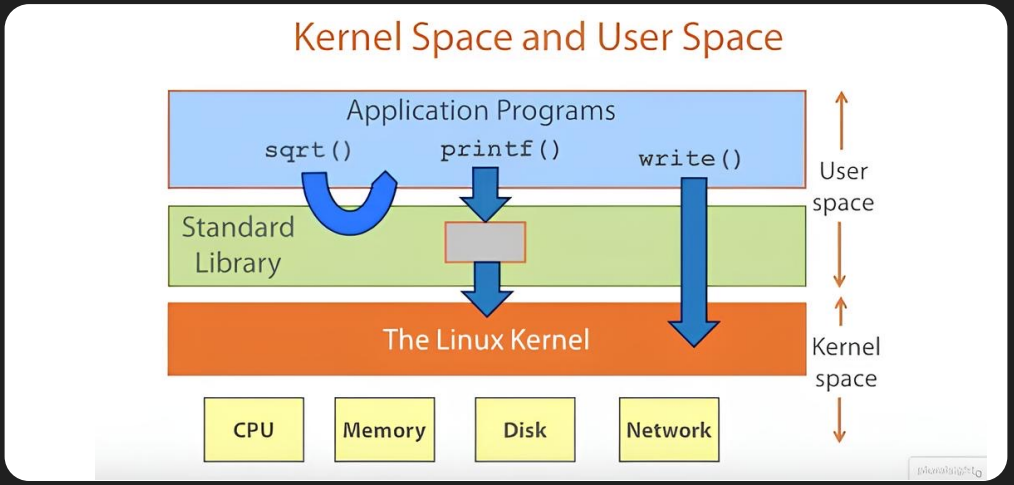
SPECTRAL:
TECHNOLOGIES

WEEK 7

Basic Linux internals

OS role

- The middleman between applications and hardware
- Provides convinient abstractions
- Network stack implementation
- Resource management
- And so on



Kernel vs user mode

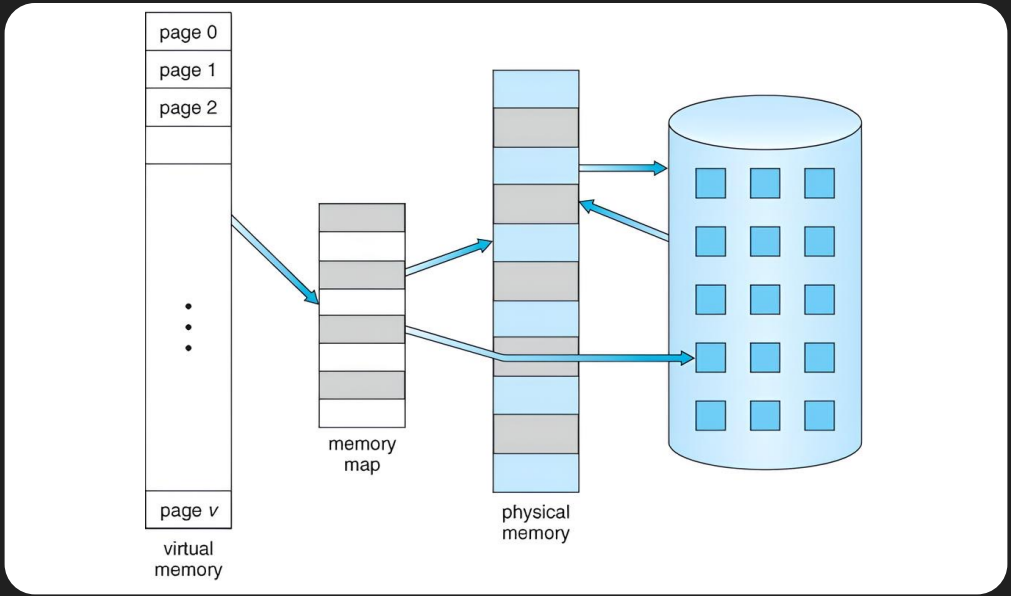
- x86 provides 4 cpu modes (*rings*), only 2 are used actively: supervisor (kernel) mode and user mode
- These modes provide different set of privilleges
- Virtual address space is also divided into kernel and user space
- Only linux kernel and some drivers run in kernel mode
- Regular applications can run in user mode only
- Linux provides api to access kernel mode via *syscalls*

Syscall

- Kernel space C function
- In x86-64 limited to 6 arguments, all passed via registers
- On old systems `int 0x80` was used for call
- On current systems `syscall` instruction is used
- The switch to kernel is heavy (**~300 ns**)
  - Also introduces same performance penalties regular context switch does.
- vDSO is used to make following syscalls execute in userspace:
  - __vdso_clock_gettime
  - __vdso_getcpu
  - __vdso_gettimeofday
  - __vdso_time
- `strace` can be used to check which syscalls a program calls

Virtual memory

- Recall virtual memory abstraction from previous lectures
- If some page cannot be found in the page table, *minor* or *major* page fault is issued



Minor page fault

- The page is in RAM already, but its not in the page table for this process
- Happens due to lazy allocation of pages and sharing of pages
  - malloc e.g
- Costs about 1000 cycles (a few hundred ns)

Major page fault

- This page isn't in RAM and needs to be loaded
- Can happen due to *swapping*
- Can also happen as the result of referencing mapped memory region from disk (mmap)
- So can take arbitrary amount of time

mmap

- System call, that maps files or devices into process memory
- Another way of doing I/O
- Does demand paging, so the actual reads happen lazily
- mmap vs read
  - mmap syscall costs more
  - Less copying of data with mmap
  - Previously mapped pages may remain for some time, so you can have free file reopening
- Allows to share memory between a parent and a child process directly
  - shm_open for independent processes
- *Anonymous* mapping may be used for memory allocation
  - glibc's allocator uses mmap for allocations of more than 128Kb

Processes and threads

- The main difference is that threads and the process they belong to share memory and processes don't do that
- In Linux there is almost no difference internally between a process and a thread
  - Both processes and threads are represented by the same "process" (in kernel terminology), i.e. something, that can be scheduled to run
  - threads are the same "processes", but sharing some things like memory space and etc
  - So scheduler in Linux schedules threads from the user's perspective
- Most properties (like affinity via *taskset*, priority) are actually applied to threads
  - we'll see some examples in the optimization's lecture
- A lot of process info can be accessed via virtual file system `/proc`
  - cmd args, env values, cwd, exe link, memory maps, etc
- `ptrace` syscall can be used to read and write to running process memory (gdb)

IPC

- There are multiple ways to perform inter-process communication in Linux
  - files
  - pipes
  - message queues
  - sockets
  - signals
  - shared memory
- Shared memory is the most efficient one, but synchronization is harder
- Basically maps the same page into multiple processes' address space
  - Once its mapped, it can be used as a regular chunk of memory
- Linux also provides `/dev/shm`, which is a filesystem-like storage in memory (and basically, a backend for shared memory)

Interrupts

- Event, that alters the normal execution flow of a program and can be generated by hardware devices or by the CPU itself
  - synchronous (usually named exceptions), generated by executing an instruction (division by zero, syscall)
  - asynchronous (interrupts or *IRQ*), generated by an external event (network packets have arrived)
- Every IRQ has associated Interrupt Service Routine (ISR), which is executed after a context switch to it
  - IRQ also have affinity (which cores can execute the corresponding ISR)
  - Due to the context switch, its one of the elements to tune on low-latency setups

Additional materials

- https://0xax.gitbooks.io/linux-insides/content/