EECS 482
Introduction to Operating Systems

Winter 2019

Manos Kapritsos

Thanks to Harsha Madhyastha and Peter Chen for the slides and notes
Unix process creation

- Fork clones an existing process

The dangers of cloning

---

**OK Hobbes, Press the button and duplicate me.**

**Are you sure this is such a good idea?**

**Brother! You doubting Thomases get in the way of more scientific advances with your stupid ethical questions! This is a brilliant idea! Hit the button, will ya?**

**I'd hate to be accused of inhibiting scientific progress... Here you go.**

**Boink**

**Scientific progress goes "Boink"?**

**It worked! It worked! I'm a genius!**

**No you're not, you liar! I invented this!**

---
Unix process creation

- Fork uses return code to differentiate
  - Child gets return code 0
  - Parent gets child’s unique process id (pid)

```c
if (fork() == 0) {
    exec ();         /* child */
} else {
    /* parent */
}
```
Subtleties of fork

What does this code do?

```c
for (int i=0; i<100; i++) {
    fork();
}
```
Multi-process issues

- How to partition physical memory allocation among processes?
  - Fairness versus efficiency

- Global replacement
  - Can evict pages from this process or other processes

- Local replacement
  - Can evict pages only from this process
  - Must still determine how many pages to allocate to this process
Thrashing

- What would happen if many large processes all actively used their entire address space?

- Performance degrades rapidly as miss rate goes up
  - Avg access time = hit rate * hit time + miss rate * miss time
  - E.g., hit time = .0001 ms; miss time = 10 ms
    - Average access time (100% hit rate) = .0001 ms
    - Average access time (99% hit rate) = ?
    - Average access time (90% hit rate) = ?
Solutions to Thrashing

- Buy more DRAM
  - Price per GB fallen by 4x since 2009

- Run fewer processes for longer
  - Example: Longer time slice
  - Reduces page faults
Working set

- Thrashing depends on portion of address space actively used by each process
  - What do we mean by “actively using”? 
- Working set = all pages used in last T seconds
  - Larger working set $\Rightarrow$ process needs more physical memory to run well (i.e., avoid thrashing) 
- Sum of all working sets should fit in memory
  - Only run subset of processes that fit in memory 
- How to measure size of working set? 
  - Periodic sweep of clock hand in LRU clock
Project 3

- **Process view:**
  - Every process has an address space starting from VM_ARENA_BASEADDR of size VM_ARENA_SIZE
  - When a process starts, entire address space is invalid
  - Process calls vm_map to make pages valid
  - Pages become invalid when process ends

- **Pager view:**
  - One process runs at a time
  - Sets up page table that MMU uses for translation
  - Handles vm_create, vm_map, and vm_fault
Project 3

- **Swap-backed pages:**
  - Global swap file shared by all processes
  - Pager controls where in swap file page is stored
  - Private to a process

- **File-mapped pages:**
  - Process specifies (file, offset)
  - Can be shared across processes
Project 3

- Do the project incrementally

- Swap-backed pages only without fork

- Then add support for fork and file-backed pages one after the other

- Pro Tip: Start with state diagrams
  - Separate for swap-backed, file-backed pages
Project 3: State Diagram

- For each unique state, consider:
  - Transitions? Read, write, clock, copy, ...
  - Attributes? Valid, resident, dirty, ...
  - Protections? Enable read, enable write?

**Mapped**
- Valid: Yes
- Resident: Yes
- Dirty: No
- Zero-filled: Yes
  
**Written**
- Valid: Yes
- Resident: Yes
- Dirty: Yes
- Zero-filled: No