ELEC 424/553
Mobile & Embedded Systems

Lecture 4 - Process Management, Getting Comfortable with Linux, & Exploring Linux Source Code

Project 1: Cracking Open A Cold Raspberry Pi

- To be posted soon
- Setting up your RPi Zero W

“Linux: Processes”. User: The Linux Man. https://www.youtube.com/watch?v=NB5QcYhGY3c
“What is a process in UNIX / Linux?”. https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

CLEAR Storage

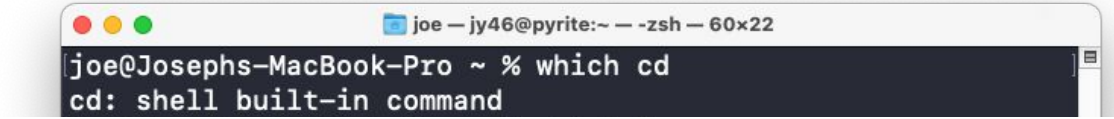
Everyone’s CLEAR storage should have been increased by 2GB

IT WILL GO BACK DOWN BY 2GB AROUND DECEMBER 21st!

YOU COULD LOSE THINGS IF YOU DON’T DELETE THINGS BY THEN

Process: Program Instance

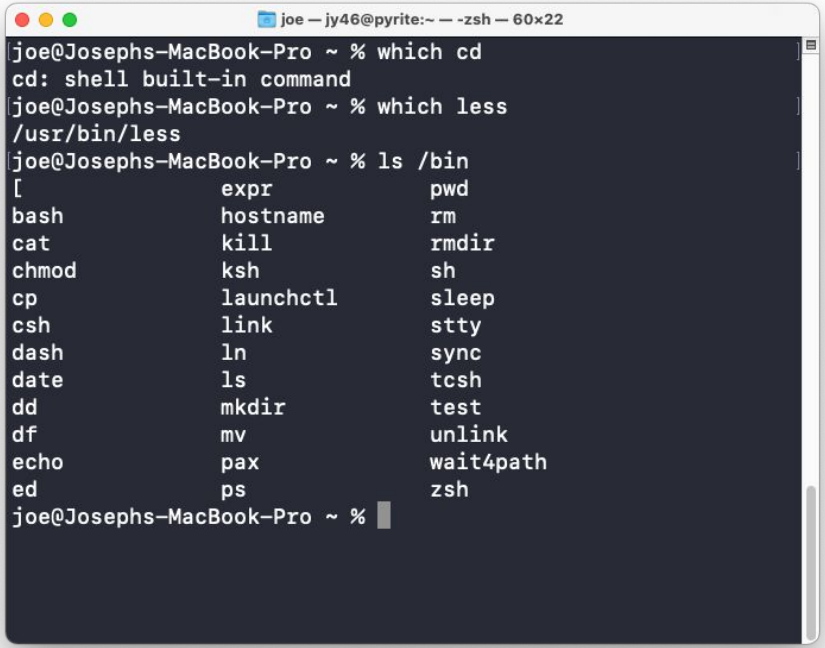
- Examples of programs:
  - Internal shell command
    - cd



“Linux: Processes”. User: The Linux Man. https://www.youtube.com/watch?v=NB5QcYhGY3c
“What is a process in UNIX / Linux?”. https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

Process: Program Instance

- Examples of programs:
  - Internal shell command
    - cd
  - Binary executable
    - 01001111101…



"Linux: Processes". User: The Linux Man. https://www.youtube.com/watch?v=NB5QcYhGY3c
"What is a process in UNIX / Linux?". https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

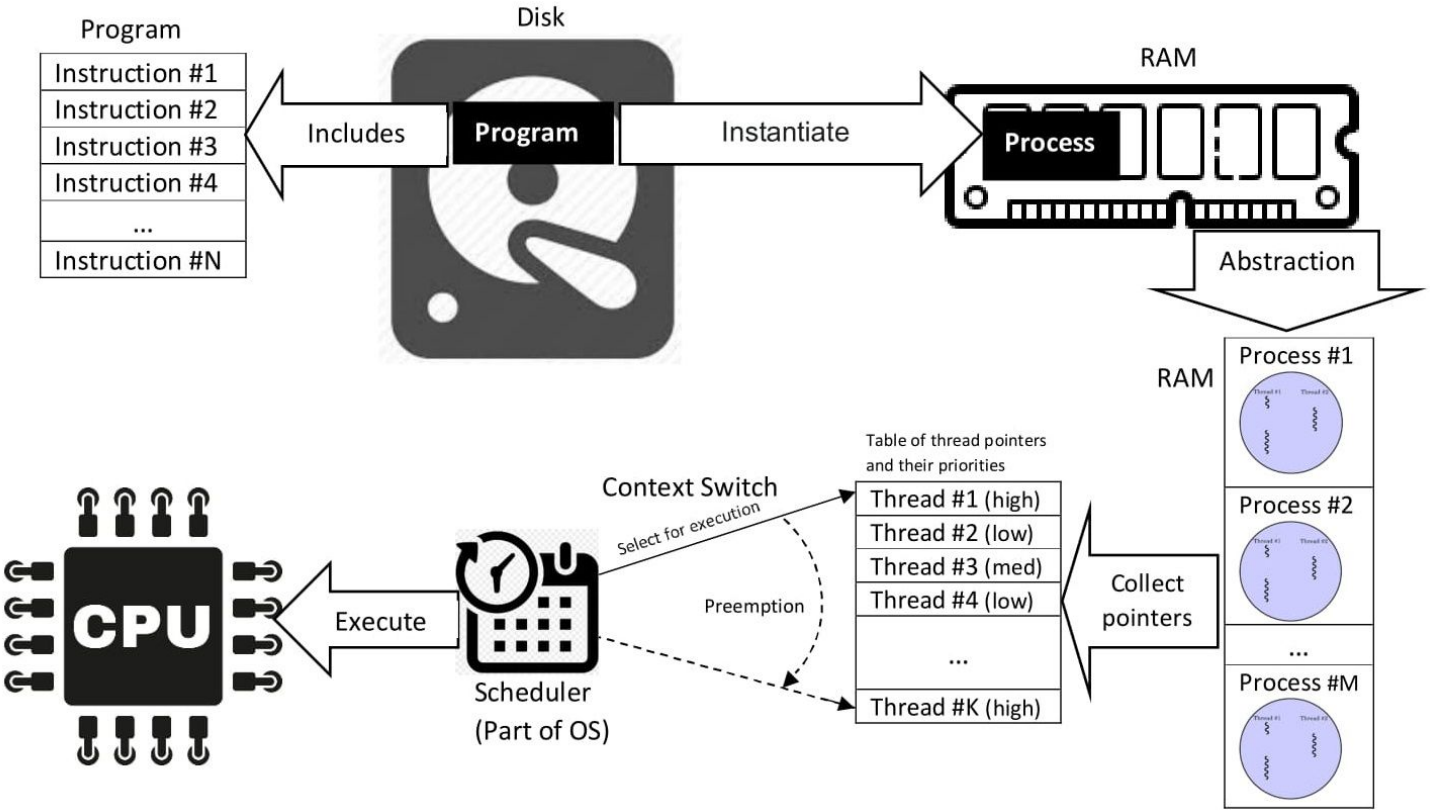
Process: Program Instance

- Examples of programs:
  - Internal shell command
    - cd
  - Binary executable
    - 01001111101…
  - Shell script
    - File of commands

https://linuxhint.com/30_bash_script_examples/

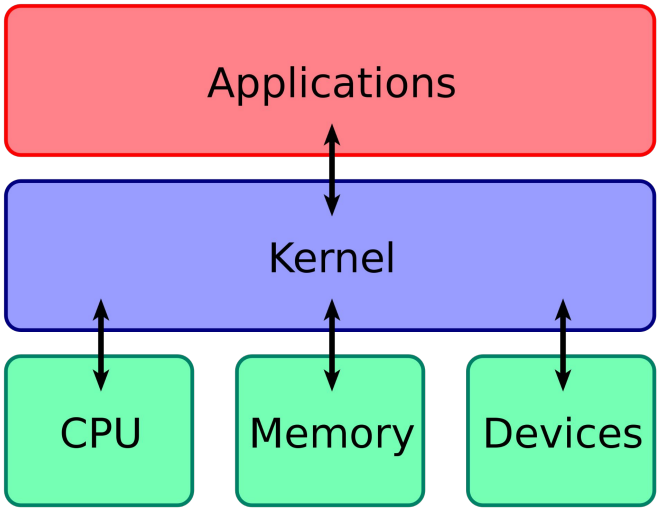
“Linux: Processes”. User: The Linux Man. https://www.youtube.com/watch?v=NB5QcYhGY3c
“What is a process in UNIX / Linux?”. https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

Programs, Processes, Threads, …



Processes in Linux

- The goal of an OS is to host applications
- Process management is critical
- From Love (the author):
  “A **process** is a program (object code stored on some media) in the midst of execution”
- Processes don’t (generally) share memory



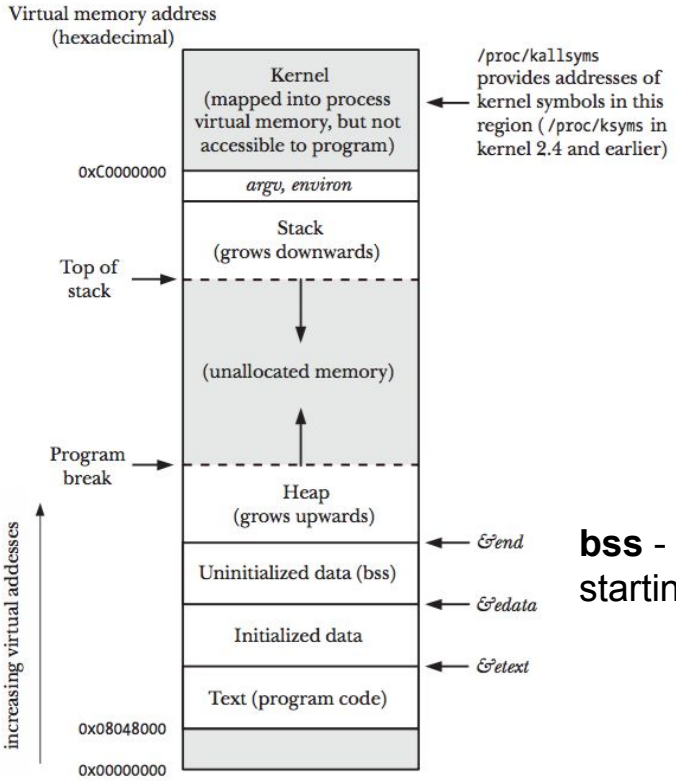
Robert Love, *Linux Kernel Development*, 3rd Edition

Process Virtualization

- Virtual processor
  - Makes it look like process has processor all to itself
  - Threads view private virtualized processors
- Virtual memory
  - Makes it look like process has total memory to itself
  - Threads view the same virtual memory

Robert Love, *Linux Kernel Development*, 3rd Edition

What Does a Process Consist Of?



bss - block starting symbol

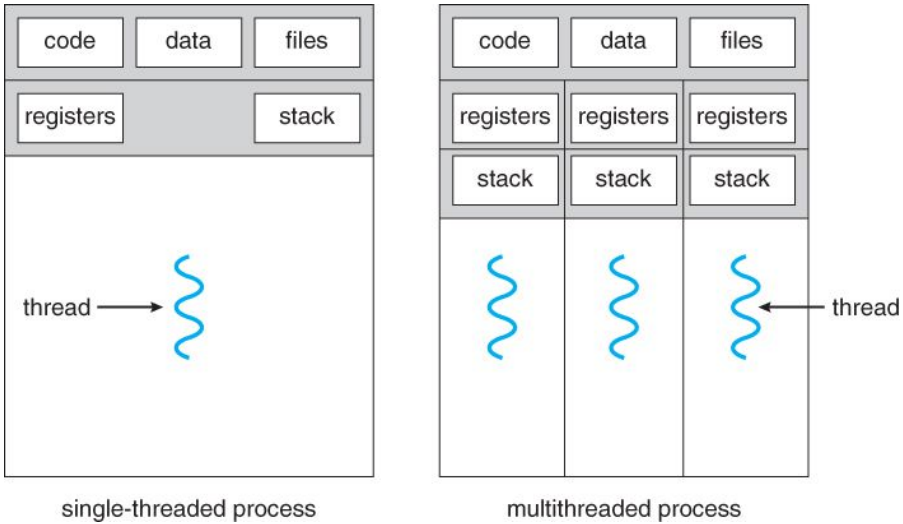
- Not only the program code (“text”)
- Also contains:
  - Open files
  - Pending signals
  - Internal kernel data
  - Memory address space
  - 1+ threads
  - Data section

Shichao's Notes. https://notes.shichao.io/tlpi/ch6/

Robert Love, *Linux Kernel Development*, 3rd Edition

Threads

- **Threads** are *technically* the minimum set of instructions that a scheduler can handle
- Schedulers generally manage threads, but in Linux the situation is unique
- A thread gets its own:
  - Program counter
  - Stack
  - Registers
- **Multithreaded** programs have 2+ threads
- In contrast to processes, threads do share memory



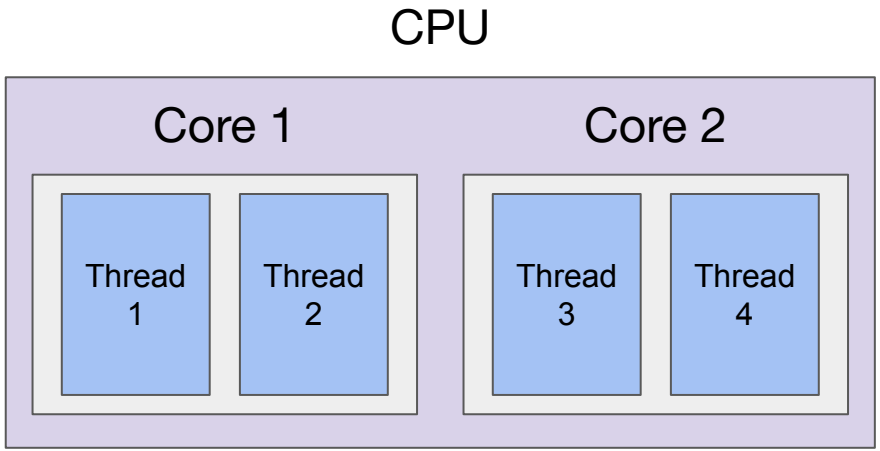
URL: https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/4_Threads.html

Figure source: Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts, Ninth Edition ", Chapter 4

Robert Love, *Linux Kernel Development*, 3rd Edition

Multithreading in Software & Hardware

- It’s possible to execute multiple threads at the same time
- Multithreading
  - Requires SW & HW support
- Multicore

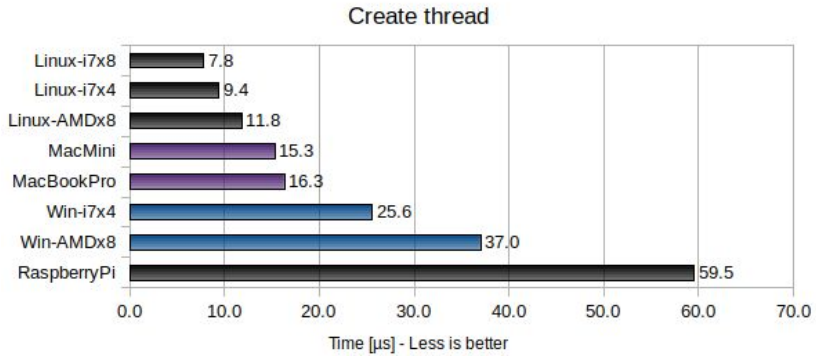


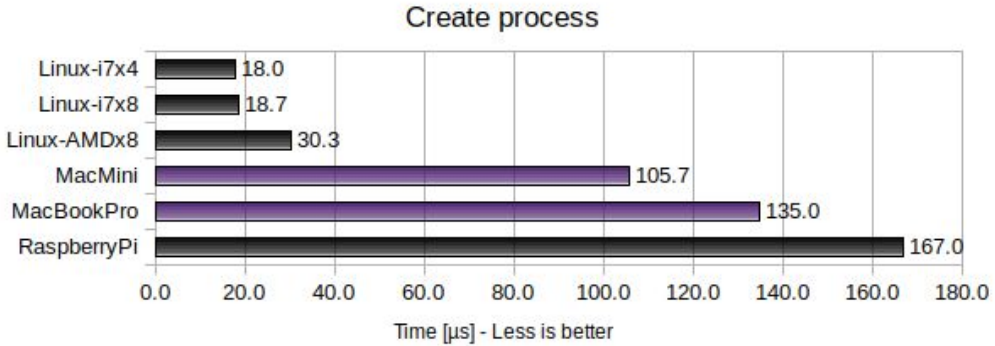
Threads vs. Processes in Linux

- Linux views threads as particular processes
  - Processes with items common to other processes
  - Get their own `task_struct`
- Naming confusion: **Task**
  - Love says task and process are synonyms
  - Wikipedia will tell you task and thread are synonyms (link)
- Windows different - threads particularly handled by kernel

Robert Love, *Linux Kernel Development*, 3rd Edition

Linux Thread/Process Creation Time





Marcus Geelnard, “Benchmarking OS primitives”. https://www.bitsnbites.eu/benchmarking-os-primitives/

Running a Program (Creating a Process)

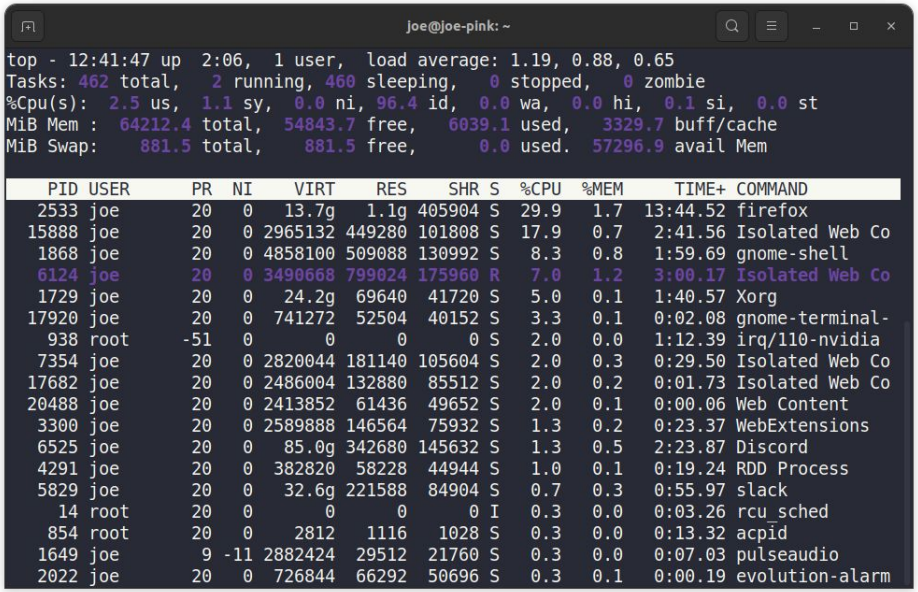
- nano hello.c

```
#include <stdio.h>
int main()
{
        printf("Hello, World!\n");

}
```

- gcc hello.c -o hello
- objdump -d hello
  - Assembly language
  - It’s just instructions! :)

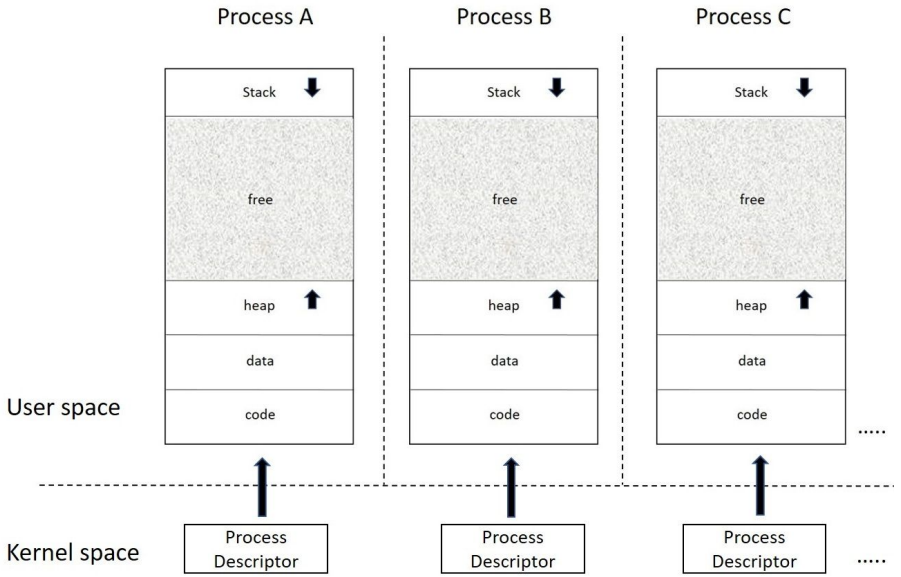
Viewing Existing Processes

- `top` shell command shows processes
- shift+f provides interactive menu
- Can press s on PID then q
- Then can do shift R to change to ascending
- Notice `systemd` 😈



Process Attributes

- We've seen that a process has an address space
- **Process descriptor** additionally associated with process
- Descriptor exists in kernel space so processes can be managed effectively



Raghu Bharadwaj, *Mastering Linux Kernel Development*. (2017)
https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

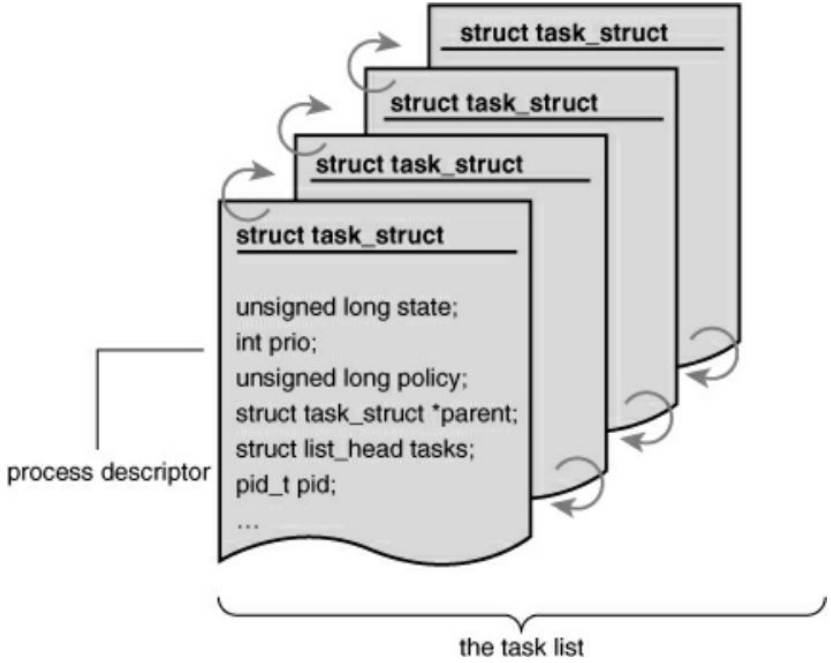
Linux Source Party

Link to festivities

https://www.pcgamer.com/penguin-party-steam-linux-launches-with-week-long-celebration-sale/

Process Descriptor

- Instance of `struct task_struct`
  - mm_struct
  - Thread_struct (x86)
- Motivates viewing the source code of Linux
- Source located at www.kernel.org
- Can find `struct task_struct` defined at line 661 in `include/linux/sched.h` (link)

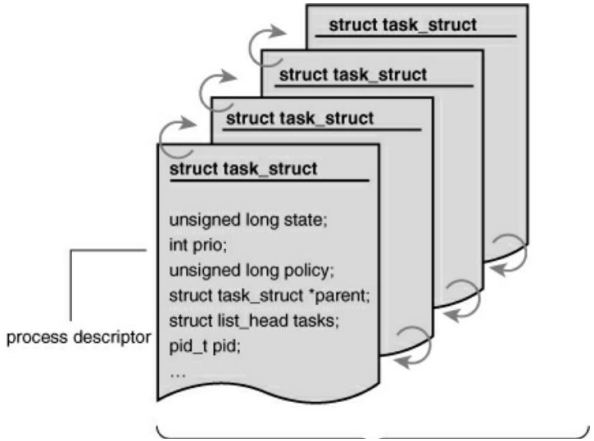


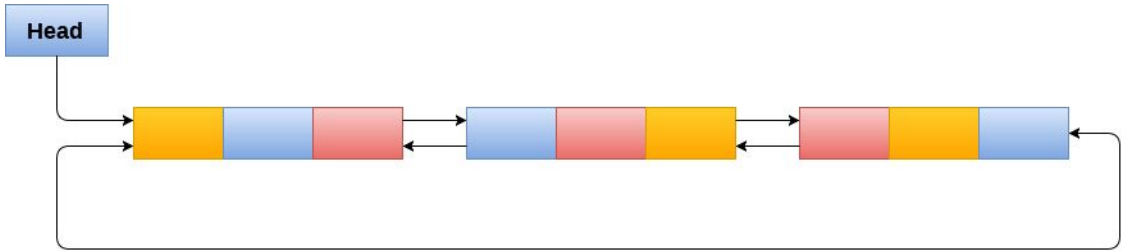
Raghu Bharadwaj, *Mastering Linux Kernel Development*. (2017)
https://www.theunixschool.com/2012/09/what-is-process-in-unix-linux.html

Robert Love, *Linux Kernel Development*, 3rd Edition

Task List

- Kernel tracks processes in this list
- Circular doubly linked list





Circular Doubly Linked List

https://www.javatpoint.com/circular-doubly-linked-list

Memory Representation of a Circular Doubly linked list

https://www.javatpoint.com/circular-doubly-linked-list

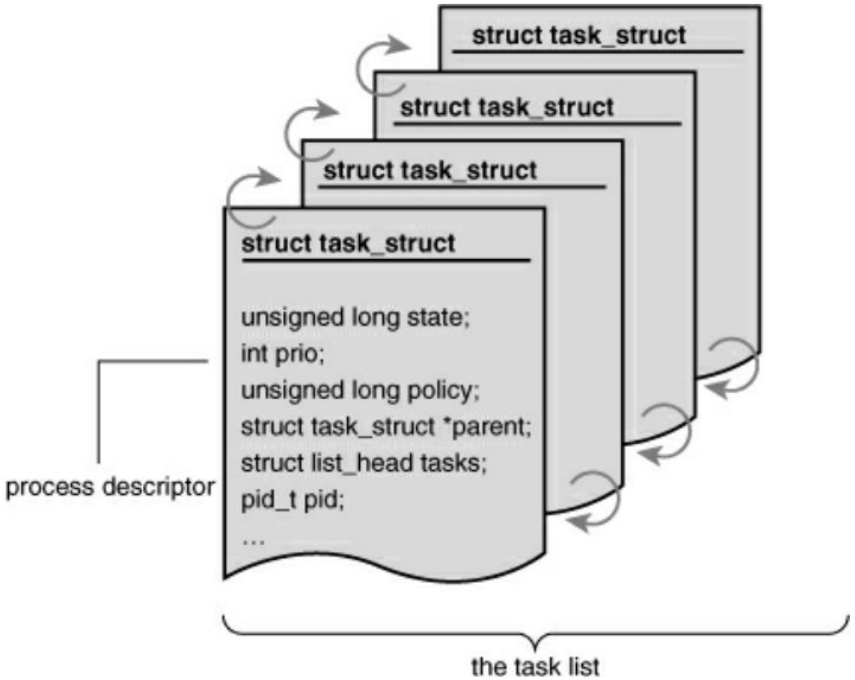
Implementation of Circular Doubly Linked List in C

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
```

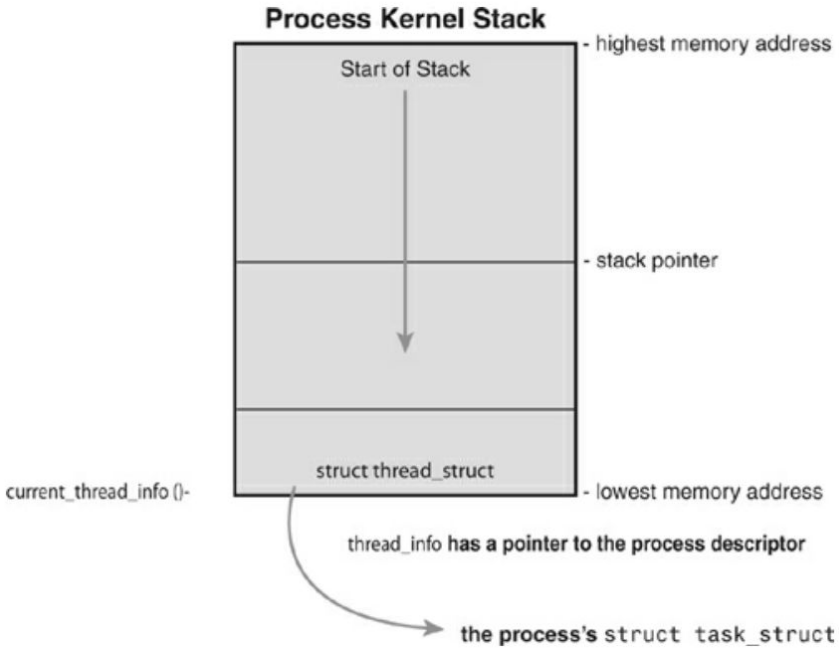
“Circular Doubly Linked List”. https://www.javatpoint.com/circular-doubly-linked-list

What's Inside `task_struct`? [Process Descriptor]

- Open files
- Address space
- Pending signals
- State
- …



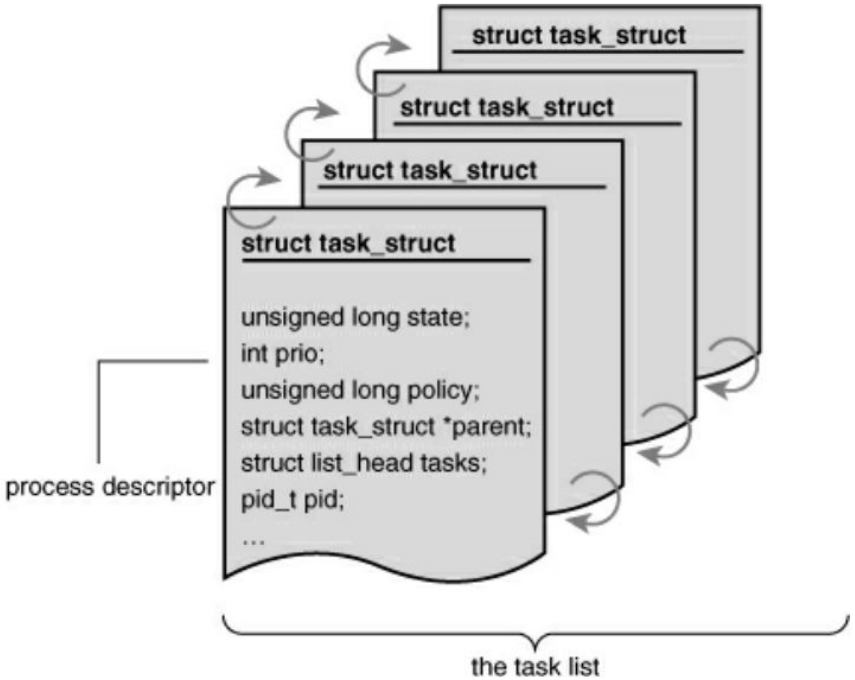
Robert Love, *Linux Kernel Development*, 3rd Edition



Robert Love, *Linux Kernel Development*, 3rd Edition

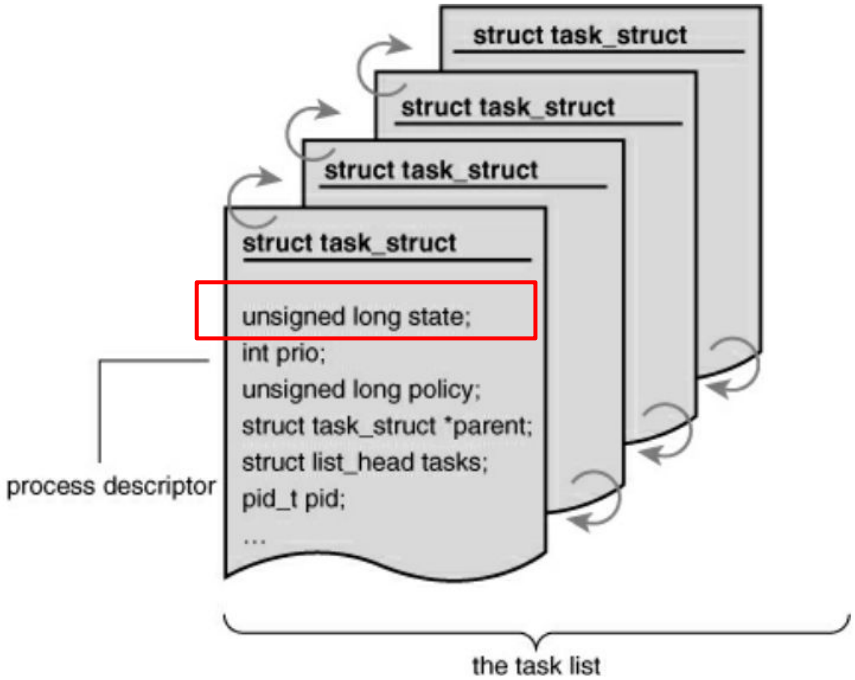
Working With Process Descriptors

- Process identification value (PID) gets assigned to processes
  - In **`task_struct`**:
    - `pid_t` **`pid;`**
  - `pid_t` is an opaque type, usually `int`
- Kernel generally addresses processes' `task_struct` (process descriptor)
- We have a macro (`current`) that gets us the presently running process' process descriptor



Robert Love, *Linux Kernel Development*, 3rd Edition

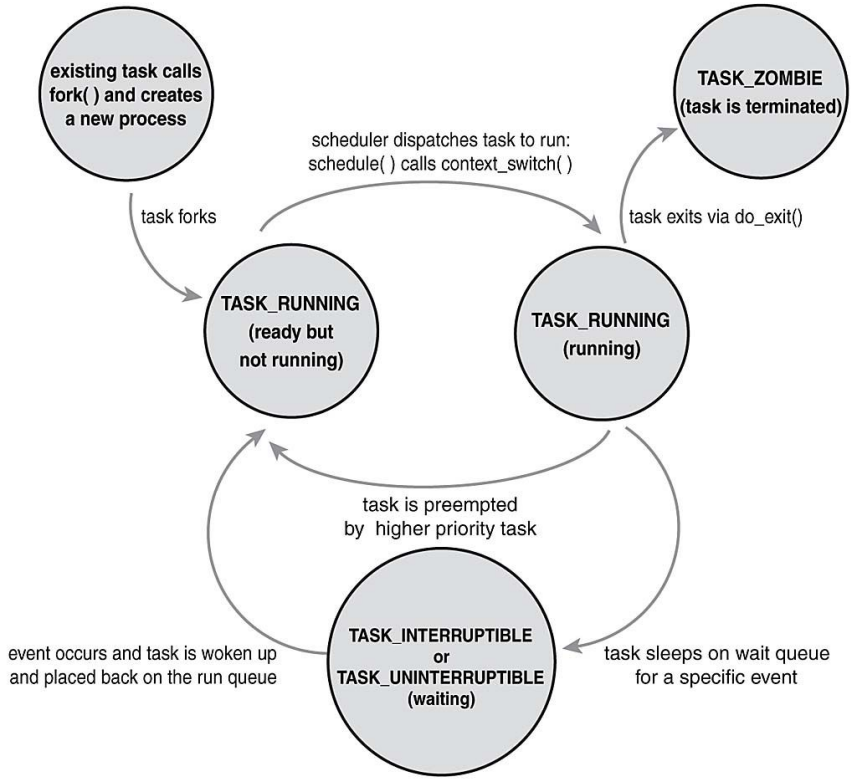
Moving to Process State…



Robert Love, *Linux Kernel Development*, 3rd Edition

States of Processes (source)

- State stored as a “flag”
- TASK_RUNNING
  - Can be executed
- TASK_INTERRUPTIBLE
  - Sleeping/blocked
  - Awaiting condition or signal
- TASK_UNINTERRUPTIBLE
  - Same as INTERRUPTIBLE, but ignores signals
- Which one could clearly (and does) go wrong?



Robert Love, *Linux Kernel Development*, 3rd Edition

Robert Love, *Linux Kernel Development*, 2nd Edition
https://www.informit.com/articles/article.aspx?p=370047

Altering States of Processes

- Kernel can modify state of process
- include/linux/sched.h

Linux 4.11 compat: set_task_state() removed

Replace uses of set_task_state(current, STATE) with set_current_state(STATE).

In Linux 4.11, torvalds/linux@642fa44, set_task_state() is removed.

All spl uses are of the form set_task_state(current, STATE). set_current_state(STATE) is equivalent and has been available since Linux 2.2.26.

Furthermore, set_current_state(STATE) is already used in about 15 locations within spl. This change should have no impact other than removing an unnecessary dependency.

https://git.with.parts/mirror/zfs/commit/8d5feecacfdcca29336209bdccd6493a5fa53576

/ include / linux / sched.h

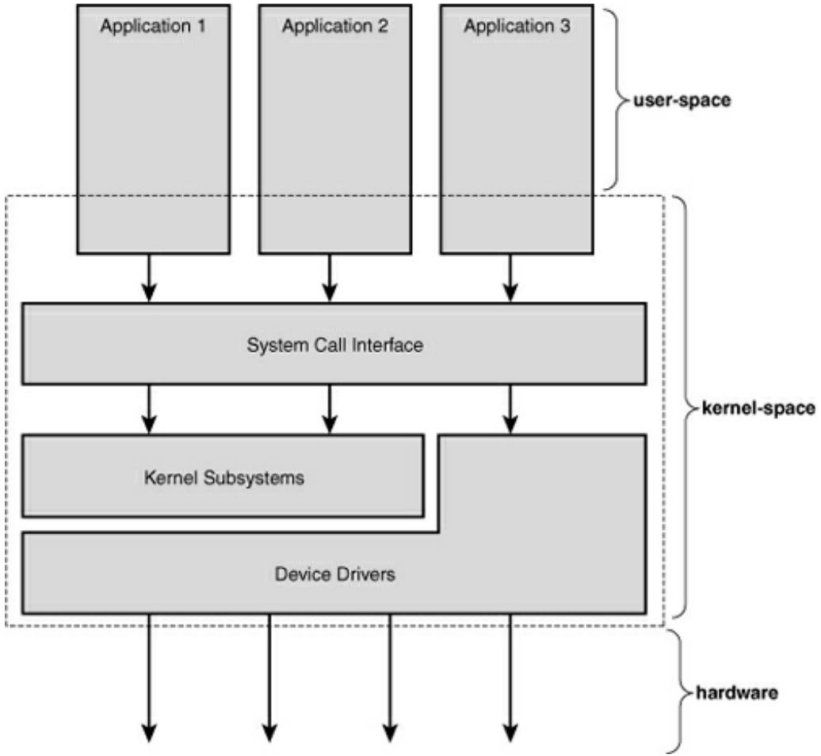
```
191     *
192     * Also see the comments of try_to_wake_up().
193     */
194    #define __set_current_state(state_value)
195            WRITE_ONCE(current->__state, (state_value))
196
197    #define set_current_state(state_value)
198            smp_store_mb(current->__state, (state_value))
199
200    /*
201     * set_special_state() should be used for those states
```

Linux 5.14.1 Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/v5.14.1/source/include/linux/sched.h#L116

Process Context

STOPPED HERE

- **User space** is where typical program running happens
- System calls made by the program into **kernel space**
  - Exception could also happen
- ‘Kernel is said to be “executing on behalf of the process” and is in *process context*”
  - Interrupt context is another context
- Legitimate `current` macro
- **System calls** & exception handlers are the exclusive ways into the kernel



Robert Love, *Linux Kernel Development*, 3rd Edition

Process Hierarchy

- **`init`**, the ultimate ancestor of all processes
  - Can you guess the PID that it gets?
  - PID: 1
  - **`systemd`** has replaced this
- 1 parent per process
- 0+ children per process
  - Children referred to as siblings

Process Descriptor (`task_struct`)

/ include / linux / sched.h All sym Search Identifier

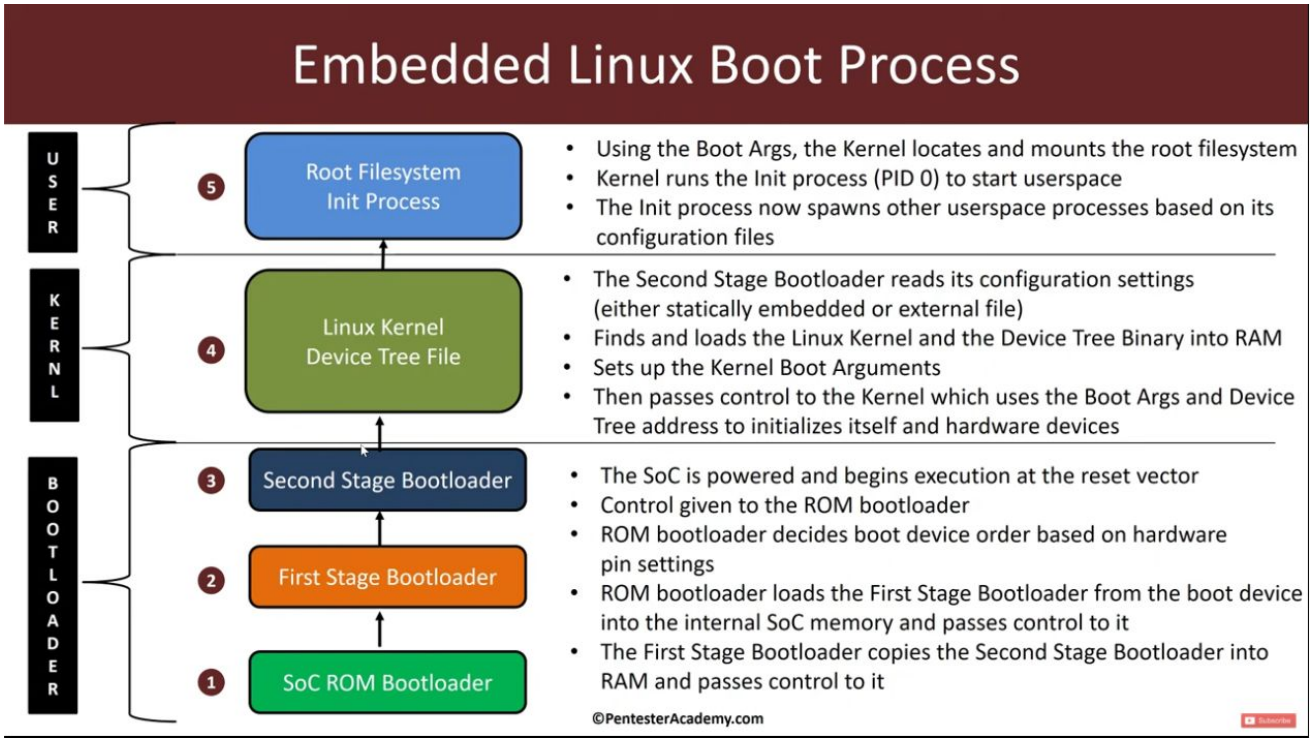
```
878         /*
879          * Pointers to the (original) parent process, youngest child, younger sibling,
880          * older sibling, respectively.  (p->father can be replaced with
881          * p->real_parent->pid)
882          */
883
884         /* Real parent process: */
885         struct task_struct __rcu        *real_parent;
886
887         /* Recipient of SIGCHLD, wait4() reports: */
888         struct task_struct __rcu        *parent;
889
890         /*
891          * Children/sibling form the list of natural children:
892          */
893         struct list_head                children;
894         struct list_head                sibling;
895         struct task_struct              *group_leader;
```

Linux 5.14.1 Source via Bootlin Elixir Cross Referencer
https://elixir.bootlin.com/linux/v5.14.1/source/include/linux/sched.h#L878

```
struct task_struct *my_parent = current->parent;
```

Robert Love, *Linux Kernel Development*, 3rd Edition

Boot Process (Slide From Pentester Academy TV)



Slide from: Pentester Academy TV, “Embedded Linux Booting Process (Multi-Stage Bootloaders, Kernel, Filesystem)”. (2018) https://www.youtube.com/watch?v=DV5S_ZSdK0s

Generating Processes

- Our first system call: `fork()`
- Duplicates living process
- **Parent** process calls `fork()` to create **child** process
- Notably, `fork()` takes advantage of **copy-on-write**

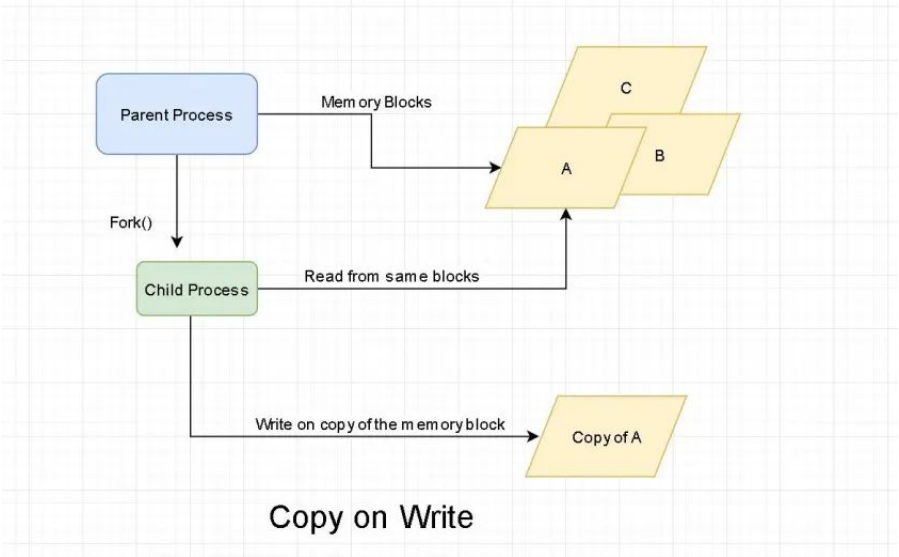
```
// SPDX-License-Identifier: GPL-2.0-only
/*
 *  linux/kernel/fork.c
 *
 *  Copyright (C) 1991, 1992  Linus Torvalds
 */

/*
 *  'fork.c' contains the help-routines for the 'fork' system call
 * (see also entry.S and others).
 * Fork is rather simple, once you get the hang of it, but the memory
 * management can be a [redacted]. See 'mm/memory.c': 'copy_page_range()'
 */
```

https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/kernel/fork.c?h=v5.14.1

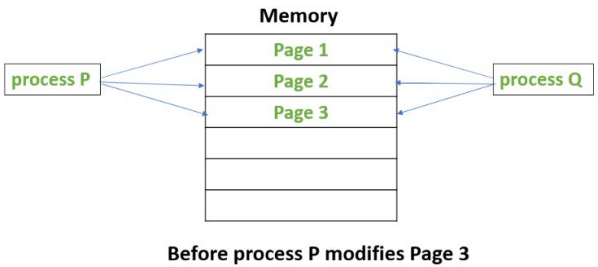
Robert Love, *Linux Kernel Development*, 3rd Edition

Copy-on-Write (COW)

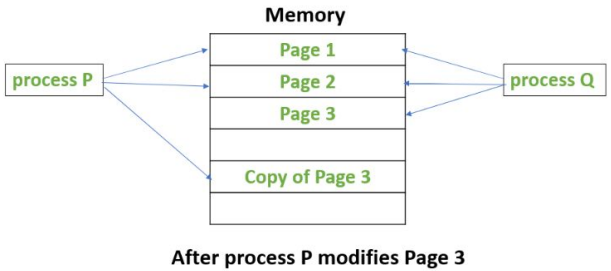
- Upon generation of child process by parent process, memory pages common to both
- If one process edits a page, it becomes a modified copy of the original page
- Also called **implicit sharing**



Gaurav Yadav, “What is Copy on Write and where is it used?”. https://www.learnsteps.com/what-is-copy-on-write-and-where-is-it-used/



Before process P modifies Page 3

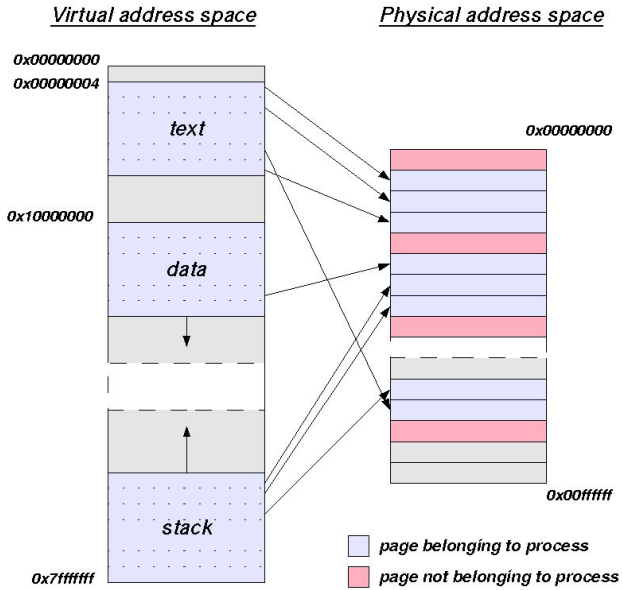


After process P modifies Page 3

User sumit_7, “Copy on Write”. https://www.geeksforgeeks.org/copy-on-write/

Copy-on-Write (COW): Reasoning

- Consider if copying happened first
  - Making copies when could just share
  - Copies may also be immediately disregarded
- COW increases efficiency
  - Copy later
  - May never need to copy
- `fork()` is fast
  - Copy page tables & generate PID



Author: Dysprosia. Copyright © Dysprosia. BSD license. URL to image and license: https://en.wikipedia.org/wiki/File:Virtual_address_space_and_physical_address_space_relationship.png

Robert Love, *Linux Kernel Development*, 3rd Edition

Virtual Memory - Page Table

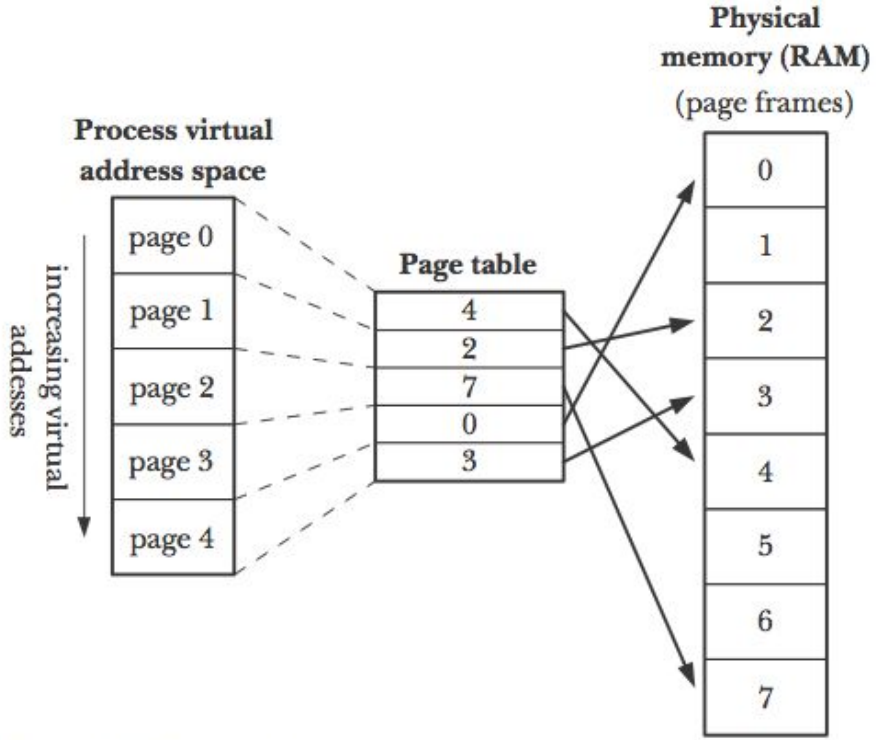
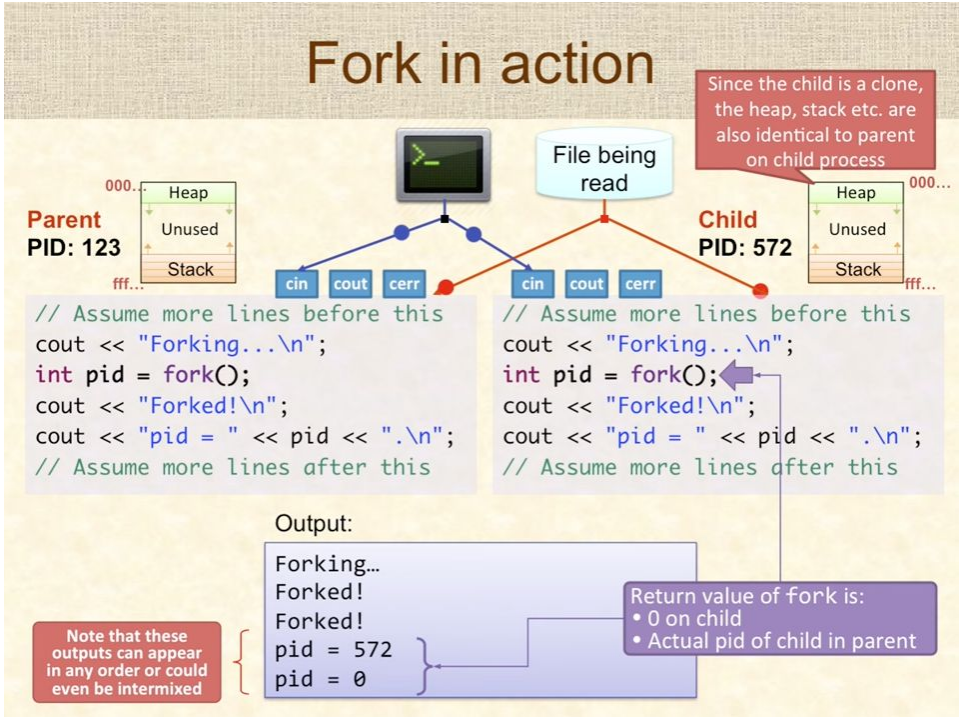


Figure 6-2: Overview of virtual memory

User: rmrf100, “The Linux Programming Interface 笔记 (三)”. URL: https://eyelublog.wordpress.com/2013/02/13/the-linux-programming-interface-%E7%AC%94%E8%AE%B0-%E4%B8%89/

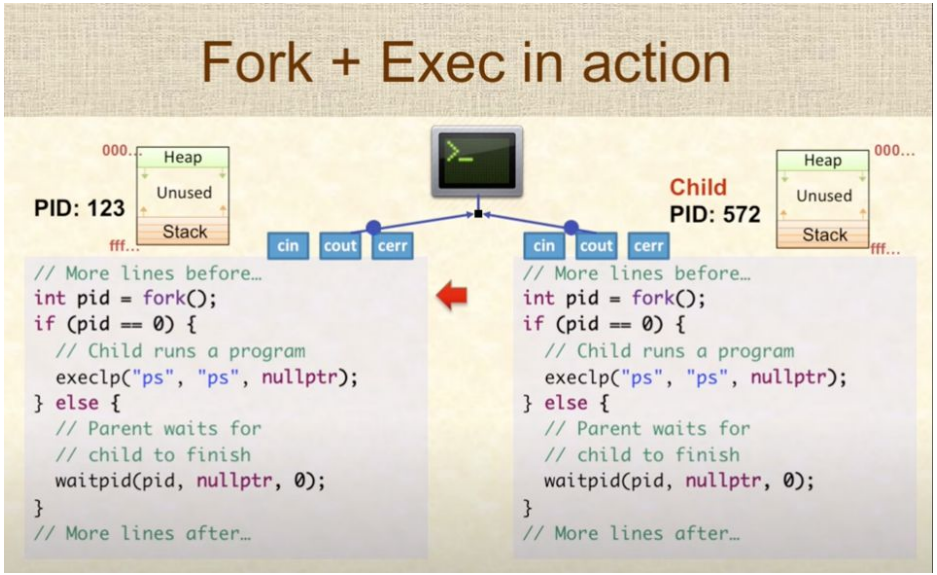
Fork

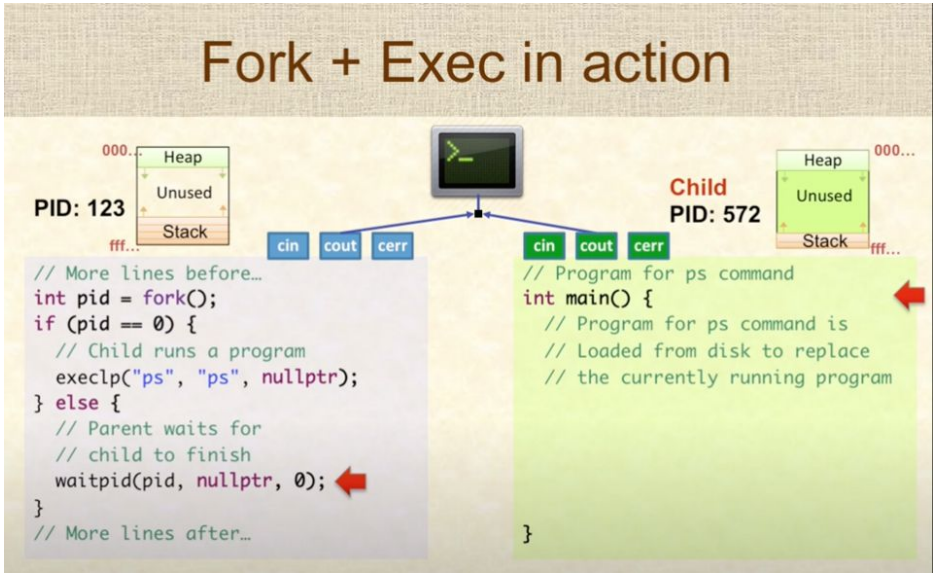
- `/kernel/fork.c`
  - Check out the source here
  - Trace code to `kernel_clone()`
- Returns:
  - 0 if child
  - PID of child if parent



User: Dhananjai Rao, “Fork and Exec in Linux”. https://www.youtube.com/watch?v=nwm7rJG90i8

Fork & Exec





User: Dhananjai Rao, “Fork and Exec in Linux”.
https://www.youtube.com/watch?v=nwm7rJG90i8

Kernel Threads

- Kernel-space processes
- Unlike normal processes, **no address space assigned to kernel processes**
  - `mm` literally a NULL pointer
- Schedulable
- Preemptable
- `ps -ef`

Robert Love, *Linux Kernel Development*, 3rd Edition

The Walking Dead As a Model For Linux Process Death

https://twitter.com/BestOfTWD/status/508725748551942145?s=20

Where Processes Go To Die (Or How to Get Away With [Process] Murder)

- Process terminates
- Process' resources freed via kernel
- Parent process informed via kernel
- Usually process uses system call `exit()` for termination
  - Could additionally be due to inability to disregard or manage signal/exception
- `do_exit()` takes care of most of the termination process
  - Link to source in `/kernel/exit.c`
- Final state: `EXIT_ZOMBIE`

Robert Love, *Linux Kernel Development*, 3rd Edition

Kernel Tree Subdirectories: See Description From David Adams

David Adams, “How to view and browse the linux kernel source?”.
https://linuxhint.com/browse_linux_kernel_source/

Next Lecture:

How Does Linux Schedule Processes?