Basic Signal Programming

What is a *signal*?

- Signals are generated when an event occurs that requires attention. It can be considered as a software version of a hardware 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`)

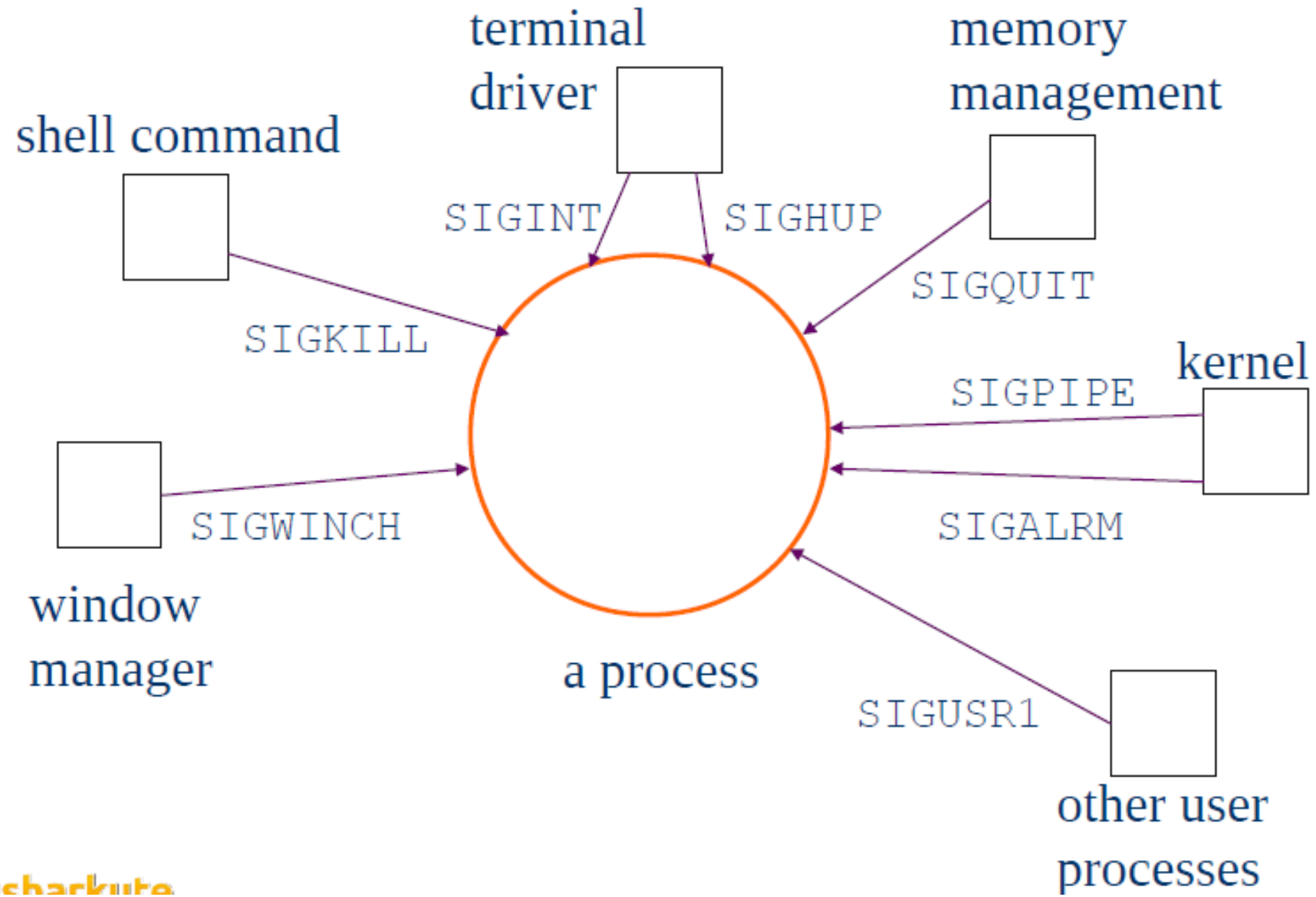
What is a Signal?

- A signal is an asynchronous event which is delivered to a process.
- Asynchronous means that the event can occur at any time may be unrelated to the execution of the process.
- Signals are raised by some error conditions, such as memory segment violations, floating point processor errors, or illegal instructions.
  - e.g. user types ctrl-C, or the modem hangs

What signals are available?

- Signal names are defined in `signal.h`
- The following are 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.

Signal Sources



POSIX predefined signals

- SIGALRM: Alarm timer time-out. Generated by alarm( ) API.
- SIGABRT: Abort process execution. Generated by abort( ) API.
- SIGFPE: Illegal mathematical operation.
- SIGHUP: Controlling terminal hang-up.
- SIGILL: Execution of an illegal machine instruction.
- SIGINT: Process interruption. Can be generated by <Delete> or <ctrl_C> keys.
- SIGKILL: Sure kill a process. Can be generated by
  - "kill -9 <process_id>" command.
- SIGPIPE: Illegal write to a pipe.
- SIGQUIT: Process quit. Generated by <crtl_\> keys.
- SIGSEGV: Segmentation fault. generated by de-referencing a NULL pointer.

POSIX predefined signals

- SIGTERM: process termination. Can be generated by
  - “kill <process_id>” command.
- SIGUSR1: Reserved to be defined by user.
- SIGUSR2: Reserved to be defined by user.
- SIGCHLD: Sent to a parent process when its child process has terminated.
- SIGCONT: Resume execution of a stopped process.
- SIGSTOP: Stop a process execution.
- SIGTTIN: Stop a background process when it tries to read from from its controlling terminal.
- SIGTSTP: Stop a process execution by the control_Z keys.
- SIGTTOUT: Stop a background process when it tries to write to its controlling terminal.

Actions on signals

- Process that receives a signal can take one of three action:
- Perform the system-specified default for the signal
  - notify the parent process that it is terminating;
  - generate a core file; (a file containing the current memory image of the process)
  - terminate.
- Ignore the signal
  - A process can do ignoring with all signal but two special signals: SIGSTOP and SIGKILL.
- Catch the Signal
  - When a process catches a signal, except SIGSTOP and SIGKILL, it invokes a special signal handing routine.

Example of signals

- **User types Ctrl-c**
  - Event gains attention of OS
  - OS stops the application process immediately, sending it a 2/SIGINT signal
  - Signal handler for 2/SIGINT signal executes to completion
  - Default signal handler for 2/SIGINT signal exits process
- **Process makes illegal memory reference**
  - Event gains attention of OS
  - OS stops application process immediately, sending it a 11/SIGSEGV signal
  - Signal handler for 11/SIGSEGV signal executes to completion
  - Default signal handler for 11/SIGSEGV signal prints “segmentation fault” and exits process

Signal Number

Send signals via commands

- **kill Command**
  - **`kill -signal pid`**
    - Send a signal of type signal to the process with id pid
    - Can specify either signal type name (-SIGINT) or number (-2)
  - *No signal type name or number specified => sends 15/SIGTERM signal*
- Default 15/SIGTERM handler exits process
  - Better command name would be sendsig
- Examples
  - **`kill –2 1234`**
  - **`kill -SIGINT 1234`**
    - Same as pressing Ctrl-c if process 1234 is running in foreground

Signal Concepts

- Signals are defined in <signal.h>
- **man 7 signal** for complete list of signals and their numeric values.
- **kill –l** for full list of signals on a system.
- 64 signals. The first 32 are traditional signals, the rest are for real time applications

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.

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

```
#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 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.

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

```
#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
```

Handling Multiple Signal Types Example: 2/4

```
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);
}
```

This is a Unix system call

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);
}
```

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.

Check here if it is a `SIGUSR1`*!*

```
#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);
}
```

Raise a Signal within a Process: 2/2

```
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.

The Unix `Kill` Command

- The `kill` command can also be used to send a signal to a process:

```
kill -l  /* list all signals */
kill -XXX pid1  pid …… pid
```

- In the above `XXX` is the signal name without the initial letters `SIG`.
- `kill -KILL 1357 2468` kills process 1357 and 2468.
- `kill -INT 6421` sends a `SIGINT` to process 6421.
- A `kill` without a signal name is equivalent to `SIGTERM`.
- `-9` is equal to `-SIGKILL`.

Example:

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void ohh(int sig)
{
    printf("Ohh! - I got signal %d\n", sig);
    (void) signal(SIGINT, SIG_DFL);
}
int main()
{
    (void) signal(SIGINT, ohh);
    while(1)
    {
        printf("Hello World!\n");
        sleep(1);
    }
  return 0;
}
```

Example:2

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
void error(int sig)
{
     printf("Ohh! its a floating point error...\n");
     (void) signal(SIGFPE, SIG_DFL);
}
int main()
{
     (void) signal(SIGFPE, error);
     int a = 12, b = 0, result;
     result = a / b;
     printf("Result is : %d\n",result);
     return 0;
}
```