COMP 3430

Operating Systems

May 13th, 2019

Goals

By the end of today's lecture, you should be able to:

- Compare and contrast *processes* and *programs*.
- Show how a process is launched by an OS.

Pixabay License

Launching a process

Let's launch some processes on `aviary`.

Relationship between processes?

With the person beside you, answer:

What do these processes *share*?

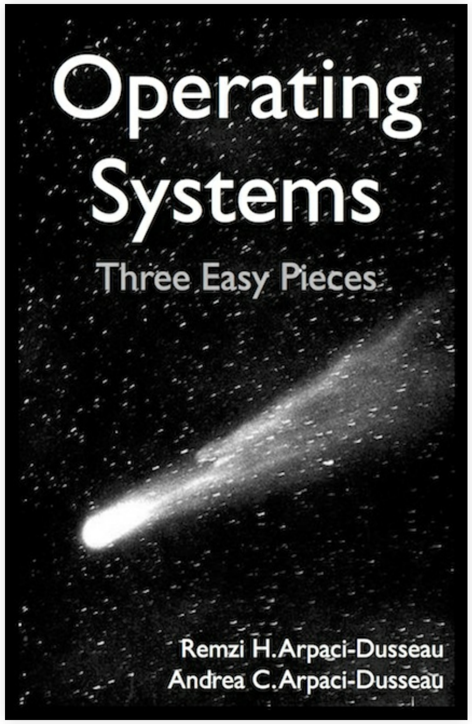
Pixabay License

Launching a process

- That’s how *we* launch a process…
- What does the OS do after we press Enter?

OS: Three Easy Pieces

- The book *describes* the process (in *excruciating* detail).
- … but leaves the implementation as an exercise.
  - Let’s take a look.



© RH & AC Arpaci-Dusseau

What *is* a **program**?

- Let's take a *big* step back: What even *is* a **program**?
  - That is, what's the result of

```
clang -Wall myprog.c -o
 myprog
```

(*do not* overthink this…)

Pixabay License

Files!

- A program is *literally* a file.
  - A binary, ELF-formatted file (on a Linux system…)
  - Let’s take a closer look at an `ELF` file.
- So… a program is a *file*
  - How do you read a file in C?

Pixabay License

COMP 2160

Programming Practices

May 13th, 2019

Today

- Reading files in C.

Reading files in C

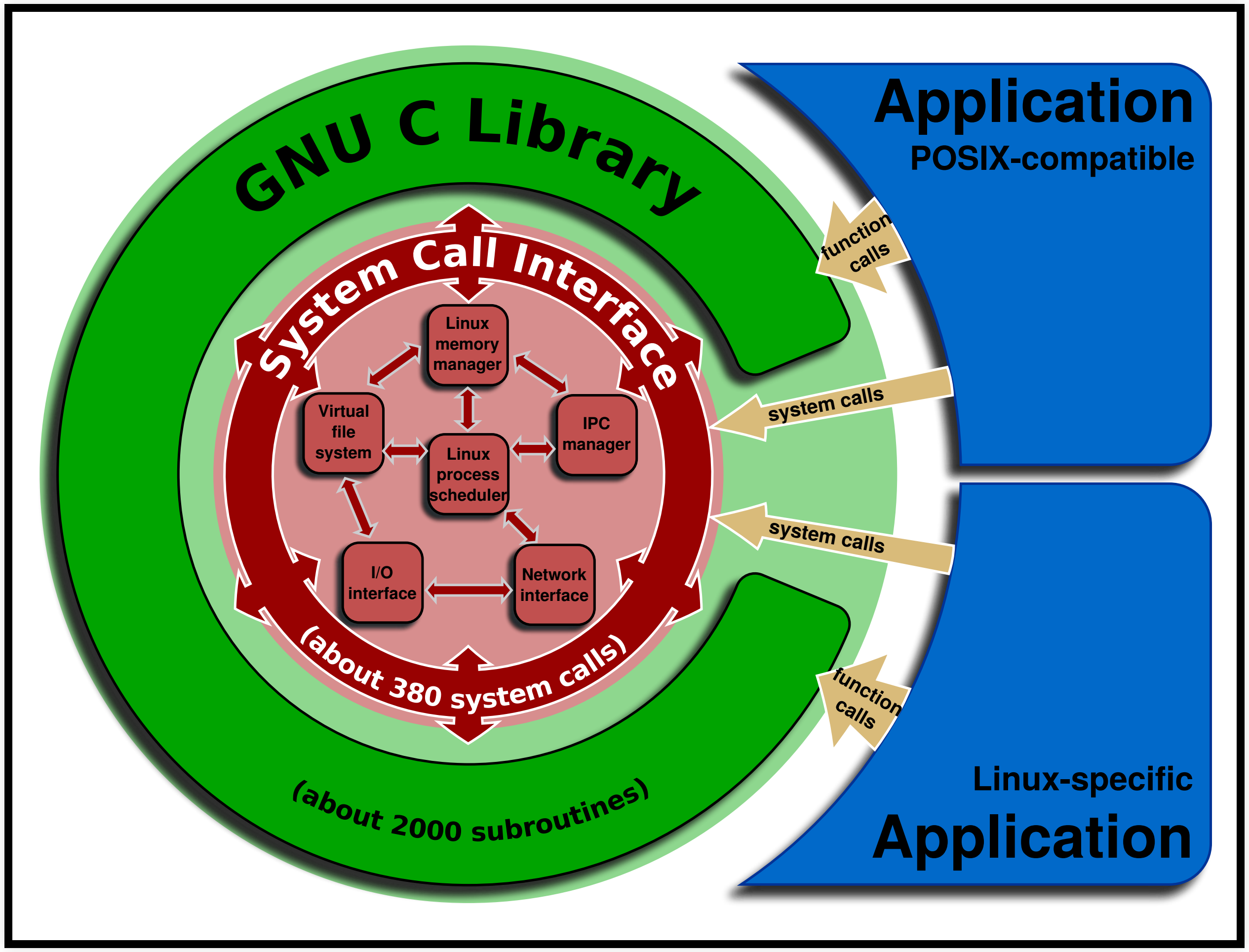
- Start with `#include <stdio.h>`, then:
  1. `fopen`
  2. *Check* the value that was returned.
  3. `fgets`, `fgetc`, … wait.

Pixabay License

(Back to 3430)

- What kind of data do these functions deal with?
  - What kind of data is in an `ELF` formatted file?
- Most importantly: Where do these functions *come* from?

What *is* an OS?


GNU C Library
(about 2000 subroutines)
System Call Interface
(about 380 system calls)
Linux memory manager
Virtual file system
IPC manager
Linux process scheduler
I/O interface
Network interface
Application
POSIX-compatible
function calls
system calls
system calls
function calls
Linux-specific
Application

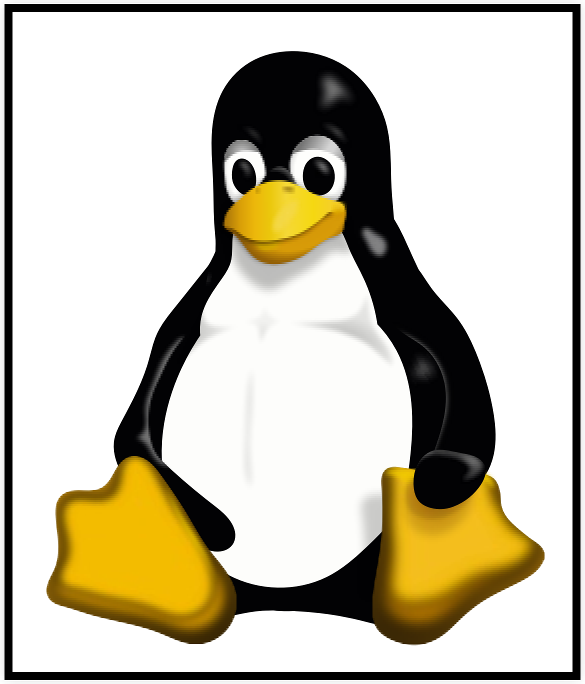
Linux

- Let's take a look at how a *real* OS opens a file.

1. Go to https://github.com/torvalds/linux
2. Do some cave diving – see if you can find where a process is launched.
3. Related to COMP 3350: Is there a discernable architecture? (3-tier 4 lyfe)

Extra reading on LWN:

- How programs get run
- How programs get run: ELF binaries



CC0 Creative Commons

Opening a file

(With an OS)

- Remember: A program *is* a file – the OS needs to *open* and **read** the file.
- We don't have a C library (`StackOverflowException`).
  - The OS uses its own internal mechanisms to interact *directly* with the file system (persistence!).

Pixabay License

Starting the process

- The OS can read a file.
- The OS needs to load the file into *memory*.
  - Into a **data structure** that has information about the process.

Pixabay License

With *a* person beside you (try finding someone new!), answer the question:

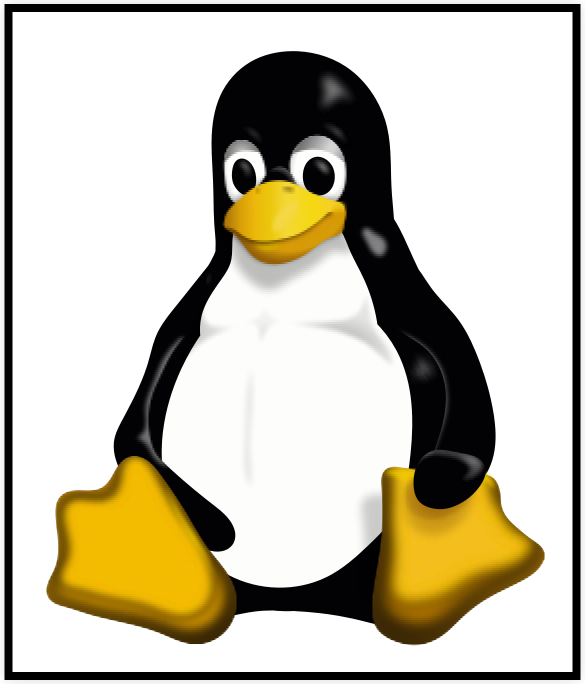
What kind of information might an OS need to keep *about* a process while the process is running ?

Pixabay License

What does a *real* OS know?

Let's take a look at `struct task_struct`.

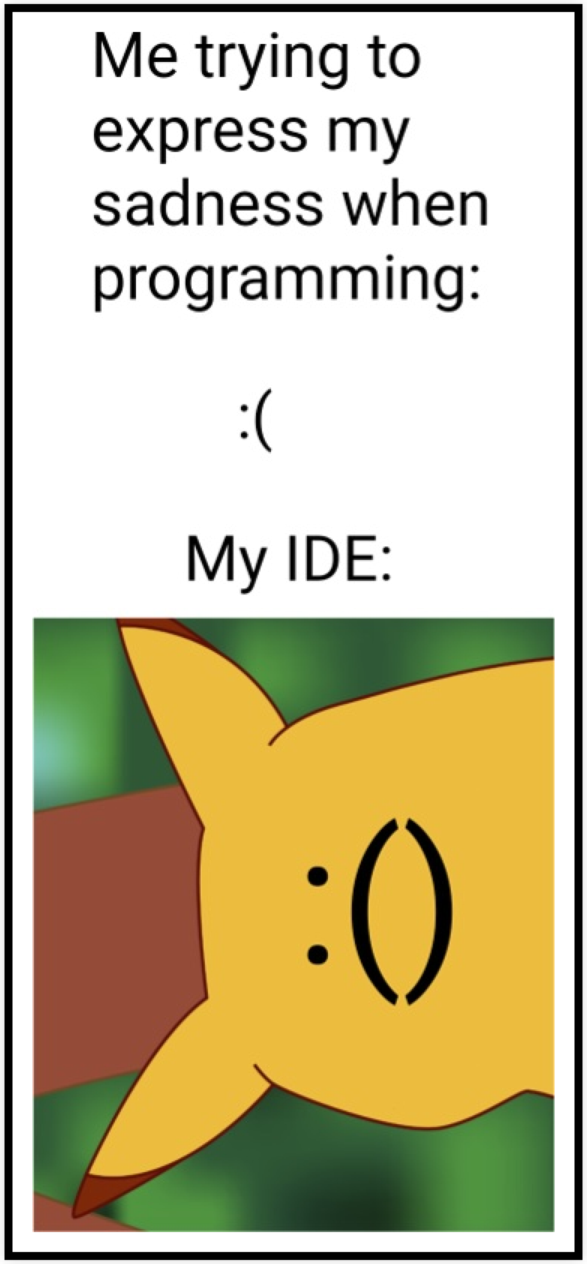
1. Did you think about anything that *isn't* in `struct task_struct`?
2. What kinds of things are in `struct task_struct` that you didn't think of?
3. Left as an exercise for the reader: How is `struct task_struct` in Linux different from `struct task` in Darwin ?


CC0 Creative Commons

Launching a process

- An OS first needs to **read** a program.
  - The program is read into a **data structure** (in memory)
  - The data structure maintains **metadata** *about* the running program.
  - The data structure includes an address where the program *starts* in memory.
- …what's next? Ideas?
  - `JMP`/`JSRR`/`JSR main`
- A question for next time: What state is a process in *when it starts*?

© Thilo Kranz/DLR (CC BY 2.0)


Me trying to express my sadness when programming:
:(
My IDE:
:()