Operating Systems

Roles of the OS

Referee

Controls/orchestrates/shares access to resources and hardware by multiplexing. Used for CPU, memory, devices and more. Needs to protect applications from each other through memory, CPU, etc. Must still allow communication. Fairness, no starving and everybody makes progress. Be efficient while doing all of that and guarantee real-time performance.

Illusionist

Creates illusion of real resources which seem to be physical but are simpler, larger, better. Used for processor, storage, network, links, etc. virtual memory, virtual machines, files, windows are examples.

Glue

API/Interfaces between components. Virtual machine interface, physical machine interface. High-level abstractions, extends hardware.

Kernel

Privileged mode is strictly separated on a hardware level from user mode. Kernel is the part that runs in privileged mode. Microkernels outsource a lot of operations to user mode. Kernel typically event-driven server, reacts on syscalls, interrupts and traps. Applications can use kernel functions via syscalls or a more convenient system library. Kernel should not be bloated -> less bugs, can't get stuck in kernel on heavy load.

System Calls

On syscall, enter privileged mode and return when done. Argument passing varies: As registers, in memory, in stack. System library is declares C functions for syscall.

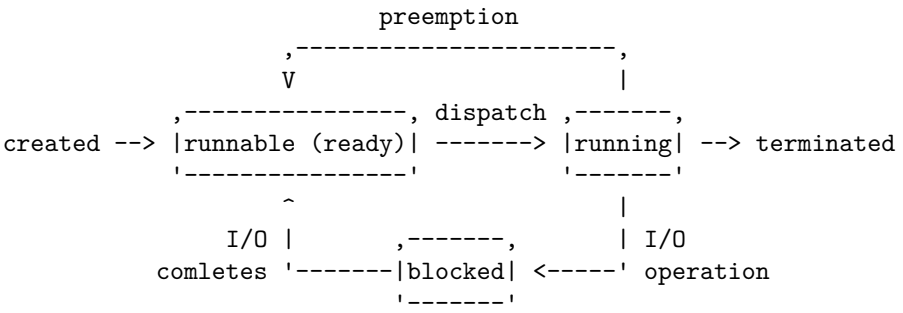
Processes

Process is the execution of a program with restricted right, like a VM.

Parts

- Virtual Processor: Has address space, registers, instruction pointer, program counter
- Proogram text, program data
- OS stuff: open files, sockets, CPU share, security rights

Lifecycle



Multiplexing Time- and space-division of processor.

Process Control Block (PCB) In kernel structure, holds name/registers/page table/files, etc.

Creation

Explicit (Windows): Syscall with all arguments

Fork/exec: `fork` current process, copying all arguments (PCB specifically). Use `exec` to set new binary to execute. Creates process tree (show with `pstree`)

Example:

```
pid_t p = fork();
if (p < 0) {
    // Error
    exit(-1);
} else if (p == 0) {
    // In child
    execlp("/bin/ls", "ls", NULL);
} else {
    // In parent, `p` is pid of child
    wait(NULL); // Wait for child to complete. Child can't exit until this is called, "Zomb
    exit(0);
}
```