Today's plan

- Virtual Memory: page faults, copy-on-write, free page management, kernel memory, segmentation
- Virtual Memory in Minix
- Virtual Memory in Linux
- Virtual Memory in seL4

Page Faults

- On every access, permissions are checked
- If the permission is wrong, or the page is not mapped, the access fails
  - and control goes to the kernel
- Some virtual pages are never mapped
  - e.g. the location of NULL
- Some virtual pages are saved to disk (swap)
  - and so they are not mapped
  - but will be when there is a page fault

Fork and Copy-on-Write

- for fork, map all the virtual pages of the two processes to the same physical pages
  - simply copy the page tables
  - but mark all the pages read-only
- when there is a page fault due to a write:
  - if the page really should be writable (not code)
  - make a copy of the page, mark it writable

Finding a free page

- Once pages are written to, they are *dirty*
  - hardware can mark pages dirty, or
  - again, software can mark pages read-only
    - then mark them dirty when a page fault shows a write
- pages that are not dirty can be reused immediately
  - after updating the appropriate page tables
- pages that are dirty may be reused after writing them to disk
  - this applies to file buffers or process pages

Kernel Memory

- The kernel may or may not use virtual memory
- Important: the page fault handler should never cause a page fault!
- So, some pages are always kept in memory, and at fixed addresses: *pinned* pages
  - pinned pages can never be swapped out

Segmentation: Another Way to do VM

- every memory access is associated with a segment register
- the value in the register is added to the virtual address to give the physical address
  - the segment register may also include a limit value, to detect illegal accesses
- segments have variable sizes, unlike pages
  - so memory management may lead to fragmentation
- the x86 architecture used segments to address 1MB of memory with only 16-bit addresses

Virtual Memory in Minix

- `umap_local`, `umap_remote`, `umap_bios`, starting on p. 760
- a virtual address `vir_addr` corresponds to a physical address `pa` based on two per-segment fields, `mem_vir` and `mem_phys` (see lines 10015 and 10018.)
- possible segments include text, data, and stack
- `vir_addr - mem_vir` is the offset *o* into the segment
- *p* = *o* + `mem_phys` is the physical address
- computation on lines 10015-10019
- all computations aligned on "page" (click) boundaries

Virtual Memory in Linux

- Virtual Memory Areas, VMAs
- include/linux/mm_types.h
  - one `struct mm_struct` per task
  - one or more `struct vm_area_struct` per task
    - fields of this struct are assigned to specific cache lines
    - start and end are the virtual start and end addresses
    - VMAs kept in a doubly-linked list
    - `struct vm_operations_struct` is a list of pointers to functions to open, close, split, remap, fault, etc
  - `struct page`: 24/48 bytes per physical page
- vm_ops and flags defined in include/linux/mm.h
  - VM_IO: memory-mapped I/O area
  - VM_LOCKED: pinned pages (`mlock()` system call)

VM of an actual process in Linux

```
esb@maru:~/ltmp/linux-4.20.5$ sleep 1000 &
[4] 8090
esb@maru:~/ltmp/linux-4.20.5$ pmap 8090
8090:   sleep 1000
000055eb0e79a000     28K r-x-- sleep
000055eb0e9a1000      4K r---- sleep
000055eb0e9a2000      4K rw--- sleep
000055eb105bc000    132K rw---   [ anon ]
00007f21746b4000   2936K r---- locale-archive
00007f2174992000   1948K r-x-- libc-2.27.so
00007f2174b79000   2048K ----- libc-2.27.so
00007f2174d79000     16K r---- libc-2.27.so
00007f2174d7d000      8K rw--- libc-2.27.so
00007f2174d7f000     16K rw---   [ anon ]
00007f2174d83000    156K r-x-- ld-2.27.so
00007f2174f8f000      8K rw---   [ anon ]
00007f2174faa000      4K r---- ld-2.27.so
00007f2174fab000      4K rw--- ld-2.27.so
00007f2174fac000      4K rw---   [ anon ]
00007ffc7fcaf000    132K rw---   [ stack ]
00007ffc7fcde000     12K r----   [ anon ]
00007ffc7fce1000      8K r-x--   [ anon ]
ffffffffff600000      4K r-x--   [ anon ]
 total             7472K
```

```
$ cat /proc/8090/maps

55eb0e79a000-55eb0e7a1000 r-xp
00000000 08:05 23986363
       /bin/sleep

55eb0e9a1000-55eb0e9a2000 r--p
00007000 08:05 23986363
       /bin/sleep

55eb0e9a2000-55eb0e9a3000 rw-p
00008000 08:05 23986363
       /bin/sleep

55eb105bc000-55eb105dd000 rw-p
00000000 00:00 0
       [heap]

...
```

Virtual Memory in seL4

- seL4 Reference Manual, Chapter 7
- a `Page` represents a physical memory frame
  - Pages support `map`, `unmap`, `remap`, `GetAddress`
- ASID is an Address Space ID, usually mapped to a hardware resource
- two-level page tables for 32-bit architectures, 4-level page tables for 64-bit architectures
  - some levels use 8 bits, some 9, some 10, some 12
- Page faults send a message to the exception handler for the thread
  - replying to the message restarts the thread