the handler.
  (iv) `SA_SIGINFO`: Use the function specified for `sa_sigaction` as the signal handler (rather than `sa_handler`).

Program Example: Handout 17.1.

Other actions:

(a) To ignore `SIGINT`:

```
act.sa_handler = SIG_IGN;
```

(b) To restore previous action:

```
static struct sigaction act, oact;

/* Save old action. */
sigaction(SIGINT, NULL, &oact);
```

```
/* New action. */
act.sa_handler = SIG_IGN;
sigaction(SIGINT, &act, NULL);
    .
    .
/* Restore old action. */
sigaction(SIGINT, &oact, NULL);
```

(c) To exit gracefully when interrupted:

- Define an appropriate function and specify that function as `sa_handler`.