ABI vs sys call interface - ABI is the executable version of program, compiled for that arch and contains instr for the ISA. The ABI has the sys calls for each arch, and uses the sytem call interface for the sys for which it was compiled for. The sys call interface is a subset of the ABI.

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 diff 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 instr - Hit the trap, raise to kernel. 1st level handler saves registers and saves the PC and PS, gathers info, selects 2nd level handler – 2nd level handler actually deals with the problem such as handle the event, kill the process, return-from-trap Trap table: The trap table is used to specify what actions the operating sys should take when a particular exception occurs. It will contain pointers to exception handling routines, indexed by a number specifying the particular exception that occurs. It is consulted whenever an exception actually occurs.

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 (diff 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 diff physical address and has a diff 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 sys 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 sys calls after allocation compared to sys call for every read/write in sockets.

Context switch - save PC/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 instrs.

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

Non preemptive - sometimes does not require scheduling overhead, less context switch (none during run)

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

Buddy allocation - give base 2 sized pieces, ask buddies if they are also free to coalesce.

Page vs page frame - page is VM abstraction to store 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.

Segmentation and paging - Segmentation arbitrarily sized ranges of the AS to be used for diff parts of process. Paging is used to divide allocated memory space into smaller pieces for VM sys. Segmentation and paging might be used together to both specify the portions legally accessible and paging allows VM in a more flexible way.

Worst fit external fragmentation - worst fit finds the largest free area and allocates space to a process. This leaves largest chunks to be allocated to other proceses. Best fit will leave 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 (so we don't have to save real timestamps by using a separate bit) 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 - lookup in page table (if invalid give segfault), check permission (if invalid give protection fault), if in memory copy to TLB and retry, if not in memory create page fault. If no empty page, kick out a page. Once empty page available, copy from disk to memory and retry.

Invalid bit - invalid PTE means no page for allocated for this process, invalid TLB means not your page. PTE has a present bit which indicates if in memory or disk

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.