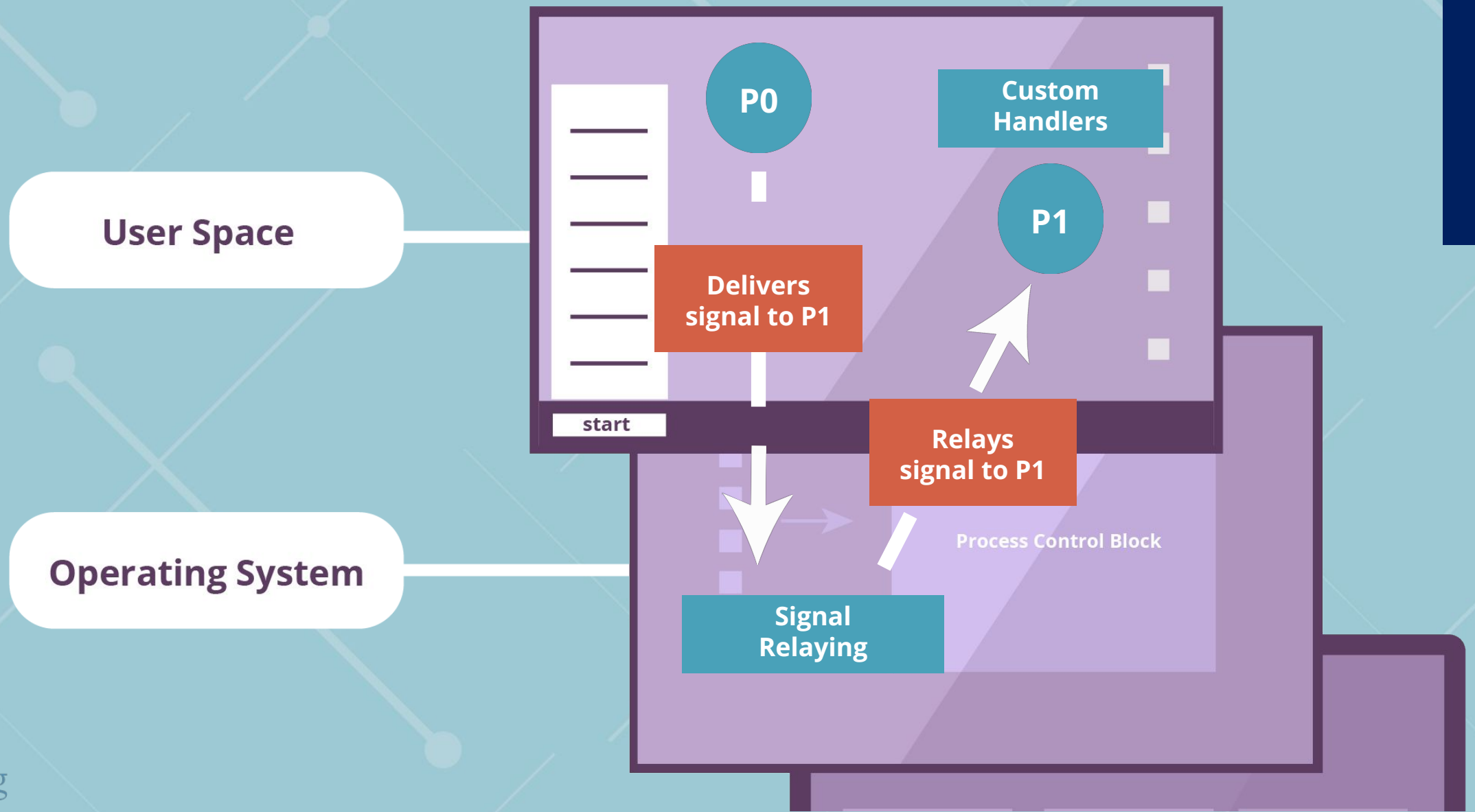
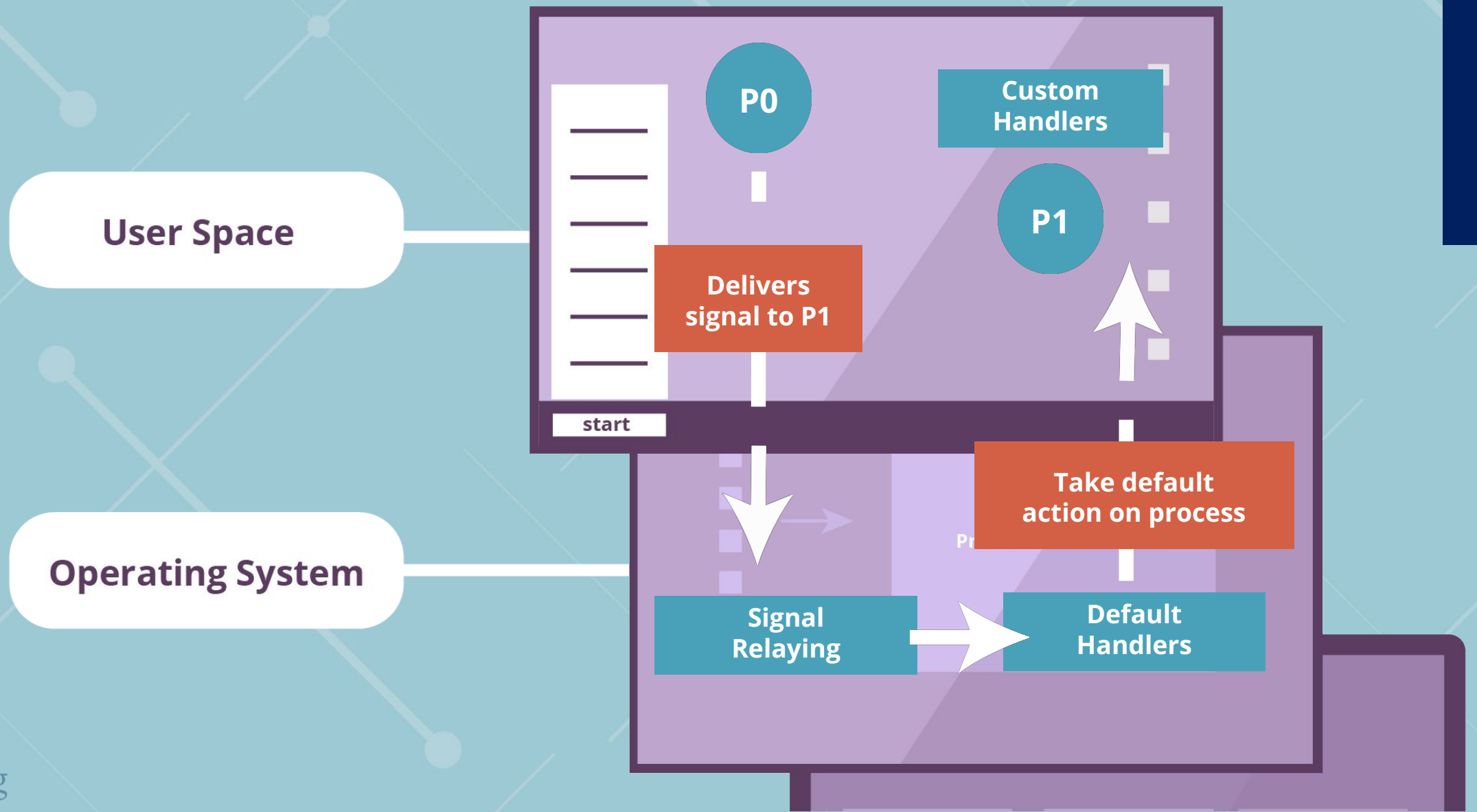
CIS 3800 Recitation 2: Software Signals

Penn OS Course

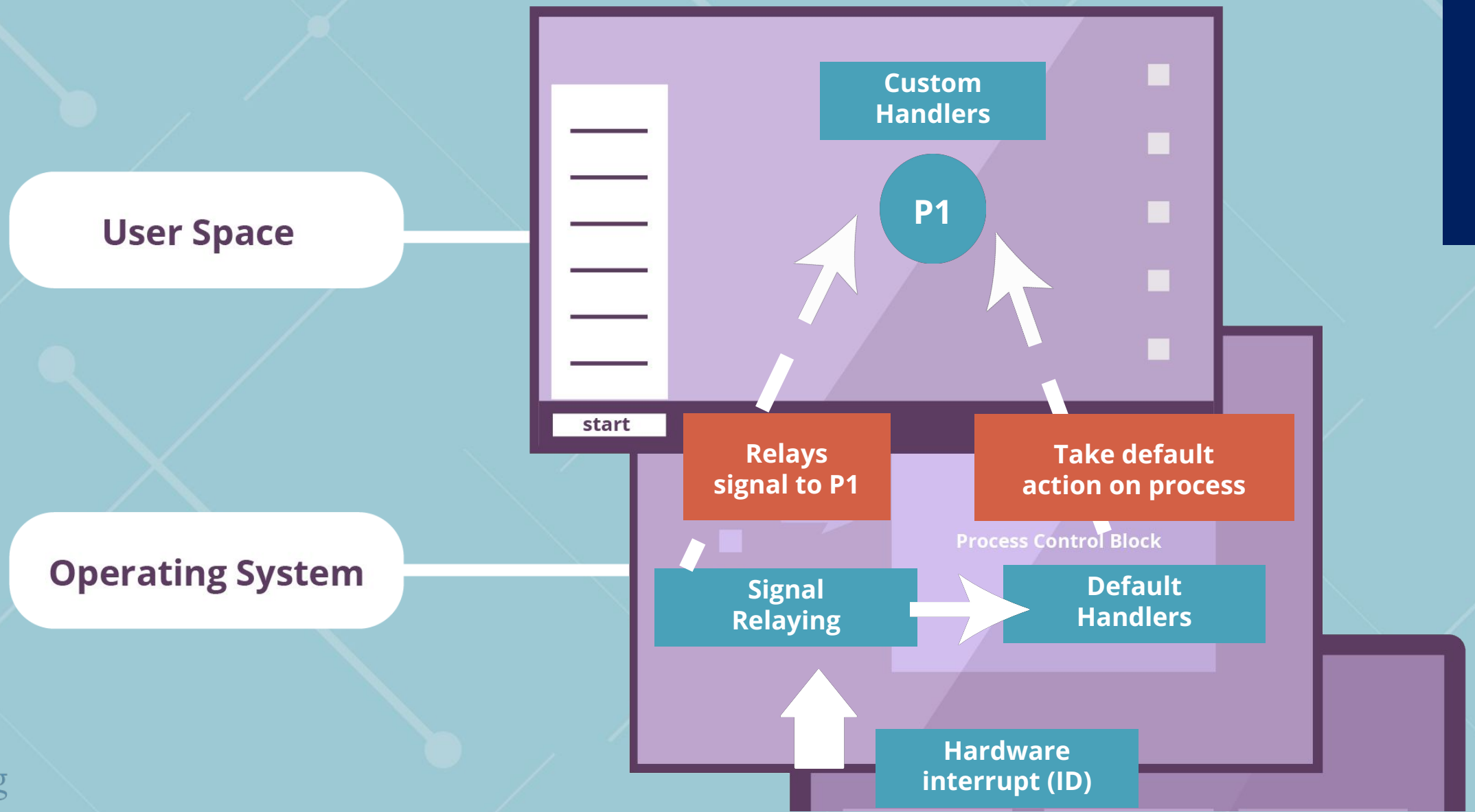
Inter-Process Signaling



Inter-Process Signaling



Inter-Process Signaling



Terminology: Signal

- Asynchronous software notification to a process of an event
- "*Software Interrupt*" but can only be initiated by another process, not necessarily by the OS
- Simplest form of inter-process communication
- Each signal has a symbolic name
  - Starts with `SIG*`
  - Defined in `signals.h`

Terminology: Signal Handler

- Used by the process that receives the signal to handle the signal
- May overwrite default action

Example Signals

Signal	Default Action	Description
SIGALRM	Terminate	Alarm clock
SIGCHLD	Ignore	Child process terminated/stopped or continued
SIGINT	Terminate	Terminal interrupt signal (Ctrl-C)
SIGKILL	Terminate	Kill (`kill -9 ...`) Cannot be caught or ignored
SIGTERM	Terminate	Graceful termination request Can be caught or ignored
SIGUSER1 / SIGUSER2	Terminate	User-defined signal 1 and 2
SIGFPE	Terminate (core dump)	Erroneous arithmetic operation
SIGSEGV	Terminate (core dump)	Invalid memory access
SIGSTOP / SIGCONT	Stop / Continue	Stop/continue execution Stop cannot be caught or ignored

Example Terminal Commands

Terminal Cmd	Default Action	Description
Ctrl + C	Sends SIGINT signal	Terminal interrupt signal
Ctrl + \	Sends SIGQUIT signal	Similar to SIGINT, terminates process and produces a core dump, just like a program error signal. You can think of this as a program error condition “detected” by the user.
Ctrl + Z	Sends SIGTSTP signal	Interactive stop signal. Unlike SIGSTOP, this can be handled and ignored.
Ctrl + D	Sends EOF character	Sends EOF character, typically closes File Descriptors/Pipe ends

- Explore the man page for stty(1) to learn more about terminal commands and changing behavior of terminal commands

Wait Macros

Macro	Description
WIFEXITED(wstatus)	returns true if the child terminated normally, that is, by calling exit(3) or _exit(2), or by returning from main().
WEXITSTATUS(wstatus)	returns the exit status of the child. This consists of the least significant 8 bits of the status argument that the child specified in a call to exit(3) or _exit(2) or as the argument for a return statement in main(). This macro should be employed only if WIFEXITED returned true.
WIFSIGNALED(wstatus)	returns true if the child process was terminated by a signal.
WTERMSIG(wstatus)	returns the number of the signal that caused the child process to terminate. This macro should be employed only if WIFSIGNALED returned true.
WCOREDUMP(wstatus)	returns true if the child produced a core dump. This macro should be employed only if WIFSIGNALED returned true.
WIFSTOPPED(wstatus)	returns true if the child process was stopped by delivery of a signal; this is possible only if the call was done using WUNTRACED or when the child is being traced.
WSTOPSIG(wstatus)	returns the number of the signal which caused the child to stop. This macro should be employed only if WIFSTOPPED returned true.
WIFCONTINUED(wstatus)	returns true if the child process was resumed by delivery of SIGCONT.

First Example

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

void sig_handler(int signo)
{
  if (signo == SIGINT) {
    printf("Receives SIGINT\n");
  }
}

int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR){
    printf("Unable to catch SIGINT\n");
  }

  while(1) {
    sleep(1);
  }

  return 0;
}
```

Custom Handler

Register SIGINT Handler

Sleep Loop

Ctrl-C no longer kills the process!

You can send a kill signal from another terminal or use Ctrl-Z to stop the process

Attempting to Overwrite Default Handlers

```
int main(void)
{
  if (signal(SIGINT, sig_handler) == SIG_ERR){
    printf("Unable to catch SIGINT\n");
  }

  if (signal(SIGKILL, sig_handler) == SIG_ERR){
    printf("\ncan't catch SIGKILL\n");
  }

  if (signal(SIGSTOP, sig_handler) == SIG_ERR){
    printf("\ncan't catch SIGSTOP\n");
  }

  while(1) {
    sleep(1);
  }

  return 0;
}
```

Cannot overwrite default handlers!

Example with Alarms

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

void sig_handler(int signo)
{
  if (signo == SIGALRM) {
    printf("Receives SIGALRM.\n");
  }
}

int main(void)
{
  if (signal(SIGALRM, sig_handler) == SIG_ERR){
    printf("Unable to catch SIGINT\n");
  }

  alarm(15);

  while (1) {}

  return 0;
}
```

Alarm handler that overwrites the default behavior (exit)

Register SIGALRM handler

Set a 15 second alarm