Karlsruher Institut für Technologie

2. Operating Systems Tutorial

Multiprogramming, Processes, Interrupts

Péter Bohner | 2023-11-08

Inhaltsverzeichnis

Organisation

- Everything is in ILIAS
- My website `https://bohner.me/teaching/ws2023_ostut/`
- ATIS: `https://www.atis.informatik.kit.edu/513.php`
- Compiler Explorer: `https://godbolt.org/`
- Feedback: `https://forms.gle/9CwJSKidKibubran9`
- Contact: email: `peter.bohner@student.kit.edu`, discord: `xzvf`
- Does everyone have a proper development environment?

Recap

- 3 major tasks/responsibilities of an OS
  - Abstraction, Resource management,Protection
- difference between kernel and user mode
  - Ability to execute privileged instructions

Switch to terminal

Multiprogramming

What is that?

Allow multiple processes to run concurrently (in parallel?)

How can they share a CPU? How can you make them take turns?

- Two possible broad categories:
  - Cooperative Multitasking
  - Preemption

Cooperative Multitasking

How does Cooperative Multitasking work? Do you know any system using it?

- Processes voluntarily yield control so another one can take over
- When?
  - Typically when they are blocking on some I/O or have finished their job for now
- Do you know any language or system using that concept?
  - await in JavaScript or Python
  - Old Mac OS
- Can you think of any problems? Any advantages?
- A bad actor or buggy program can bring the whole system to its knees!
- It can lead to easier programs, as you can just assume you are never interrupted
  - Usage on small resource-constrained embedded systems

Preemtive Multitasking

How does Preemption work?

- Processes get periodically interrupted by Timer-interrupts
- This transfers control to the OS, which can choose to schedule another process instead

Advantages?

- Process A runs while Process B waits for an I/O device to complete its task
  - Better CPU and I/O utilization if you have a mixed workload
- What do the I/O devices and CPU need to support for this to be an improvement?
  - Interrupts, else the CPU needs to continuously poll

Challenges?

- More complex
- Protection: How do you separate all system activities from each other?
- Scheduler: Fairness and accounting

CPU and I/O-bound

What are CPU and IO-bound Processes?

- CPU-bound: Process that rarely invokes I/O, unlikely to block, likes your CPU very much
- I/O-bound: Often blocks to wait for I/O, performs short CPU-bursts in between

Why do we want both I/O and CPU bound processes?

Resource usage: To be able to use CPU and IO time effectively.

Address Spaces

Why can't a process read another process's memory?

- Each process has it's own address space
- Let's have a look at two processes...
  - Address `0x2345` in Process A is not the same as Address `0x2345` in Process B
  - "If you can't name it, you can't touch it"
- You will learn about this in way more detail in later lectures

Fork

What does fork do?

- Creates a mostly identical child process:
- Same address space layout
- Same data (as if it was all copied over)
- Share open file descriptors
- But the child has its own PID and address space
- `... man fork`

Exercise

Write a small C program that creates a child process. Each process shall print out who it is (i.e., parent or child). The parent shall also print out the child's PID and then wait for the termination of its child.

Fork/exec

Is fork sufficient to create a small shell?

- No, you also need to load the other program in the child
  - `exec(l|lp|le|vp|vpe)`

Kernel Entry

What three events lead to an invocation of the kernel?

Interrupts, Syscalls, Exceptions

What is an Interrupt?

- A signal generated outside the processor (typically from an I/O device)

What is an Exception?

- Caused by the CPU
- Inform the operating system of an error condition during execution (E.g. division by zero, page faults)

What is a System Call?

- Explicit call from user application into OS
- Used to interact with the OS (e.g. to read a file)

Kernel Entry - Categorization

	Sync	Async
Voluntary	System call	Exception
Involuntary	???	Interrupt

Exceptions

Can you think of an exception the OS can't repair?

i.e. it will terminate the process :(

- Divide by Zero
- The wrong kind of page fault (access forbidden or invalid memory)

Can you think of an exception the OS can repair?

- The right kind of page fault (e.g. access memory currently paged out to disk)
- When the OS is run in a virtual environment, it might try to access privileged instructions
  - CPU throws an exception
  - Virtual machine monitor handles that and executes the action on the real physical device (if allowed)

Trap instruction

How is the trap instruction related to system calls?

1. trap is executed and switches the CPU to kernel mode
2. The CPU continues execution at a hardware dependent location (where exactly depends on your architecture and a few other things, but the important part is that the operating system registered itself there)
   - trap can be seen as a low-level instruction making a higher-level mechanism (syscalls) possible

- You could also synchronously trap into the kernel using a division by zero or other fun stuff. That isn't done normally though...

Syscalls

What is a system call number? Why do you need them?

- The user-level application can't just call a kernel function!
  - The kernel somehow needs to dispatch to the correct function
  - Array of function pointers with system call numbers as indices

How can you pass parameters to the kernel when executing a syscall?

- Registers
- Stack
- Main memory (and location in register)
- see `man 2 syscall` . . .
- There is also implicit context (which process caused the syscall?)

Syscalls and Libraries

How do syscalls relate to libraries? Abstraction? How do you call them?

- Syscall mechanism is usually hidden behind library functions (libc on linux)
  - on most platforms only these are considered stable (most BSD, OSX, **not linux**)
- The functions take care of preparing arguments, setting the syscall number and trapping in an efficient way
- Windows uses the ntdll.dll / Win32 API

Syscalls - Security

What do we need to keep an eye on when performing a system call in the kernel?

- Validate all parameters! Twice! And a third time to be sure.
- Otherwise you might have some arbitrary reads/writes into system memory

End

Are there any questions?

Leave anonymous feedback

https://forms.gle/9CwJSKidKibubran9

Bye!

Until next time, **Wed, 15.11.2023**!