Intro

ABI vs system call interface - ABI is the executable version of program, compiled for that architecture and contains instruction for the ISA. The ABI has the system calls for each architecture, and uses the sytem call interface for the system for which it was compiled for.

Relocation problem - virtualization hides physical address from process, allows it to continue running if data moves as long as base address of translation changes too.

Shared libraries data - shared libraries cannot be written to since modifying the library would change the behavior of the other processes (illegal since processes must be isolated).

Running on programs on different machines - same API and ABI -> copy the compiled program to the other machine. diff API same ABI -> copy source code to new machine and recompile. diff API and ABI -> hard to do, if APIs similar try to modify source code to match new API and then compile on new machine, or if ABIs similar try to modify compiled program.

Federation framework - standard interface used for compatibility and programs/device drivers with the similar functionality.

Trap table - actions for the OS when an exception occurs, contains pointers to handling routines indexed by a trap #. Loaded at boot time and not changed.

Process

Information hiding - abstract away hardware and other complications from developer's view

Copy on write linux fork - instead of copying and loading the entire data and code segment into the child process, they are done on demand as needed. Both processes (different PIDs) can point to the same data, until a process has to write, at which time a new copy of the data (not code) is made. This saves space potentially and reduces overhead of spawning new threads.

Shared memory IPC and data linux fork - after fork when the child writes to a page, the virtual page # no longer points to the page frame number that the parent points to. Instead it makes a copy at a different physical address and has a different value. In shared memory IPC the VPNs of the thread are mapped to the same PFNs, so changes in one thread are reflected in the other. Shared memory IPC faster since less system calls, but are create concurrency issues and are limited to same machine.

Flow control sockets vs shared memory - OS cannot regulate shared memory since no system calls after allocation compared to system call for every read/write in sockets.

Scheduling

Context switch - save PS and virtual address space (reg, stack, heap, resources, VM, page table)

LDE - direct execution lets programs mostly stay in user mode (save overhead), limited so that programs don't break the OS, do something illegal, take up all CPU time. Programs have restrictions and have to give control to kernel for priviledged instructions.

Timer interrupts - allow for preemptive scheduling, allows OS to stop a running program. This allows OS to take back control of buggy programs and regulate scheduling round robin style

Soft real time preemption - usually yes, preemption lets the scheduler stop processes and prioritize other ones, which is important if there are preferred deadlines. If the OS knoms the runtimes, preemption is not needed and causes extra overhead.

Turnaround time - Preemptive SJF minimizes and needs preemption to stop long running processes

MLFQ - if finishes early regularly, move to short queue. If finishes late regularly, move to long queue frequently. Done by counting how often process finishes executing.

Memory

Page vs page frame - page is VM abstraction to sore data in memory or disk. If in memory, it will be stored in a page frame, which is an open spot in memory to fit the page in.

Worst fit external fragmentation - worst fit finds the largest free area and allocates space to a process. This leaves largest chunks possible to be allocated to other proceses. Best fit will leave tiny slices in free memory, which leads in external fragmentation.

Working Sets and Page Stealing - work by finding the optimal number of pages to allocate to a process (limited by physical constraint of memory), evicting pages if there are too many, and stealing pages from another process if there are not enough.

Dirty bit - if a page on mmeory has been modified, then when it is modified the dirty bit tells the OS that the copy on disk must also be updated instead of just moving to swap space. To reduce I/O in dirty page eviction, we have the LRU approx clock algorithm prioritize operations with bit=0 and dirtyBit=0 over bit=0 and dirtyBit=1. We can also cluster dirty pages together and evict/write to disk in a group for effiency.

TLB miss - TLB miss means physical page # (PFN) for virtual page # (VPN) is not in TLB. If the page is in memory, the TLB will be updated with the corresponding PFN and the instruction is retried, resulting in a TLB hit. If the page is on disk, the process is blocked and the page is loaded into memory (page fault). The TLB is updated with the new PFN and the instruction is retried, resulting in a hit. if the page is not on disk either, then the page is invalid.

ASID - means new process doesn't need to flush the TLB if the ASID matches (reduces context switch)

Page size - decreasing page size lowers internal fragmentation (50% of size), no external fragmentation anyway. Disadvantage is less spatial locality, more jumping around, more swaps.