Signals

- http://www.csl.mtu.edu/cs4411.choi/www/Resource/signal.pdf
- The following slides are from
  - www.cs.fsu.edu/~xyuan/coßp4610/lecture_7_osinterface5.ppt

IPC Mechanism

- IPC mechanism: Signal
  - Tells a process that some event occurs. It occurs
    - In the kill command.
      - Try ‘kill –l’
      - ‘kill –s INT ####*(pid)*’
    - When Ctrl-C is typed (SIGINT).
    - When Ctrl-\ is typed (SIGQUIT)
    - When a child exits (SIGCHLD to parent)
    - When a timer expires
    - When a CPU execution error occurs
    - ……
  - A form of inter-process communication.

Available Actions and Signals

- When a process receives a signal, it performs one of the following three options:
  - Ignore the signal
  - Perform the default operation
  - Catch the signal (perform a user defined operation).
- Some commonly used signals:
  - SIGABRT, SIGALRM, SIGCHLD, SIGHUP, SIGINT, SIGUSR1, SIGUSR2, SIGTERM, SIGKILL, SIGSTOP, SIGSEGV, SIGILL
  - All defined in signal.h

Processing Signals

- Processing signals:
  - similar to an interrupt (software interrupt)
  - when a process receives a signal:
    - pause execution
    - call the signal handler routine
    - Continue execution
  - Signal can be received at any point in the program.
  - Most default signal handlers will terminate the program.
  - You can change the way your program responses to signals.
    - E.g Make Ctrl-C have no effect.

Simplified Signal Interface

- ANSI C signal function to change the signal handler
  - syntax:
    - #include <signal.h>
    - void (*signal(int sig, void (*disp)(int)))(int);
    - Alternately
      - typedef void (*sighandler)(int);
      - sighandler signal(int sig, sighandler disp);
  - semantics:
    - sig -- signal (defined in signal.h)
    - disp: SIG_IGN, SIG_DFL or the address of a signal handler.
    - Handler may be reset to SIG_DFL after one invocation
      - AT&T UNIX does the reset, BSD UNIX does not do the reset
      - Using signal with a handler function isn't portable; use sigaction(2)
      - How to get continuous coverage?
      - Still have problems – may lose signals

Question

A call to *signal()* in C

A. Carries a payload of 128 bytes

B. Always causes the process to terminate

[✓ C. Establishes the behavior for the process when it receives a particular signal]

D. Catches the signal for later use by the C Standard Library

Today's number is 54,321

A Non-Portable Example Using signal(2)

```
#include <signal.h>

void sigcatcher(int);
void sigexiter(int);

int main(int argc, char *argv[]) {
        signal(SIGINT, sigcatcher);   // control-c
        signal(SIGQUIT, sigcatcher);  // control-\
        signal(SIGTERM, sigexiter);   // “kill process-id”

        printf("My process id is %d\n", getpid());
        while (1) {
                printf("Waiting 30 seconds on a signal\n");
                sleep(30);
        }
}
void sigcatcher(int s) {
        signal(s, sigcatcher);
        printf("Caught signal %d\n", s);
}
void sigexiter(int s) {
        printf("Exiting on signal %d\n", s);
        exit(1);
}
```

At Least Two Things Wrong

```
#include <signal.h>

void sigcatcher(int);
void sigexiter(int);

int main(int argc, char *argv[]) {
        signal(SIGINT, sigcatcher);   // control-c
        signal(SIGQUIT, sigcatcher); // control-\
        signal(SIGTERM, sigexiter);   // "kill process-id"

        printf("My process id is %d\n", getpid());
        while (1) {
                printf("Waiting 30 seconds on a signal\n");
                sleep(30);
        }
}
void sigcatcher(int s) {
        signal(s, sigcatcher);
        printf("Caught signal %d\n", s);
}
void sigexiter(int s) {
        printf("Exiting on signal %d\n", s);
        exit(1);
}
```

Printf() isn't "signal safe" and can cause a deadlock in a signal handler

write() is OK though

This resets the signal handler if the OS is one that resets it to default

There is a race condition if another signal comes in just before this call – the signal will cause the default action instead of calling the handler

Blocking Temporarily Suspends Signal Actions

- Block/unblock signals
  - Manipulate signal sets
    - #include <signal.h>

      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);
  - Manipulate signal mask of a process
    - int sigprocmask(int how, const sigset_t *set, sigset_t *oset);
    - How: SIG_BLOCK, SIG_UNBLOCK, SIG_SETMASK

Example of Deferring a Signal

- For a critical region where you don’t want a certain signal to be deferred, the program will look like:

```
#include <signal.h>
sigset_t newmask, oldmask;
sigemptyset(newmask);
sigaddset(newmask, SIGINT);

sigprocmask(SIG_BLOCK, &newmask, &oldmask);
……. /* critical region */
sigprocmask(SIG_SETMASK, &oldmask, NULL);
```

- sigaction
  - Supersedes the signal function
  - #include <signal.h>
  - int sigaction(int signo, const struct sigaction *act, struct sigaction *oact)

```
struct sigaction {
  void (*sa_handler)(); /* signal handler */
  sigset_t sa_mask;  /*additional signal to be block */
  int sa_flags;        /* various options for handling signal */
};
```

Send a Signal

- Kill:
  - Send a signal to a process

  - #include <signal.h>
  - #include <sys/types.h>
  - int kill(pid_t pid, int signo);
    - Pid > 0, normal
    - Pid == 0, all processes whose group ID is the current process' group ID.
    - Pid <0, all processes whose group ID = |pid|

Which of the following is NOT a characteristic of C signals?

A. Signals can be deferred until the program is ready to process them

B. A process can send a signal to another process or to itself

C. The receipt of a signal by a program can be ignored, can be processed according to a default rule, or can cause a function to be called

[✓ D. Signals are only processed at the entry and exits of C functions]

Signalling Example: Counting Child Processes

- Signalling example: keeping track of number of child processes in a shell: when a process exits, it sends a SIGCHLD to its parent.

Example: First Half

```
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>

int numofchild = 0;

// If we catch a SIGCHLD, decrement the number of active children
void sigchildhandler() {
  numofchild --;
  write(1, "Child exited\n", sizeof("Child exited\n"));
}

int main() {
  char cmd[1000], buf[1000], *argv[2];
  struct sigaction abc;
  int pid;

  // Set a sigchildhandler() to catch SIGCHLD signals
  abc.sa_handler = sigchildhandler;
  sigemptyset(&abc.sa_mask);
  abc.sa_flags = 0;
  sigaction(SIGCHLD, &abc, NULL);
```

Signal catcher

Install the signal catcher

Example: Second Half

Read an input command

If “quit”, exit if no children active

Fork and execute a child

If the fork succeeded increment the child count

```
while(1) {
        // Read in a command name to execute
        printf("<%d>", numofchild);
        fflush(stdout);
        while(fgets(buf, 100, stdin) == NULL)
            ;
        sscanf(buf, "%s", cmd);

        // Command is "quit"; exit if all children are complete
        if (strcmp(cmd, "quit") == 0) {
            if (numofchild == 0)
                exit(0);
            printf(”There are still %d children.\n", numofchild);
        // Execute a child process running the specified command
        } else if ((pid = fork()) == 0) {
            argv[0] = cmd;
            argv[1] = NULL;
            execv(argv[0], argv);
            exit(0);
        // If fork() doesn't fail, increment the number of children
        } else if (pid != -1)
            numofchild ++;
    }
} /* example6.c */
```