The Process Model (2)

L41 Lecture 4, Part 3: More on VM
Dr Robert N. M. Watson
2020-2021
Virtual memory (quick, painful)

Virtual address space 1

Virtual address space 2

Physical memory

Heap

Stack

Library

Code

Kernel
So: back to Virtual Memory (VM)

- The process model’s isolation guarantees incur real expense
- The VM subsystem works quite hard to avoid expense
  - Shared memory, copy-on-write, page flipping
  - Background page zeroing
  - Superpages to improve TLB efficiency
- VM avoids work, but also manages memory footprint
  - Memory as a cache of secondary storage (files, swap)
  - Demand paging vs. I/O clustering
  - LRU / preemptive swapping to maintain free-page pool
  - Recently: memory compression and deduplication
- These ideas were known before Mach, but...
  - Acetta, et al. impose principled design, turn them into an art form
  - Provide a model beyond \( V \to P \) mappings in page tables
  - And ideas such as the message-passing—shared-memory duality
Kernel programmer view of VM

Machine-independent virtual memory (VM)

- Stack: Read/write, grows down, anonymous object
- Heap: Read/write, anonymous object
- Library: Read/copy-on-write, named object
- Code: Read/copy-on-write, named object

“vmspace”, “vm_map”

- “vm_map_entry”
- “vm_object”
- “vm_page”
- “vm_object” anonymous swap-backed VM object
- “vm_page” anonymous swap-backed VM object
- “vm_pager” swap pager
- “vm_object” shadow anonymous swap-backed VM object
- “vm_object” vnode VM object
- “vm_object” vnode “/bin/dd”

Machine-dependant physical map (PMAP)

- Physical memory
- “pmap”
- pde
- pde
- pte
- pte
- pte
- code
- code
- data
- data
- superpage
- page-table directory
- page-table entry
- pte
- pte
- pte
- pte

L41 Lecture 2 - The Process Model
Mach VM in other operating systems

- **Mach**: VM mappings, objects, pages, etc., are first-class kernel services exposed via system calls
- In two directly derived systems, quite different stories:

<table>
<thead>
<tr>
<th>Mac OS X</th>
<th>Although not a microkernel, Mach’s VM/IPC Application Programming Interfaces (APIs) are available to user programs, and widely used for IPC, debugging, …</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeBSD</td>
<td>Mach VM is used as a foundation for UNIX APIs, but is available for use only as a Kernel Programming Interface (KPI)</td>
</tr>
</tbody>
</table>

- In FreeBSD, Mach is used:
  - To efficiently implement UNIX’s `fork()` and `execve()`
  - For memory-management APIs – e.g., `mmap()` and `mprotect()`
  - By VM-optimised IPC – e.g., `pipe()` and `sendfile()`
  - By the filesystem to implement a **merged VM-buffer cache**
  - By **device drivers** that manage memory in interesting ways (e.g., GPU drivers mapping pages into user processes)
  - By a set of VM worker threads, such as the **page daemon, swapper, syncer**, and **page-zeroing thread**
For next time

• Review ideas from the first lab report

• Lab 2: DTrace and IPC
  • Explore Inter-Process Communication (IPC) performance
  • Leads into Lab 3: microarchitectural counters to explain IPC performance

• Reading: Ellard and Seltzer 2003