Basic Signal Programming

Adding manpower to a late software project, makes it later.

*Alan TuringFrederick P. Brooks Jr.*
Author of The Mytthucal Man-Month

What is a signal?

❑ **Signals are generated when an event occurs that requires attention. It can be considered as a software version of an interrupt.**

❑ **Signal Sources:**

  ❖ **Hardware** - division by zero

  ❖ **Kernel** – notifying an I/O device for which a process has been waiting is available

  ❖ **Other Processes** – a child notifies its parent that it has terminated

  ❖ **User** – key press (*i.e.*, `Ctrl-C`)

What signals are available?

❑ **Signal names are defined in `signal.h`**

❑ **The following are Unix examples:**

    ❖ `SIGALRM` **– alarm clock**

    ❖ `SIGBUS` **– bus error**

    ❖ `SIGFPE` **– floating point arithmetic exception**

    ❖ `SIGINT` **– interrupt (*i.e.*, `Ctrl-C`)**

    ❖ `SIGQUIT` **– quit (*i.e.*, `Ctrl-\`)**

    ❖ `SIGTERM` **– process terminated**

    ❖ `SIGUSR1` **and** `SIGUSR2` **– user defined signals**

❑ **You can ignore *some* signals**

❑ **You can also catch and handle some signals.**
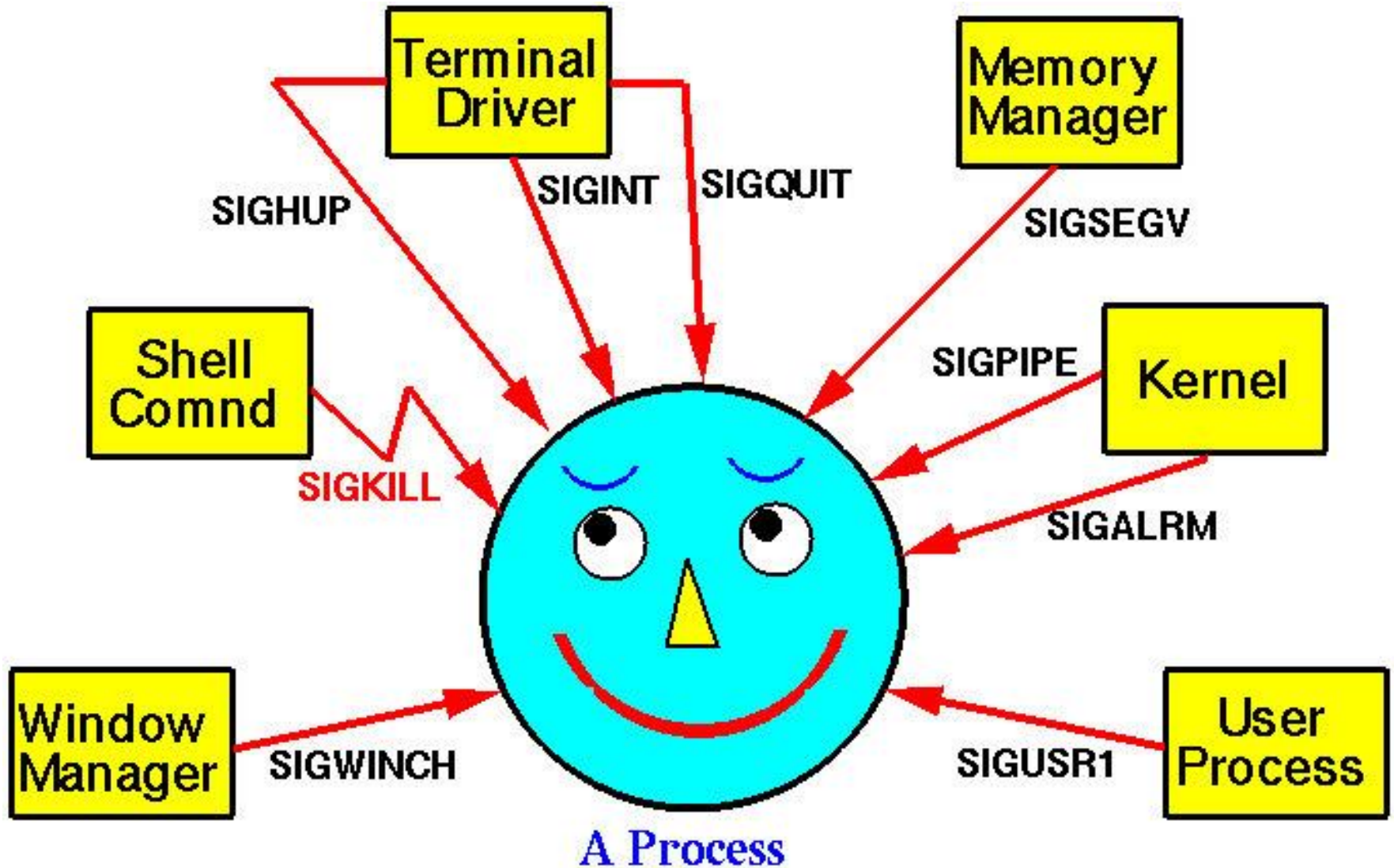
Function `signal()`

```
void (*signal(int, void (*)(int)))(int);
```

❑ `signal()` **is a function that accepts *two* arguments and returns a pointer to a function that takes one argument, the signal handler, and returns nothing. If the call fails, it returns** `SIG_ERR`**.**

❑ **The arguments are**

❖ **The first is an integer (*i.e.*, `int`), a *signal name*.**

❖ **The second is a function that accepts an `int` argument and returns nothing, the *signal handler*.**

❖ **If you want to ignore a signal, use `SIG_IGN` as the second argument.**

❖ **If you want to use the default way to handle a signal, use `SIG_DFL` as the second argument.**

5

Examples

❑ **The following ignores signal `SIGINT`**

```
signal(SIGINT, SIG_IGN);
```

❑ **The following uses the default way to handle `SIGALRM`**

```
signal(SIGALRM, SIG_DFL);
```

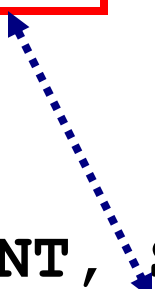❑ **The following installs function `INThandler()` as the signal handler for signal `SIGINT`**

```
signal(SIGINT, INThandler);
```

Install a Signal Handler: 1/2

```c
#include <stdio.h>
#include <signal.h>

void   INThandler(int);

void main(void)
{
    if (signal(SIGINT, SIG_IGN) != SIG_IGN)
        signal(SIGINT, INThandler);
    while (1)
        pause();
}
```

Install a Signal Handler: 2|2

```
void   INThandler(int sig)
{
    char c;
    signal(sig, SIG_IGN);
    printf("Ouch, did you hit Ctrl-C?\n",
            "Do you really want to quit [y/n]?");
    c = getchar();
    if (c == 'y' || c = 'Y')
        exit(0);
    else
        signal(SIGINT, INThandler);
}
```

ignore the signal first

reinstall the signal handler

Here is the procedure

1. **Prepare a function that accepts an integer, a signal name, to be a signal handler.**

2. **Call `signal()` with a signal name as the first argument and the signal handler as the second.**

3. **When the signal you want to handle occurs, *your* signal handler is called (by the system) with the argument the signal name that just occurred.**

4. **Two important notes:**
   a. **You might want to ignore that signal in your handler**
   b. **Before returning from your signal handler, don't forget to re-install it.**

9

Handling Multiple Signal Types: 1/2

❑ **You can install multiple signal handlers:**

```
signal(SIGINT, INThandler);
signal(SIGQUIT, QUIThandler);

void   INThandler(int sig)
{
    // SIGINT handler code
}


void   QUIThandler(int sig)
{
    // SIGQUIT handler code
}
```

Handling Multiple Signal Types: 2/2

❑ **Or, you can use one signal handler and install it multiple times**

```
signal(SIGINT, SIGhandler);
signal(SIGQUIT, SIGhandler);

void SIGhandler(int sig)
{
    switch (sig) {
        case SIGINT:    // code for SIGINT
        case SIGQUIT:   // code for SIGQUIT
        default:        // other signal types
    }
}
```

Handling Multiple Signal Types Example: 1/4

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

#define  MAX_i       10000
#define  MAX_j       20000
#define  MAX_SECOND  (2)

void  INThandler(int);
void  ALARMhandler(int);

int   SECOND, i, j;
/* see next page */
```

Handling Multiple Signal Types Example: 2/4

```c
void INThandler(int sig)
{
    char c;
    signal(SIGINT, SIG_IGN);
    signal(SIGALRM, SIG_IGN);
    printf("INT handler: i = %d and j = %d\n", i, j);
    printf("INT handler: want to quit [y/n]?");
    c = tolower(getchar());
    if (c == 'y') {
        printf("INT handler: done"); exit(0);
    }
    signal(SIGINT, INThandler);
    signal(SIGALRM, ALARMhandler);
    alarm(SECOND);
} /* see next page */
```

This is a Unix system call 13

Handling Multiple Signal Types Example: 3/4

```
void  ALARMhandler(int sig)
{
   signal(SIGINT, SIG_IGN);
   signal(SIGALRM, SIG_IGN);
   printf("ALARM handler: alarm signal received\n");
   printf("ALARM handler: i = %d and j = %d\n", i, j);
   alarm(SECOND);
   signal(SIGINT, INThandler);
   signal(SIGALRM, ALARMhandler);
} /* see next page */
```

set alarm clock to SECOND *seconds*

Handling Multiple Signal Types Example: 4/4

```
void main(int argc, char *argv[])
{
    long sum;

    SECOND = abs(atoi(argv[1]));
    signal(SIGINT, INThandler);
    signal(SIGALRM, ALARMhandler);
    alarm(SECOND);
    for (i = 1; i <= MAX_i, i_++) {
        sum = 0;
        for (j = 1; j <= MAX_j; j++)
            sum += j;
    }
    printf("Computation is done.\n\n");
}
```

Raise a Signal within a Process: 1/2

❑ **Use ANSI C function `raise()` to "raise" a signal**

```
int   raise(int sig);
```

❑ **`raise()` returns non-zero if unsuccessful.**

```
#include <stdio.h>
#include <signal.h>

long  pre_fact, i;

void SIGhandler(int);

void SIGhandler(int sig)
{
    printf("\nReceived a SIGUSR1 signal %ld! = %ld\n",
            i-1, pre_fact);
} /* see next page */
```
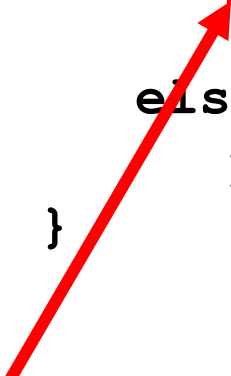
Check here if it is a `SIGUSR1`!

16

Raise a Signal within a Process: 2/2

```c
void  main(void)
{
    long fact;
    signal(SIGUSR1, SIGhandler);
    for (prev_fact=i=1; ; i++, prev_fact = fact) {
        fact = prev_fact * i;
        if (fact < 0)
            raise(SIGUSR1);
        else if (i % 3 == 0)
            printf("    %ld = %ld\n", i, fact);
    }
}
```

Assuming an integer overflow will wrap around!

Send a Signal to a Process

❑ **Use Unix system call `kill()` to send a signal to another process:**

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

❑ **`kill()` sends the `sig` signal to process with ID `pid`.**

❑ **So, you must find some way to know the process ID of the process a signal is sent to.**

Kill Example: process-a 1/3

```
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

void SIGINT_handler(int);
void SIGQUIT_handler(int);

int     ShmID;
pid_t   *ShmPTR;

/* see next page */
```

used to save shared memory ID

my PID will be stored here

```c
void  main(void)
{
    int     i;
    pid_t  pid = getpid();
    key_t  MyKey;

    signal(SIGINT, SIGINT_handler);
    signal(SIGQUIT, SIGQUIT_handler);
    MyKey = ftok("./", 'a');
    ShmID = shmget(MyKey, sizeof(pid_t), IPC_CREAT|0666);
    ShmPTR = (pid_t *) shmat(shmID, NULL, 0);
    *ShmPTR = pid;
    for (i = 0;  ; i++) {
        printf("From process %d: %d\n", pid, i);
        sleep(1);
    }
} /* see next page */
```

20

Kill Example: *process-a 3/3*

```
void SIGINT_handler(int sig)        use Ctrl-C to interrupt
{
    signal(sig, SIG_IGN);
    printf("From SIGINT: got a Ctrl-C signal %d\n", sig);
    signal(sig, SIGINT_handler);
}


void SIGQUIT_handler(int sig)    use Ctrl-\ to kill this program
{
    signal(sig, SIG_IGN);
    printf("From SIGQUIT: got a Ctrl-\\ signal %d\n", sig);
    printf("From SIGQUIT: quitting\n");
    shmdt(ShmPTR);
    shmctl(ShmID, IPC_RMID, NULL);
    exit(0);
}
```

```
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

Void main(void)
{
    pid_t  pid, *ShmPTR;
    key_t  MyKey;
    int    ShmID;
    char   c;

    MyKey = ftok("./", 'a');
    ShmID = shmget(MyKey, sizeof(pid_t), 0666);
    ShmPTR = (pid_t *) shmat(ShmID, NULL, 0);
    pid    = *ShmPTR;
    shmdt(ShmPTR);   /* see next page */
```

detach the shared memory after taking the `pid`

22

Kill Example: process-b 2/2

```
while (1) {
    printf("(i for interrupt or k for kill)? ");
    c = getchar();
    if (c == 'i' || c == 'I') {
        kill(pid, SIGINT);
        printf("A SIGKILL signal has been sent\n");
    }
    else if (c == 'k' || c == 'K') {
        printf("About to sent a SIGQUIT signal\n");
        kill(pid, SIGQUIT);
        exit(0);
    }
    else
        printf("Wrong keypress (%c).  Try again!\n", c);
}
}
```

You can kill process-a from within process-b!

The End