DON’T DO THIS AT HOME

Seriously don’t...
Sebastian Porling, Aron Hansen Berggren
- with inspiration from previous years presentations.

Signals

- Asynchronous notifications sent to a process to inform that a certain event occurred
- Causes a process to stop executing and handle the signal that has been received
- Type in ‘man 7 signal’ in the shell to list all signals and their description
- There are `SIGINT`, `SIGTERM`, `SIGKILL`, `SIGSEGV` and more

Sending signals

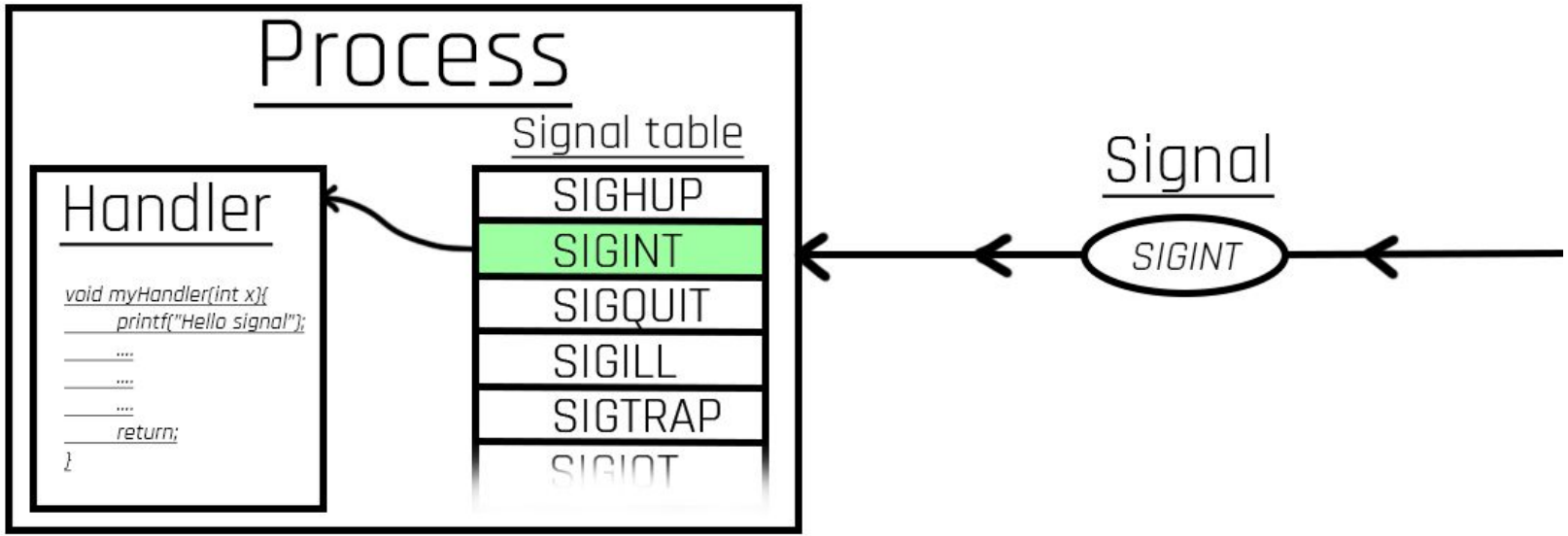
From the shell:

- `CTRL-C` (sends `SIGINT`)
- `kill [-signal] [pid]`

Using system calls

- `kill([pid], [signal])`

Signals


Process
Signal table
Handler
void myHandler(int x){
printf("Hello signal");
....
....
....
return;
}
SIGHUP
SIGINT
SIGQUIT
SIGILL
SIGTRAP
Signal
SIGINT

IDT (Interrupt Descriptor Table)

- Contains pointers to handlers for interrupts
- Every OS has one IDT
- Every process has its own Signal Table
- 256 Entries

Default Signal Handlers

- Each entry has a default handler (`SIGINT` terminates the process)
- `SIGKILL` and `SIGSTOP` cannot be changed by the user. They immediately perform the action of terminating or stopping the process

Sigaction - register a new handler

- Adds a new handler to a specific row in the IDT for a process
- Called using the following syntax:

```
sigaction(signal, handler_function, oldaction)
```

Context information in the handler

The context can be received by the handler function. It can receive **up to three** arguments.

```
handler_function(int signal [, siginfo_t info, void* u_context] )
```

`siginfo_t` - Signal information (pod of process etc.)

`u_context` - The execution context. Things such as program counter and other registers.

man pages, MAN PAGES

Probably the most useful command!

- `man 7 signal` ⟶ Overview and explanation of different signals
- `man sigaction` ⟶ Examine and change a signal action
- `man getcontext` ⟶ Get the user context (useful for section 5)

And the very useful command (explained in the assignment):

- `kill -l` ⟶ List all signals

Exam Questions

2017-06-07

A simple way to kill a program is to hit `CTRL-C`. If we write a program we might not want to die or we might want to do some last operations before terminating. What mechanisms should we use in our program to handle this?

2017-06-07

A simple way to kill a program is to hit CTRL-C. If we write a program we might not want to die or we might want to do some last operations before terminating. What mechanisms should we use in our program to handle this?

Answer: We should create a **signal handler**, a procedure that we will register for a specific signal, in this case **`SIGINT`**. When `CTRL-C` is pressed a **`SIGINT`** will be sent to the process and thus our **signal handler** will be executed.

2017-01-14

What does the IDT (Interrupt Descriptor Table) contain and what happens when a user process executes the instruction `INT` (x86 assembler). Give a short description.

2017-01-14

What does the IDT (Interrupt Descriptor Table) contain and what happens when a user process executes the instruction INT (x86 assembler). Give a short description.

Answer: The IDT is set up by the kernel and contains pointers to procedures that should be executed by different interrupts. When a user process executes for example `INT x80` the process **enters kernel mode** and jumps to the procedure indicated by position `x80` (hex).

2017-06-07

In a x86-architecture we have an IDT that is used when we implement among other things system calls. What does the operating system have to add to the IDT in order to make a system call possible?

2017-06-07

In a x86-architecture we have an IDT that is used when we implement among other things system calls. What does the operating system have to add to the IDT in order to make a system call possible?

Answer: The OS stores a pointer at a specified position in the table(`0x80`) to a procedure that handles all system-signals. When a user process executes `INT 0x80` the stored procedure will be put in charge, and will be executed in **Kernel Mode**.

2017-12-18

In the processor 80286, that was launched in 1982, Intel had added a privileged instruction **LIDT** (Load Interrupt Descriptor Table). What does it mean that the instruction is privileged and why does this instruction need to be privileged?

2017-12-18

In the processor 80286, that was launched in 1982, Intel had added a privileged instruction **LIDT** (Load Interrupt Descriptor Table). What does it mean that the instruction is privileged and why does this instruction need to be privileged?

Answer: A privileged instruction can only be executed in kernel mode. The instruction will set a pointer to a table (IDT) that describes what should be done for each exception. This is nothing that a user process should be allowed to do.