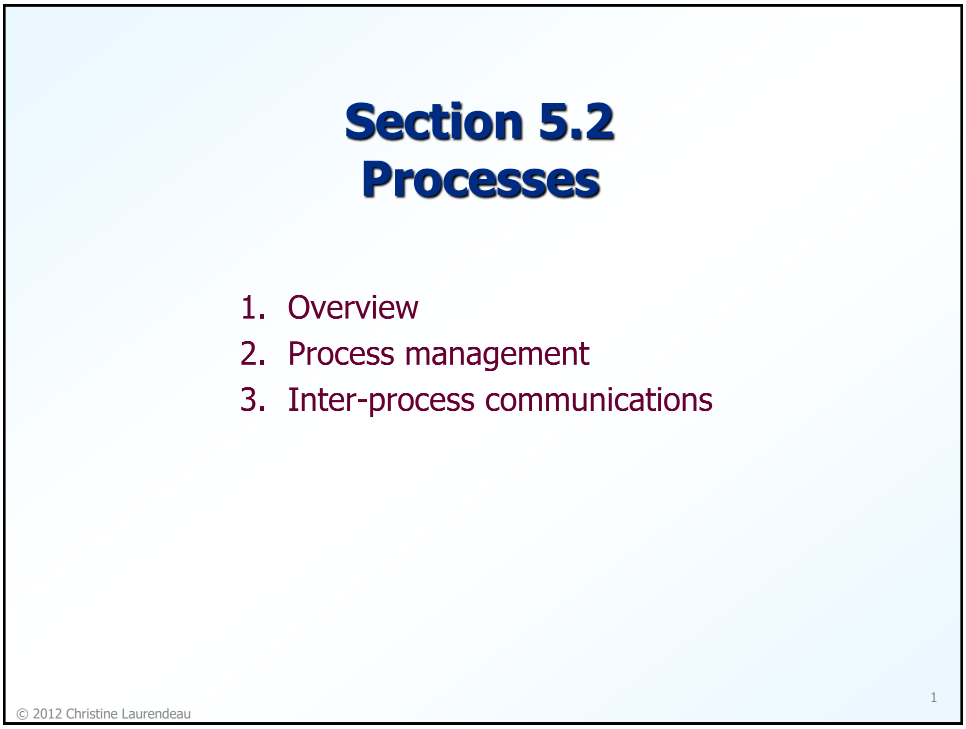
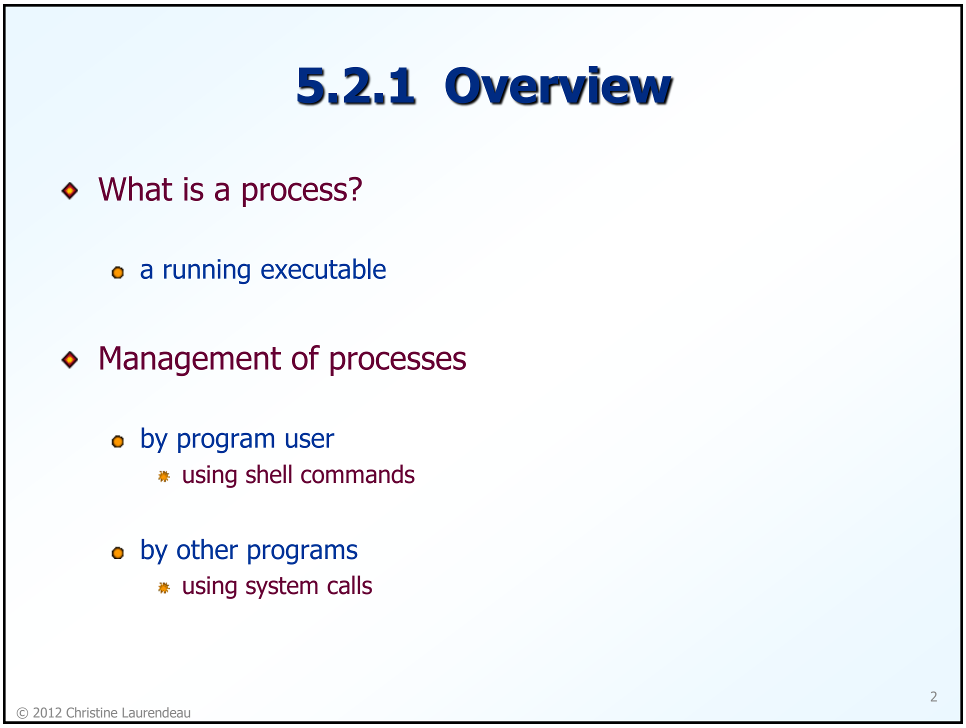


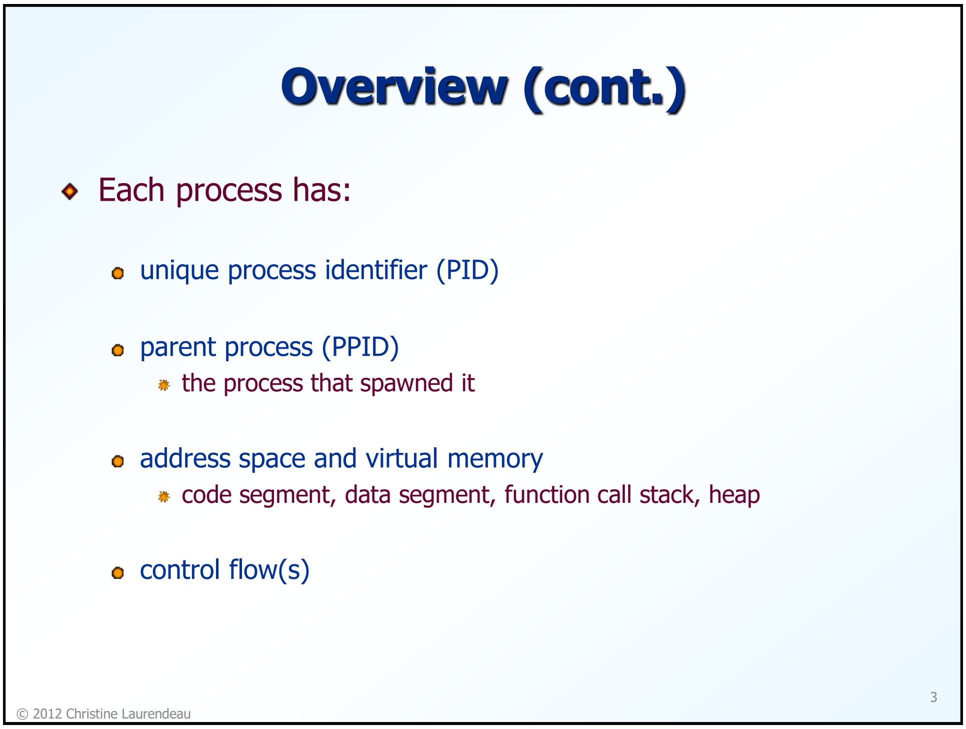
Section 5.2
Processes

1. Overview
2. Process management
3. Inter-process communications



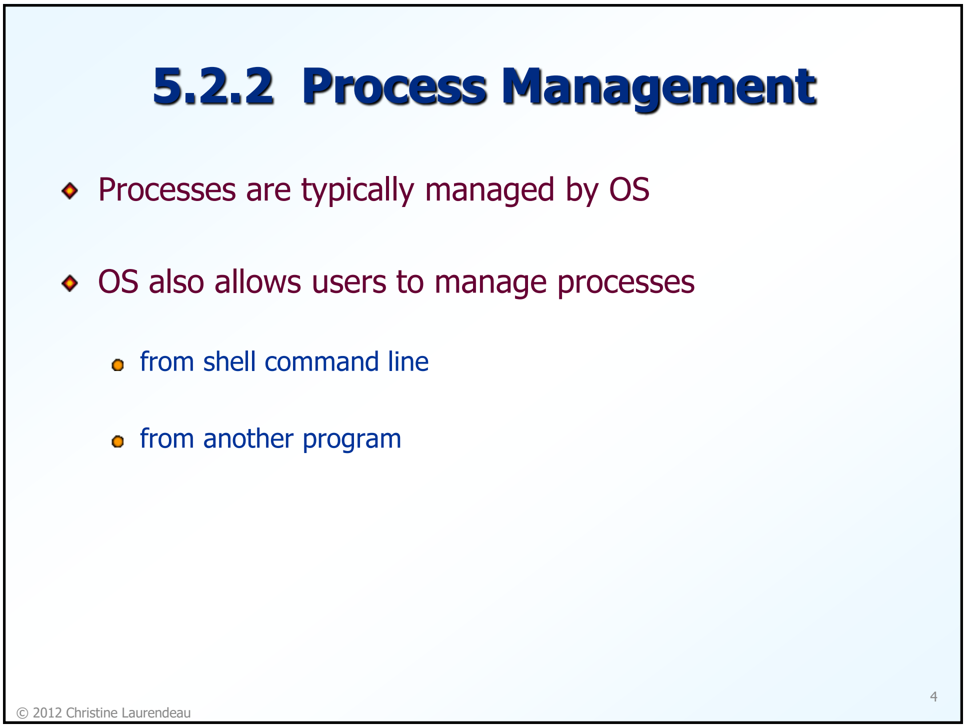
5.2.1 Overview

- What is a process?
  - a running executable
- Management of processes
  - by program user
    - using shell commands
  - by other programs
    - using system calls



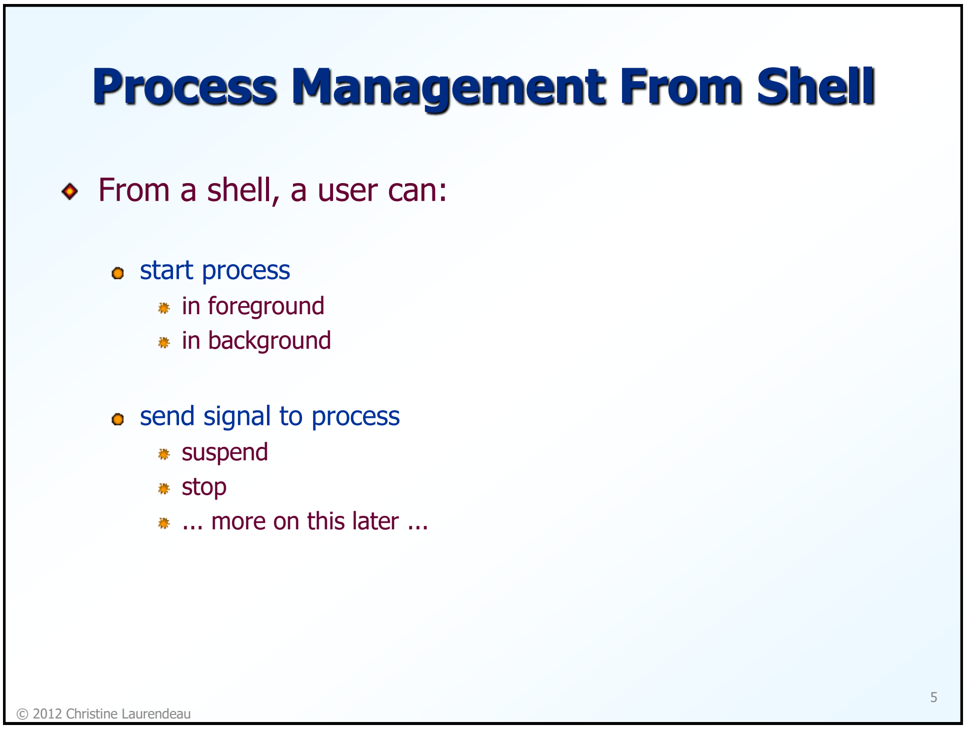
Overview (cont.)

- Each process has:
  - unique process identifier (PID)
  - parent process (PPID)
    - the process that spawned it
  - address space and virtual memory
    - code segment, data segment, function call stack, heap
  - control flow(s)



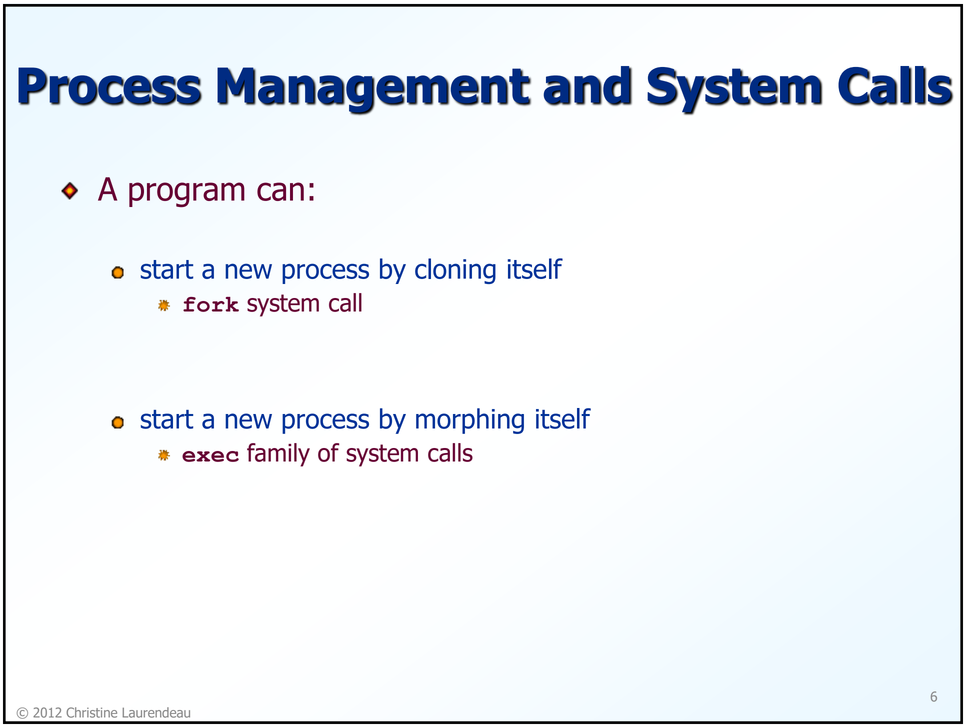
5.2.2 Process Management

- Processes are typically managed by OS
- OS also allows users to manage processes
  - from shell command line
  - from another program



Process Management From Shell

- From a shell, a user can:
  - start process
    - in foreground
    - in background
  - send signal to process
    - suspend
    - stop
    - ... more on this later ...



Process Management and System Calls

- A program can:
  - start a new process by cloning itself
    - **`fork`** system call
  - start a new process by morphing itself
    - **`exec`** family of system calls

Forking a Clone Process

`pid_t fork(void)`

- Description:
  - creates a clone of the current process
    - current process is the *parent*
    - new process is the *child*
    - child process gets copy of parent's address space
  - return value
    - in child process
      - zero
    - in parent process
      - child process id if successful
      - -1 in case of error

Forking a Clone Process (cont.)

- Multiple child processes can be spawned
  - child processes get a copy of parent code
  - multiple `forks` in the parent mean multiple `forks` in the children
- Watch for fork bombs
  - OS keeps process table
  - all tables have finite capacity

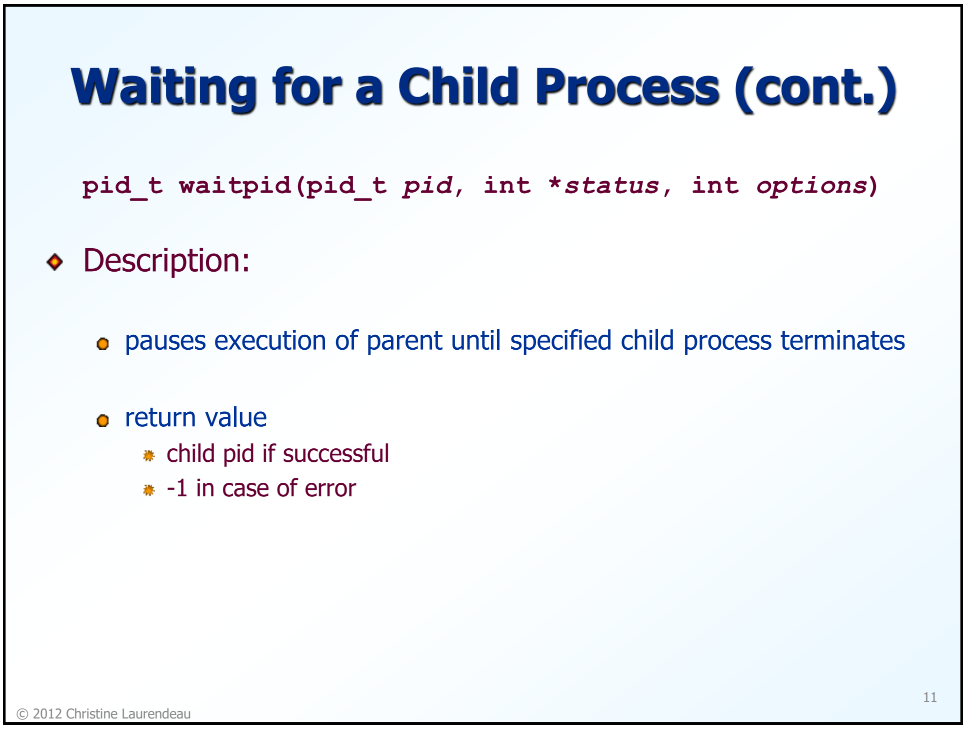
Morphing Into Another Process

- `exec` family of system calls
  - replace executing code of current process with another program
    - same PID
    - different instructions
  - include `execl(), execlp(), execle(), execv(), execvp()`
  - differences in parameters and environment settings
  - if `exec` call fails, original program continues

Waiting for a Child Process

`pid_t wait(int *status)`

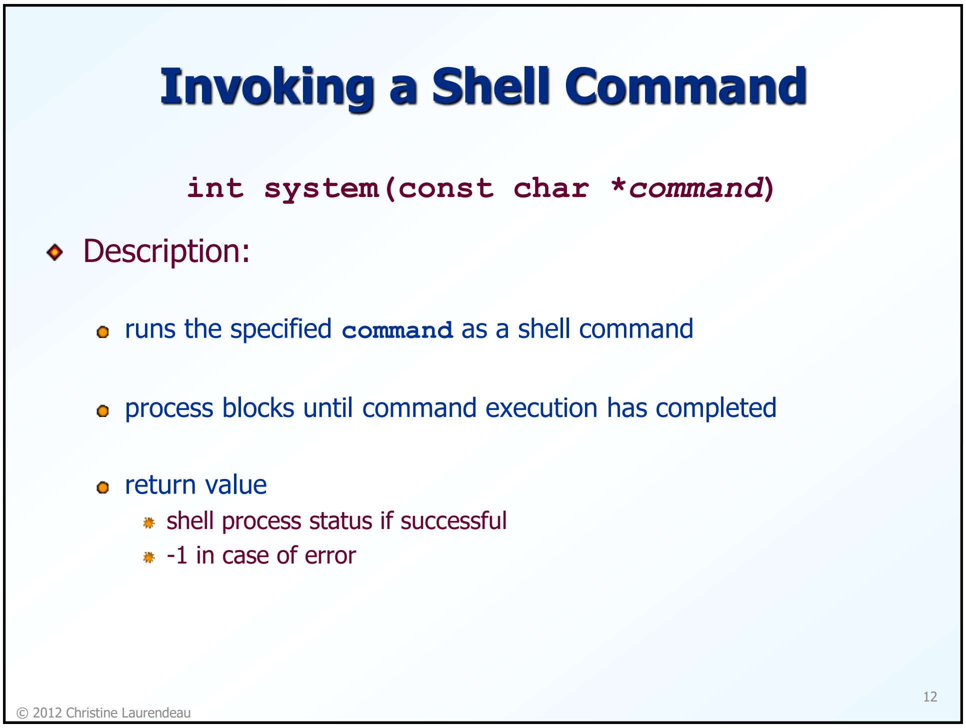
- Description:
  - pauses execution of parent until any child process terminates
  - return value
    - child pid if successful
    - -1 in case of error



Waiting for a Child Process (cont.)

```
pid_t waitpid(pid_t pid, int *status, int options)
```

- Description:
  - pauses execution of parent until specified child process terminates
  - return value
    - child pid if successful
    - -1 in case of error



Invoking a Shell Command

```
int system(const char *command)
```

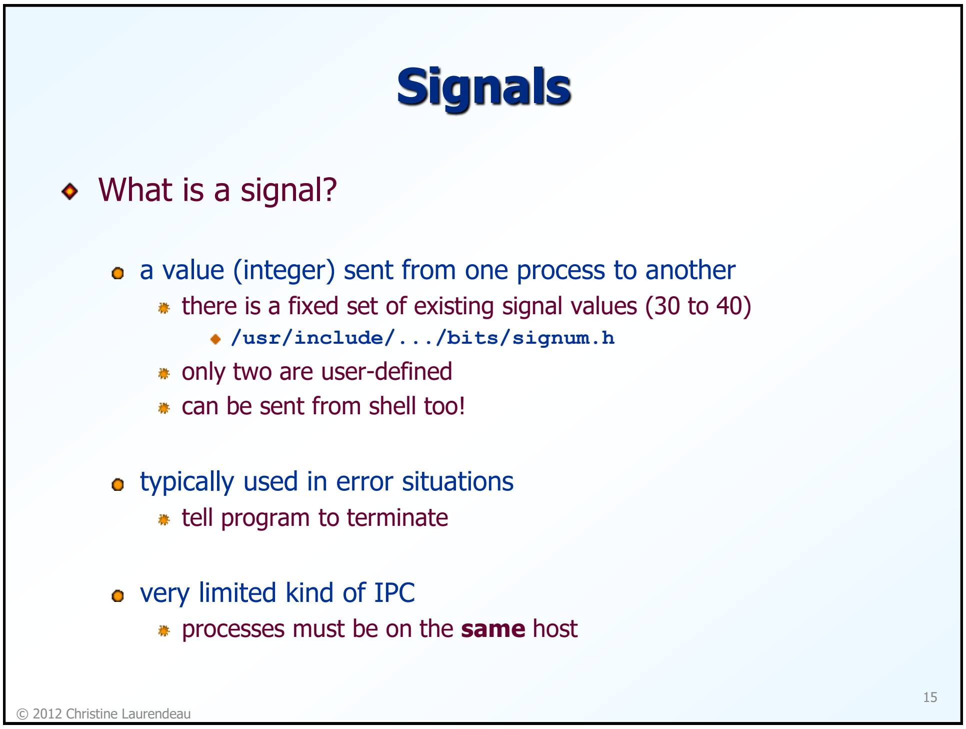
- Description:
  - runs the specified **`command`** as a shell command
  - process blocks until command execution has completed
  - return value
    - shell process status if successful
    - -1 in case of error

5.2.3 Inter-Process Communications

- IPC overview
- Signals
- Sockets

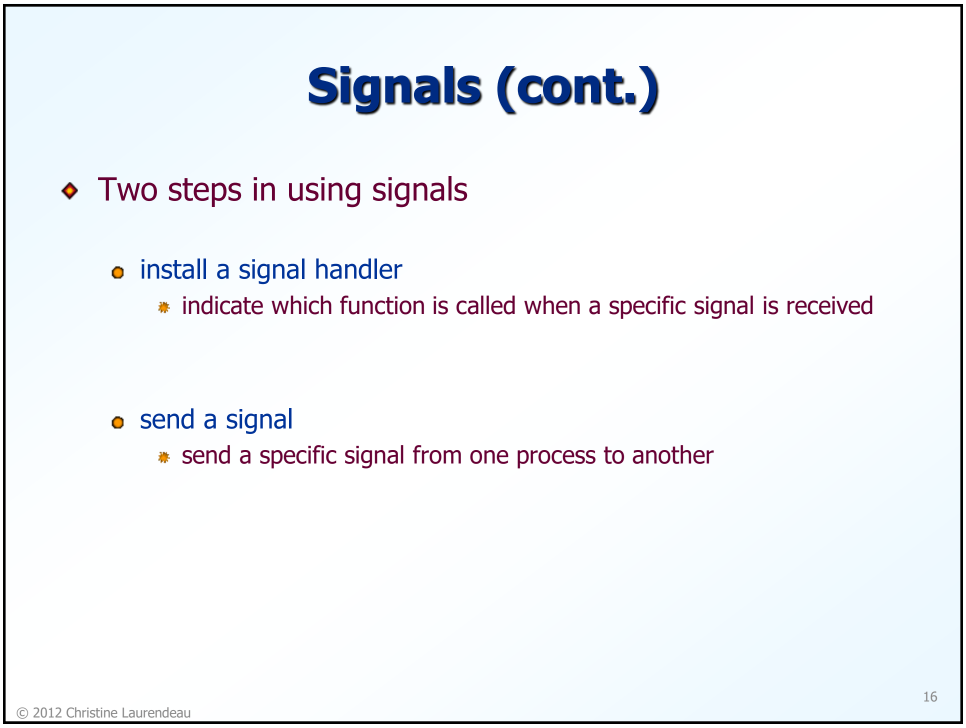
IPC Overview

- What is inter-process communications (IPC)?
  - sending and receiving information between processes
    - on the same physical host
    - on separate physical hosts
      - must be networked
- Main approaches to IPC
  - signals
  - sockets



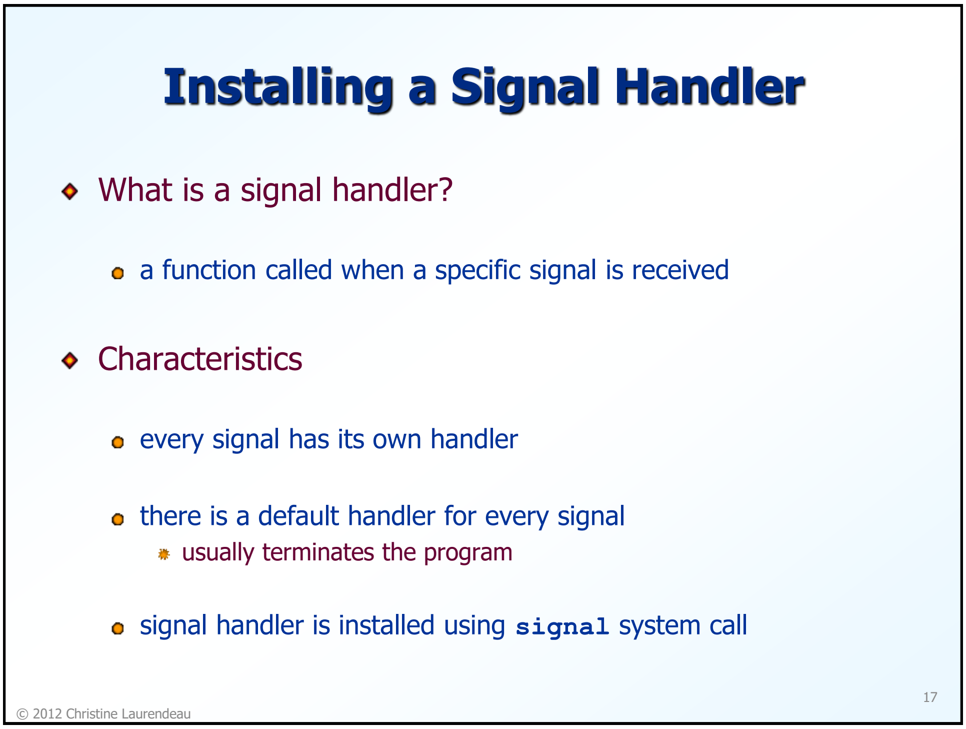
Signals

- What is a signal?
  - a value (integer) sent from one process to another
    - there is a fixed set of existing signal values (30 to 40)
      - `/usr/include/.../bits/signum.h`
    - only two are user-defined
    - can be sent from shell too!
  - typically used in error situations
    - tell program to terminate
  - very limited kind of IPC
    - processes must be on the **same** host



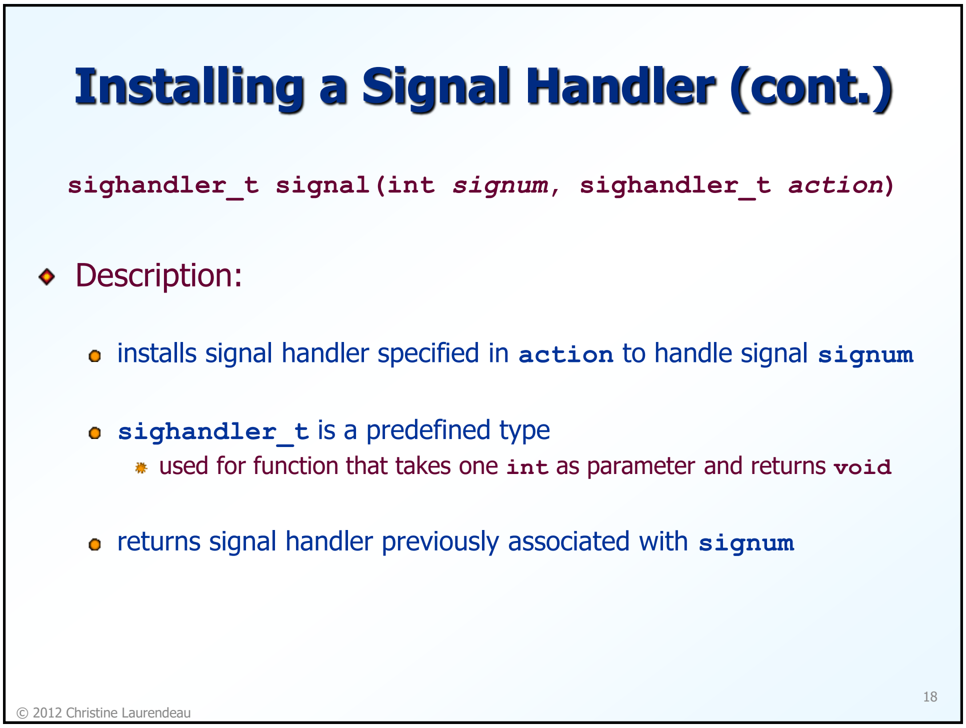
Signals (cont.)

- Two steps in using signals
  - install a signal handler
    - indicate which function is called when a specific signal is received
  - send a signal
    - send a specific signal from one process to another



Installing a Signal Handler

- What is a signal handler?
  - a function called when a specific signal is received
- Characteristics
  - every signal has its own handler
  - there is a default handler for every signal
    - usually terminates the program
  - signal handler is installed using **`signal`** system call



Installing a Signal Handler (cont.)

```
sighandler_t signal(int signum, sighandler_t action)
```

- Description:
  - installs signal handler specified in **`action`** to handle signal **`signum`**
  - **`sighandler_t`** is a predefined type
    - used for function that takes one **`int`** as parameter and returns **`void`**
  - returns signal handler previously associated with **`signum`**

Installing a Signal Handler (cont.)

- Description (cont.):
  - **`signum`** must be one of the predefined signal values
  - **`action`** can have one of the following values:
    - **`SIG_IGN`**
      - tells OS to ignore the signal and do nothing
    - **`SIG_DFL`**
      - tells OS to call the default signal handler
    - a signal handler function
      - tells OS to call the specified function

Sending a Signal

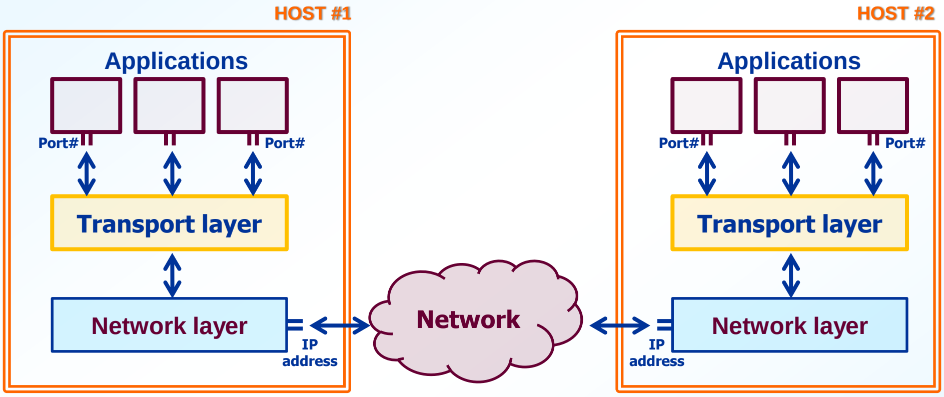
```
int kill(pid_t pid, int signum)
```

- Description:
  - sends the signal **`signum`** to the process with identifier **`pid`**
  - **`signum`** must be one of the predefined signal values
  - return value
    - 0 if successful
    - -1 in case of error

Sockets

- What is a socket?
  - an endpoint in IPC
    - processes can be on same or different hosts
  - socket address made up of:
    - IP address
      - indicating a unique host
    - port number
      - indicating a unique application running on that host
  - represented as an integer

Basic Networking



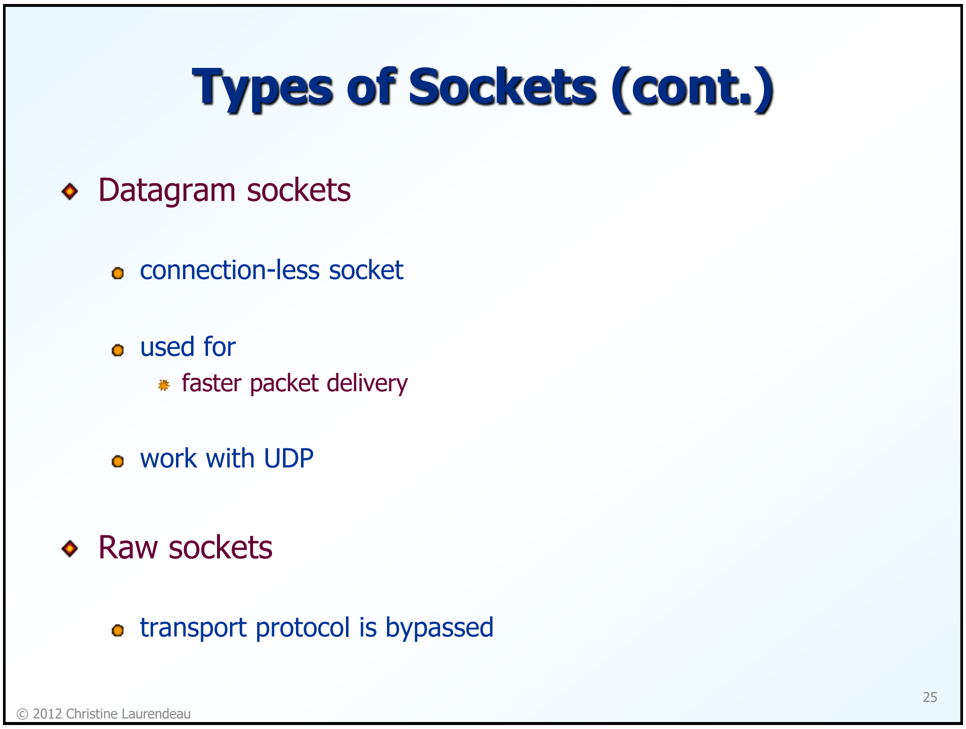
- Network layer protocol:
  - Internet Protocol (IP)
- Transport layer protocols:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)

Socket Components

- IP address
  - uniquely identifies a computer at the network layer
- Port number
  - uniquely identifies a process at the transport layer
  - only specific range of values is unreserved

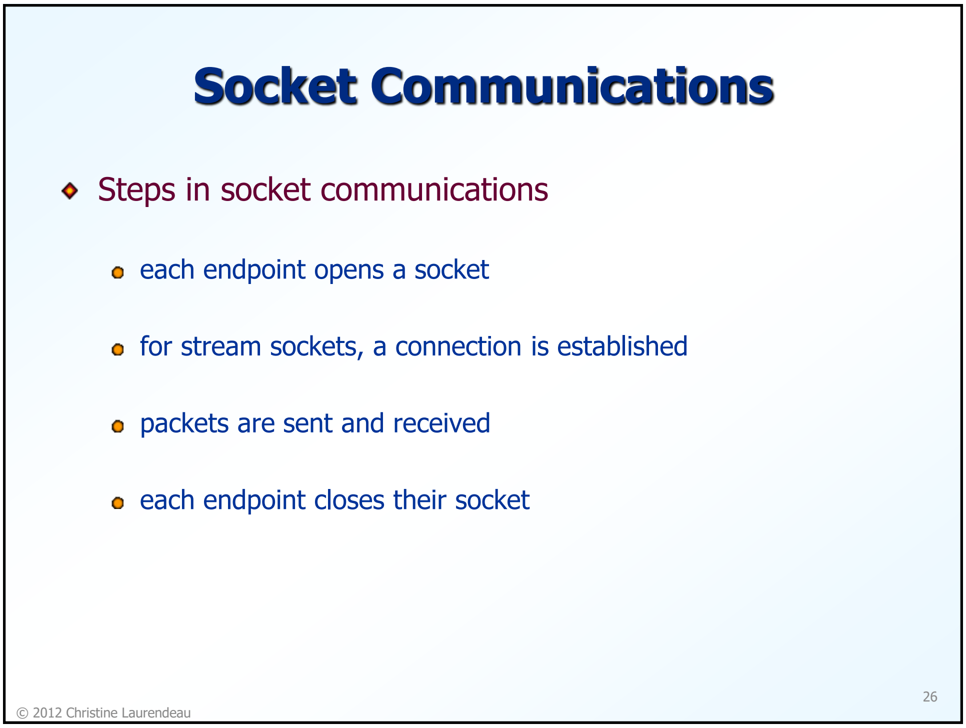
Types of Sockets

- Stream sockets
  - connection-based
    - connection must be established between sender and receiver first
    - connection is closed when communication is finished
  - used for
    - reliable packet delivery
    - packet correctness
    - reliable order of packets
  - work with TCP



Types of Sockets (cont.)

- Datagram sockets
  - connection-less socket
  - used for
    - faster packet delivery
  - work with UDP
- Raw sockets
  - transport protocol is bypassed



Socket Communications

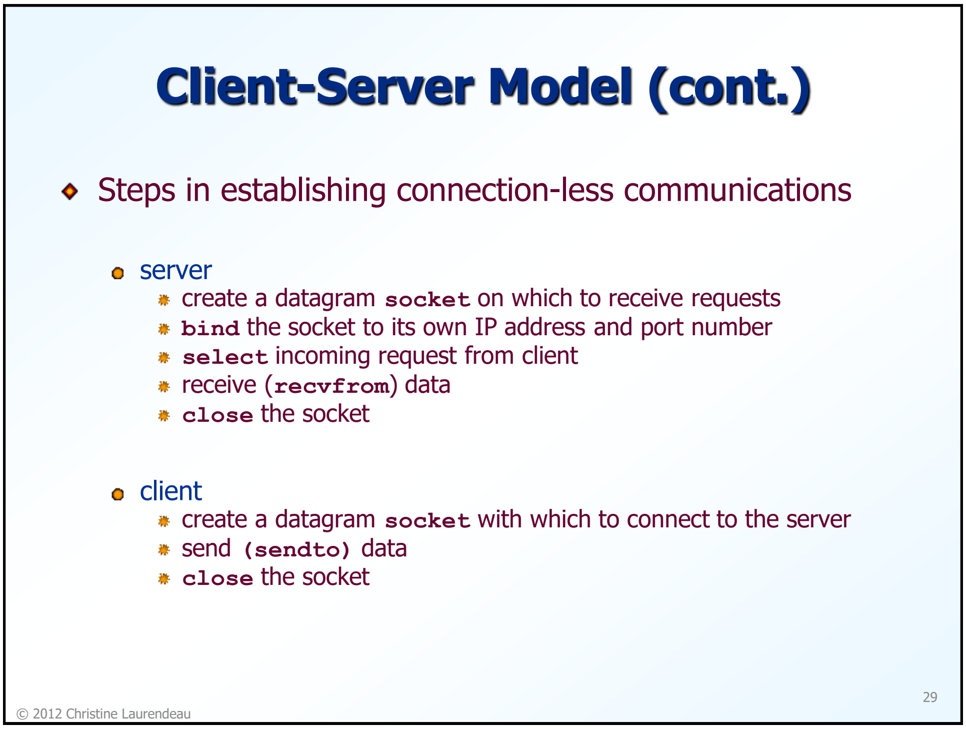
- Steps in socket communications
  - each endpoint opens a socket
  - for stream sockets, a connection is established
  - packets are sent and received
  - each endpoint closes their socket

Client-Server Model

- What is the client-server model?
  - a type of IPC architecture
- Characteristics
  - one server process receives requests and performs tasks
  - one or more client processes send requests to server

Client-Server Model (cont.)

- Steps in establishing connection-based communications
  - server
    - create a stream **`socket`** on which to receive requests
    - **`bind`** the socket to its own IP address and port number
    - **`listen`** on the socket for incoming connection request from client
    - **`accept`** a connection request from client
    - receive (**`recv`**) data
    - **`close`** the socket
  - client
    - create a stream **`socket`** with which to connect to the server
    - **`connect`** to the server at its IP address and port number
    - **`send`** data
    - **`close`** the socket



Client-Server Model (cont.)

- Steps in establishing connection-less communications
  - server
    - create a datagram **`socket`** on which to receive requests
    - **`bind`** the socket to its own IP address and port number
    - **`select`** incoming request from client
    - receive (**`recvfrom`**) data
    - **`close`** the socket
  - client
    - create a datagram **`socket`** with which to connect to the server
    - send **`(sendto)`** data
    - **`close`** the socket