Signals

Signals

- Unexpected/unpredictable asynchronous events
  - floating point error
  - segmentation fault
  - -- control-C (termination request)
  - control-Z (suspend request)
- Events are called interrupts
- When the OS kernel recognizes an event, it sends a signal to the process.
- Normal processes may send signals.

What are signals for?

- Synchronization between processes (e.g. parent and forked children)
- OS communicates hardware events to a process
- Signals are generated by
  - machine interrupts
  - the program itself, other programs or the user.

Signal table

- "man 7 signal" gives description of signals with default actions
- Here are a few…

Signal	Default Action	Comment
SIGINT	Terminate	Interrupt from keyboard
SIGSEGV	Terminate/Dump core	Invalid memory reference.
SIGKILL	Terminate (cannot ignore)	Kill
SIGCHLD	Ignore	Child stopped or terminated.
SIGSTOP	Stop (cannot ignore)	Stop process.
SIGCONT		Continue if stopped.

Sending a signal

- From the command line use

```
kill [-signal] pid [pid]…
```

- If no signal is specified, kill sends the TERM signal to the process.
- signal can be specified by the number or name of the signal.
- Examples:

```
kill -SIGINT 8883
kill -SIGSTOP 78911
kill -9 76433     (9 == KILL)
```


Signalling between processes

- One process can send a signal to another process using the misleadingly named function call:

```
kill(int pid, int sig);
```

- This call sends the signal `sig` to the process `pid`
- Signalling between processes can be used for many purposes:
  - kill errant processes
  - temporarily suspend execution of a process
  - make a process aware of the passage of time
  - synchronize the actions of processes.

Have some fun sending signals with Q 1-3 on the worksheet!

```
kill [-signal] pid
```


Signal table

- Each signal has a default action:
  - Terminate
  - Stop
  - Ignore

Signal	Default Action	Comment
SIGINT	Terminate	Interrupt from keyboard
SIGSEGV	Terminate/Dump core	Invalid memory reference.
SIGKILL	Terminate (cannot ignore)	Kill
SIGCHLD	Ignore	Child stopped or terminated.
SIGSTOP	Stop (cannot ignore)	Stop process.
SIGCONT		Continue if stopped.

Default actions

- The default action can be changed by installing a signal handler using the `sigaction()` function. The exceptions are SIGKILL and SIGSTOP.
- Note: You don't explicitly call the signal handler – OS will transfer control to this function when signal occurs.

Signal handlers

- void handler (int signum) { ... }
- Defines action(s) to be taken when receiving a particular signal.
- Is being called with the number of the signal that triggered it as an argument

`sigaction()`

- Install a signal handler, `act`, for the signal `sig`.

```
int sigaction(int sig,
              const struct sigaction *act,
              struct sigaction *oldact);
```

- Struct defined in <signal.h> to pass in for `act`.

```
struct sigaction {
  /* SIG_DFL, SIG_IGN, or pointer to function */
  void (*sa_handler)(int);
  sigset_t sa_mask; /*Signals to block during handler*/
  int sa_flags; /* flags and options */
};
```

- You may come across various extensions, including another field in the sigaction struct for a function to catch signals.

Groups of signals

- Signal masks are used to store the set of signals that are currently blocked.
- Operations on sets of signals:

```
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember(const sigset_t *set,
                int signo);
```

Work through Q5-9 to make your program sing when it receives a SIGUSR1 signal!

Create a sigset_t and pass it to sigaction, such that SIGINT will be ignored while singing (Q10-11)!