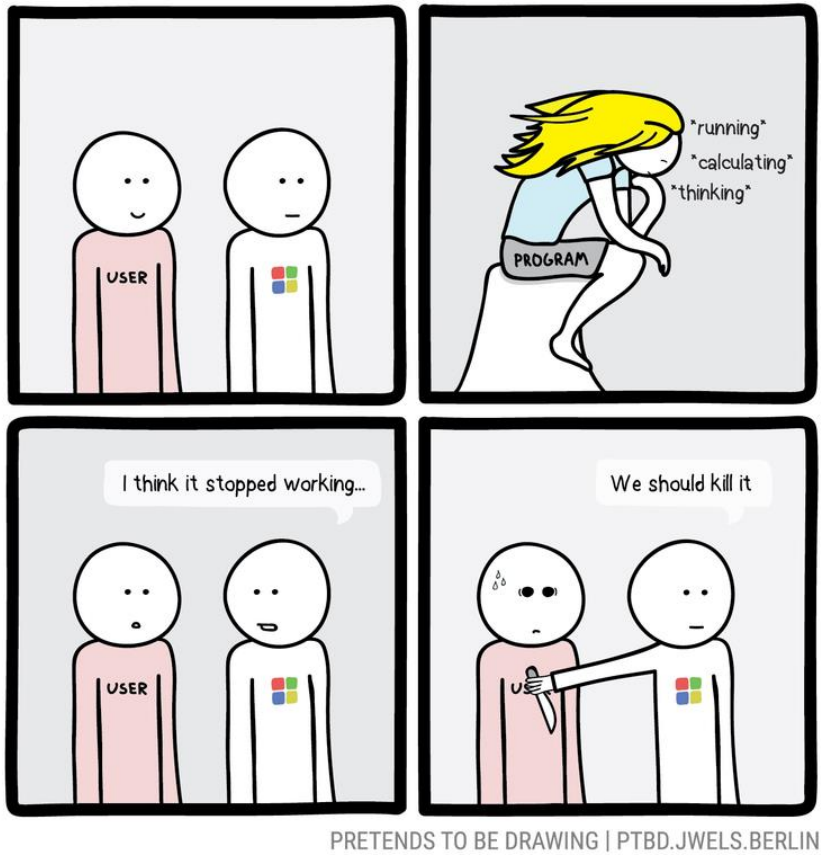
Processes II, Virtual Memory I

CSE 351 Autumn 2023

Instructor:
Justin Hsia

Teaching Assistants:

Afifah Kashif	Malak Zaki
Bhavik Soni	Naama Amiel
Cassandra Lam	Nayha Auradkar
Connie Chen	Nikolas McNamee
David Dai	Pedro Amarante
Dawit Hailu	Renee Ruan
Ellis Haker	Simran Bagaria
Eyoel Gebre	Will Robertson
Joshua Tan	



https://ptbd.jwels.berlin/comic/20/

Relevant Course Information

- HW24 due Friday (12/1), HW25 due next Wednesday
- Lab 4 due tonight, Lab 5 due Dec. 7

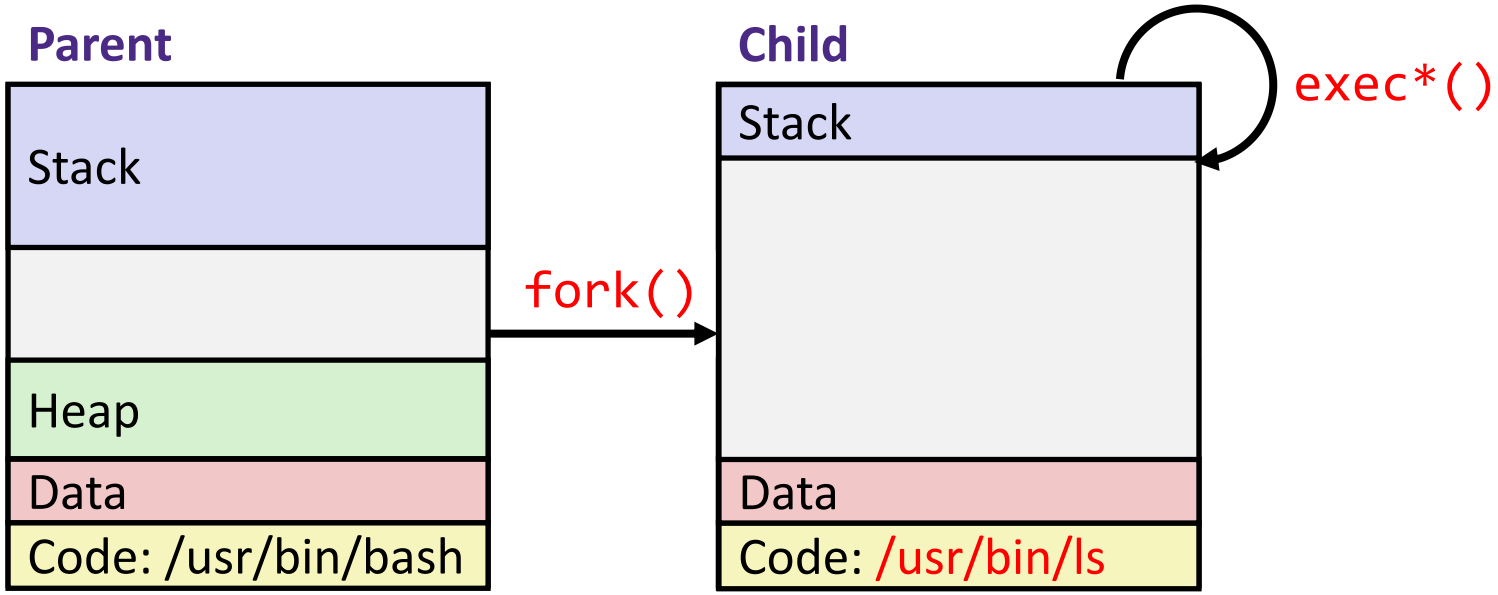
- No lessons in Week 11 – "normal" lectures

- Final Dec. 11-13
  - Structure will be very similar to the midterm
  - Not cumulative: focused on post-midterm material
  - Final review section on 12/7
  - Final review session planned for Zoom on 12/8
  - Regrade requests Dec. 17

Processes II, Virtual Memory I

Lesson Summary (1/4)

- The ***fork-exec model***
  - Every process is assigned a unique ***process ID*** (pid)
  - Every process has a parent process except for `init/system` (pid 1)
  - `fork()` returns 0 to child, child's PID to parent
  - `exec()` replaces the current process' code and address space with the code for a different program



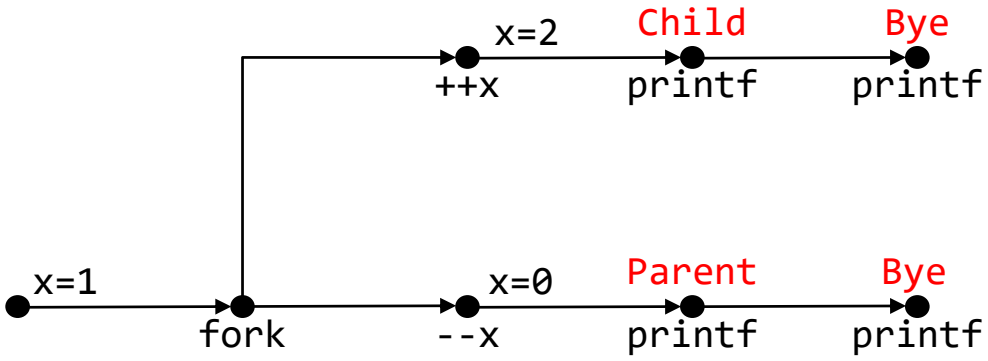
Lesson Summary (2/4)

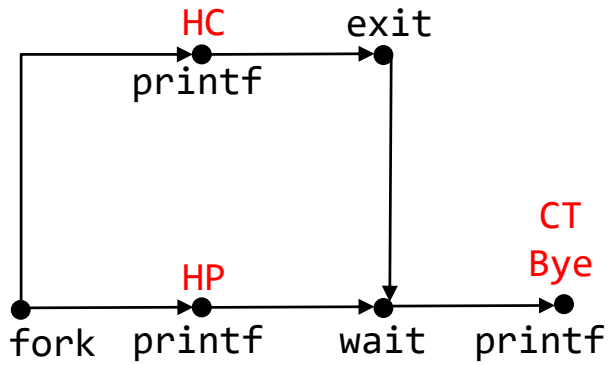
- Terminating a process
  - Return from `main()` or explicit call to `exit(status)`
  - Passes a ***status code*** (`main`'s return value or `exit`'s argument) to parent process
    - 0 for normal exit, nonzero for abnormal exit

- Processes and resources
  - A terminated (***zombie***) process still consumes system resources until ***reaped***
  - Child is reaped when parent process terminates or explicitly calls `wait`/`waitpid`
  - Orphaned children reaped by `init`/`systemd`

Lesson Summary (3/4)

- Concurrency and ***process diagrams***
  - Concurrently executing processes are scheduled non-deterministically by the operating system
  - A process graph is a useful tool for capturing the partial ordering of statements in a concurrent program
    - Vertices are program statements, directed edges capture sequencing *within a process*
    - Flexible visualization tool:

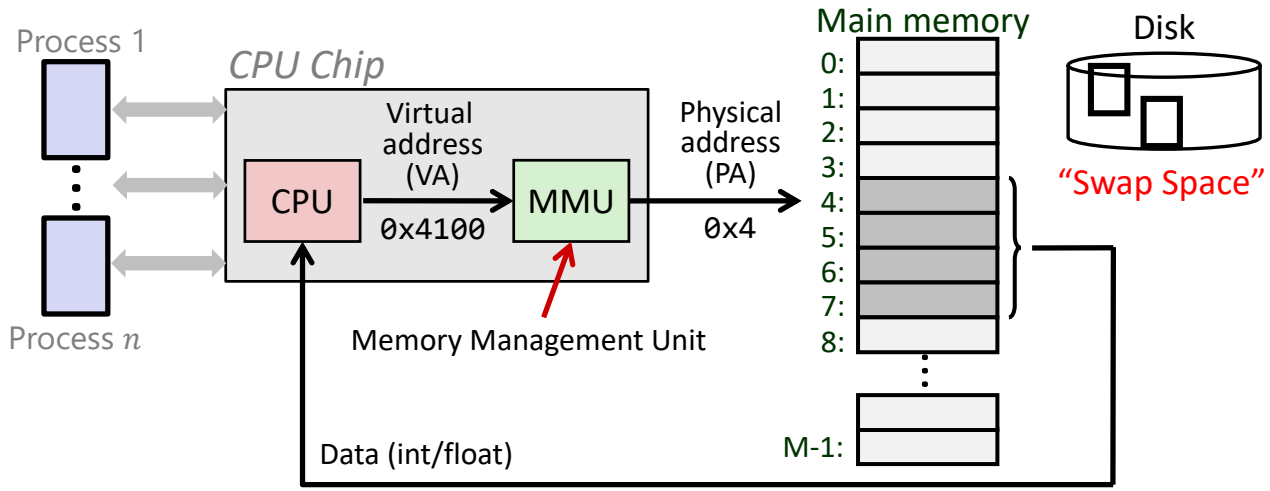




Lesson Summary (4/4)

- ***Virtual memory*** is software's perspective (*e.g.*, memory layout), ***physical memory*** is hardware's perspective (*e.g.*, memory hierarchy)

- Virtual memory manages the memory for multiple concurrently running processes (implements *protection* and *sharing*)
  - Each process has its own virtual address space that gets mapped into parts of the physical address space
  - When run out of physical address space, put least recently used data in disk



Lesson Q&A

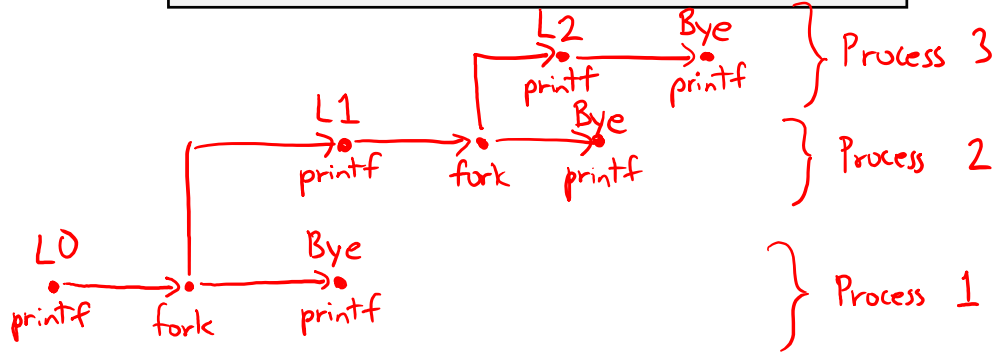
- Terminology:
  - Processes: fork-exec model, process ID, `exec*()`, `exit()`, `wait()`, `waitpid()`
  - `init`/`systemd`, reaping, zombie processes
  - Virtual memory: virtual addressing, physical addressing, indirection
- Learning Objectives:
  - Design process graphs to determine potential orderings of concurrent execution.
  - Write code that uses system calls to spawn, overlay, reap, and terminate processes on Linux x86-64.
  - Explain the benefits behind why virtual memory is used instead of only physical memory address space.
- What lingering questions do you have from the lesson?

Processes II, Virtual Memory I – Practice

Processes Practice Question

- Are the following sequences of outputs possible?

```
void nestedfork() {
    printf("L0\n");
    if (fork() == 0) {
        printf("L1\n");
        if (fork() == 0) {
            printf("L2\n");
        }
    }
    printf("Bye\n");
}
```



	Seq 1:	Seq 2:
	L0	L0 ← Process 1
	L1	Bye ← Process 1
	Bye	L1 ← Process 2
	Bye	L2 ← Process 3
	Bye	Bye ← Process 2/3
	L2 !	Bye ← Process 3/2
A.	No	No
B.	No	Yes
C.	Yes	No
D.	Yes	Yes
E.	We're lost…	

VM Practice Questions

- On a 64-bit machine currently running 8 processes, how much virtual memory is currently available?

word size is 64 bits, so $n = 64$ and $N = 2^{64}$ bytes per process.

$2^{64} \times 8 = 2^{67}$ bytes of virtual memory

- True or False: A 32-bit machine with 8 GiB of RAM installed would never use all of it (in theory).

word size is 32 bits, so each process has 2^{32} bytes = 4 GiB of virtual memory

however, we have more than 1 process, so we can easily use up all 8 GiB of physical memory

note: there are other limitations, (e.g., motherboard, OS) that restrict the maximum amount of usable RAM in practice

Processes II, Virtual Memory I – Context

Processes Demos

- How many processes are running on my computer right now?

- In Linux, the `ps` utility gives a snapshot of currently-running processes and `pstree` formats these as a tree
  - Can run `man ps` and `man pstree` for more info
  - Let's see a simple `pstree`
  - Let's check attu for some 351 zombie processes

Homework Setup

- In the MiniShell slide of HW24, you will be implementing a small command-line interface (like `bash`)
  - Should execute programs when passed the path of an executable and arguments, using `fork`, `execv`, and `wait`
- Command-line arguments in C:
  - `int main (int argc, char* argv[]);`
  - `argc` is the arg *count*, `argv` is an array of pointers to the arg *values* (C-strings)
- Process functions:
  - `execv` – 1st arg is path to executable (C-string), 2nd arg is argv
  - `wait` – only arg is a pointer to where child's status code will be placed

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the current lab

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support