Threads vs. Processes
Up until now, we’ve discussed *threads* -- the fundamental unit of computation -- and we know they’re organized into *processes*.

- Threads within a process share nearly all resources (exceptions are few, like the PC and their stack frames).
- Processes are almost _____________ from other processes.

<table>
<thead>
<tr>
<th>Threads</th>
<th>Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation</td>
<td></td>
</tr>
<tr>
<td>Overhead</td>
<td></td>
</tr>
<tr>
<td>Context Switching</td>
<td></td>
</tr>
<tr>
<td>Virtual Memory</td>
<td></td>
</tr>
</tbody>
</table>

Case Study: Chrome

Inter-Process Communication (IPC)
IPC is the broad terminology for all technologies that facilitate real-time communication between processes.

Approach #1:
Using a pipe within a terminal:
$ ps -aux | grep waf

Creating pipes in C:
int pipe(int pipefd[2]);

Approach #2: ________________

Approach #3: ________________
Sending a signal within a terminal:
$ kill -TERM <pid>

Listing all available signals:
$ kill -l

Sending a signal in C:
int kill(pid_t pid, int sig);

Approach #4: ________________
Alloacting shared memory in C (“malloc for shared memory”):
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);

Approach #5: ________________
Functions in C:

```c
mqd_t mq_open(const char *name, int oflag);
int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned int msg_prio);
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned int *msg_prio);
int mq_close(mqd_t mqdes);
```

Approach #6: ________________
Approach 7: _________________

Creating a new socket interface, returns a **file descriptor**:

```c
int socket(int domain, int type, int protocol);
```

Binding a socket interface to an address and port:

```c
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Connecting to a remote socket:

```c
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

Begin listening for a remote socket connection:

```c
int listen(int sockfd, int backlog);
```

Start a new socket channel with a remote host:

```c
int accept(int sockfd, struct sockaddr *restrict addr, socklen_t *restrict addrlen);
```

---

**High Level Overview of Sockets**

At the core of socket-based IPC, you have a _________________ coming from a “remote host”.

- ____________:
- ____________:
- Port Number:

<table>
<thead>
<tr>
<th>Server</th>
<th>Client</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs340-adm.cs.illinois.edu:34000</td>
<td></td>
</tr>
</tbody>
</table>

---

**Simple Socket Communication: telnet**

The Linux utility **telnet** provides simple socket communications by sending all data you enter directly over the socket:

```
$ telnet cs340-adm.cs.illinois.edu 34000
```

(To exit, press `Ctrl+]` to go into command mode; then type **quit**.)