Lecture 8: VM & Memory Management

VM(idea: indirection) provides fine-grained and dynamic management of address space. Each process has its own virtual address space, contiguous and linear. Virtual address space map to physical address space in unit of pages.

Page table: translate virtual address to physical address, each process has its own, managed by OS, used by hardware. One PTE per virtual page, lower bits are not translated. PTE includes valid bit, physical page number and metadata (RWX).

VM: 1. Memory management, easy to share pages across processes

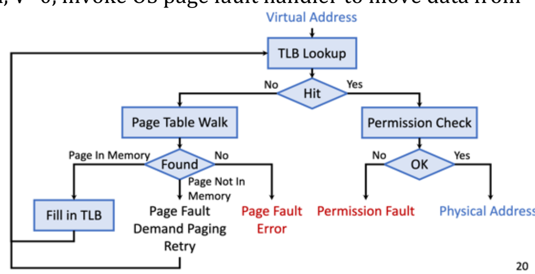
2. Memory protection, can't access physical memory not mapped to its virtual space, has permission bits;

3. data caching: a subset of allocated are mapped.

VM = Unallocated + Mapped(V=1) + Swap-out (allocated but not mapped, swap-out to disk);

DRAM is software-managed cache for disk, cache block is pages, fully associative.

Page fault (swapped-out page): a type of exception, V=0, invoke OS page fault handler to move data from storage to memory. Like cache miss. If unallocated (non-existing), terminate process, extremely slow



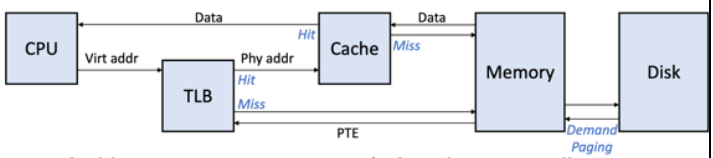

TLB: Cache for address translation

Locality in access -> locality in translation.

TLB = valid + tag + PTE, tag is the virtual page number.

Multi-level page tables: top level resident in memory, remaining levels in memory / disk, or unallocated. Save sparse virtual address significantly.

Inter-Process Sharing: different process share page by setting virtual addr to same physical addr




Copy on Write: when copying, make destination page write-protected, when writing to either virtual address, trigger a permission fault and copy actually.

Memory Mapping: associate a file with a virtual memory area.

1. Get initial data from file into memory: regular file on disk 2. Swap physical memory content to file: swap file 3. New allocated space initialized to zero: anonymous file

When doing memory mapping, OS only sets up PTE in page table(V=0), wait until first page fault, rely on demanding paging to load data. For anonymous file, physical page is only allocated when first write. (return 0 when read)

`Fork`: child create copies of the page tables and other kernel data structures. Mark each PTE in parent and child as write-protected, copy-on-write.

`Mmap`: access files as well as allocating new virtual address ranges `Execve`: free old page and initialize new page table, memory map program and data, set PC to entry point of the program

External Fragmentation: enough aggregate free space, but not contiguous. Solution: segregated free lists.

Internal Fragmentation: request size is smaller than the allocated free space. Solution: append remaining space to its corresponding free list.

Implicit Memory Management: need garbage collection (pointer is a pointer); mark and sweep (delay memory reclamation), reference counting (simple but may leak memory)

Lecture 9: I/O Devices

Challenges: thousands of devices, slow and unpredictable behavior. Solution: new and standard interface and mechanism, new access approaches.

Operational parameters for IO: data granularity (byte or block), access pattern (sequential or random), device speed rates

Memory-mapped IO: each IO device has interface registers and data buffers, assign an unused range of physical address to each IO device. (not in general DRAM, continuous in physical address space).

IO address space protected by the VM, use syscall (`read, write`) to access IO address or ask OS to map IO address to virtual address (`mmap`).

Standard IO interface: provide uniform interface, abstract IO device to fit standard types by device driver

IO types: blocking (wait until finish), non-blocking (best effort, return quickly with count of transferred bytes), asynchronous (notify when finish)

Block device (SSD): access blocks, can randomly address a location (open, close, read, write, seek)

Stream device(keyboard): no addressing, just read/write next

IO Notification: pooling, periodic checking, hard to determine pool frequency

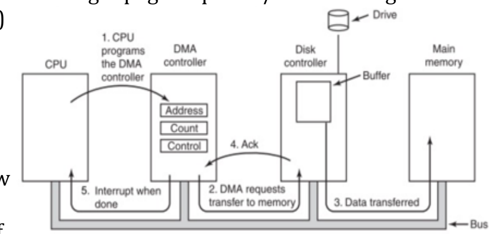
interrupt, device notify process when needs attention, avoid waste CPU time but need context switch

Direct Memory Access (DMA): a hardware for data movement, transfer data between device and memory without CPU intervention. DMA engine itself is IO device, accept data transfer descriptor (src, dest, length), keep multiple in a queue

Processor set up DMA -> DMA transfer -> notify processor using interrupt

DMA design issues: 1. OS may swap pages out to disk, use memory pinning

2. contiguous VM not physically contiguous, chain series of single-page requests / let DMA engine use virtual address (transfer descriptor need continuous)




3. Data involved in DMA transfer may be cached, sol: (1) software flushes cache / forces writeback before IO (2) search cache for copies (may impact performance negatively since when searching cache, processor can't access cache)

Disk: Long-term, non-volatile, large, inexpensive, slow usage: Virtual memory and file system

Disk -> platter -> surface -> track -> sector (groups of sectors form block) block in disk = page in SSD (4kB)

Disk access time = controller overhead + seek time + rotational delay (half cycle) + transfer time (sector pass under heads)(+queuing delay), small data dominated by seek time and rotational delay

Minimum time = controller overhead + transfer time (no seek and rotate)

Disk scheduling: use data locality, speed up large sequential access, can also use cache

FIFO, SSTF (shortest seek time first), SCAN (travel in direction), C-SCAN (travel in one direction)

SSD: made by NAND/NOR multi-level cell Flash; no moving parts, 4kB per page, 32 to 128 pages per block. Read in unit of pages, only write to erased pages, erase happen in units of blocks.

Flash Translation Layer (FTL): SSD must write data to new page in erased block then let address point to this page, FTL manage this process.

Main data structure: address mapping (logical to physical), block info table (track block status, write pages in block sequentially), wear leveling (block wear out after enough erase, avoid write same block)

GPU: as IO device to CPU, offload computation, separate memory; overlap computation and communication. CPU & GPU as management processor + accelerator / cooperative co-processors

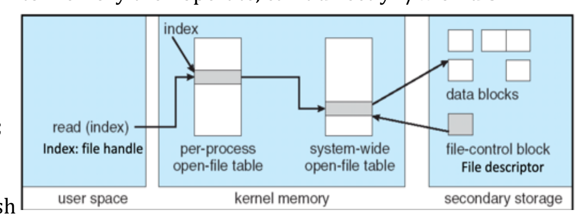
Lecture 10: File System

logical block address (LBA): from 0 to max-number of blocks, skip bad blocks

File system transform block interface into file interface, consists of files and directory structure.

User view: file; Syscall view: contiguous bytes; OS view: collection of blocks, data in system are always accessed and allocated in blocks; read block into memory then operate, can't directly r/w on disk

Named file independent of process, user, system




FS components: file and directory structure

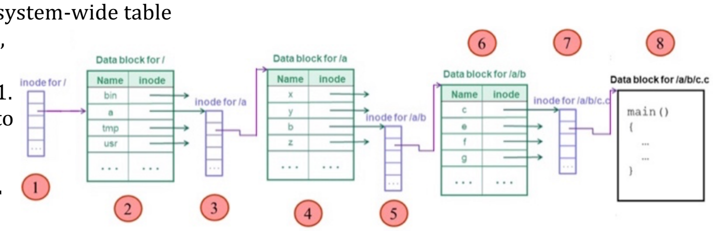
1. Disk management: track which blocks contain data for which file, track free blocks; 2. Naming: find file by name; 3. Protection: keep different user's data isolated; 4. Reliability: keep data consistent despite crash

File descriptor (inode): stored in disk, kept in kernel memory when file is open

Open: translate file name to file number (use dir) -> locate inode -> copy inode to kernel memory -> return file handle (integer, each process has own handle) -> operate on file handle

Two level of open-file tables: per-process and system-wide. Read per-process open-file table using index (file handle), follow pointer to system-wide table

File access patterns: sequential, random, keyed




Data block organization ways: 1. contiguous; 2. linked list: easy to allocate and no external fragmentation, but more disk seeks (Windows FAT, each FAT entry stores address of next

block); 3. indexed file: allocate a specific space to store pointers to data blocks (4.3 BSD Unix Multi-Level index: 12 direct blocks, 1 indirect block, 1 double indirect block, 1 triple indirect block)
Free Space Management: bitmap, search for block close to previously last block
Directory: map name to i-number (the index of inode on disk), just like regular files. Data of directory is an unordered list of <name, pointer> pairs (i-number = pointer to inode); root i-number 2, no name
Directory Structure: inode -> data block <name, i-number> -> inode
Hard link: set another directory entry to i-number for a file, add a reference counter
Soft link: special file, content is another name, stored as regular files with a flag set in file descriptor
Inode of current working directory kept in PCB
Buffer Cache: retain recently accessed disk block; implement in OS software; cache inodes, data blocks and files, indirect blocks, free bitmaps; write-through (directly to disk when writing)
Unify buffer cache and VM page pool to avoid double caching
Synchronous write: immediately write through to disk, slow
Delayed write: not immediately write to disk, fast but dangerous (may lose data when crash)
Protection: Authentication (who are you, password + one-way hash + random salt), Authorization (what can you do), Access enforcement (combine)
Authorization: which principal can do what operation on which object
Access Control List (ACL): with each object, a list of <user, permission> pairs (Owner, Group, Anyone) * (RWX), hard to store if not grouped
Capability: with each principal, a list of <object, permission> pairs, 不可伪造, 可撤销, e.g. page table for physical memory frame
Access Enforcement: enforce access controls and protect info, full power, small and simple, e.g. security kernel

## Lecture 11: Network

Multiplexing: use interior nodes as switches
Packet switching: break data into packets, each packet travels independently to dest (store and forward)
Network card (NIC): hardware that connects computer to network, MAC address (48-bit unique identifier), IP address (32-bit, dynamic)    Connection: communication channel between two processes, endpoint identified by IP + port number (16-bit).
Addressing LAN: ARP (broadcast search for IP, dest reply with MAC)
Hubs: All host in a LAN share same wire, hubs forward from one wire to all others, drop if not for this host. If congestion on wire, detect and retransmit
Switches: forward frames only to correct wire, remember MAC address connect to which port
WAN: connect LANs by routers.    Router: forward packets received from incoming link to outgoing link, based on IP address. Store and forward, use forwarding table (map between IP and output link)
Human-friendly Naming: DNS
Parameter for communication channel: Latency, Capacity, Jitter (variation in latency), Loss, reordering
Problem: Lost (timeout and retransmit), Out of order (sequence number), Corrupted (checksum)
Overload: buffering and congestion control; buffer overflow -> packet dropped -> adjust rate
Avoid overflowing buffer: ACK, if don't receive ACK after timeout, retransmit
Packet delay: propagation delay (distance/speed), transmission delay (packet size/bandwidth), queuing delay (buffering), processing delay (speed of router);
Cut through: forward packet as soon as header is received, xmit delay only appear once
TCP: start small, increase fast until overflow, then half, increase slowly until overflow again (AIMD)
Application – Intermediate layers (set of abstractions) – Transmission media
OSI Layer Model: Application, Transport, Network, Data Link, Physical; Property: service (what a layer does), service interface (how to access), protocol (how to communicate, include syntax and semantics)
1. Physical: move info between system connected by link, coding scheme representing bit
2. Datalink: enable exchange frames, Frame header: Mac src and dest addr, address using ARP
3. Network: deliver packets to specific IP addr; protocol: define global unique network address, forward tables, Net header: IP src and dest addr;    IP: try best effort to deliver
4. Transport: end-to-end communication between processes, multiple process on same hosts communicating simultaneously; protocol: port number; Optional: Reliability, Timing, Rate adaption
TCP: reliable, flow and congestion control, only for two endpoints, retransmit
UDP: extension of best-effort IP, fast but may incomplete, no retransmit, lost data can't be retrieved
7. Application: any service (http, smtp, ...)
Transport: Frame Hdr + Net Hdr + Trans Hdr + data;    Host: 5 layers, Router: 3 layers
Only one Network layer protocol: IP, application can run on IP can use any network
Drawback of layer: hurt perf, header big, layer may duplicate function, layer may need same info
Don't implement end-to-end function in communication system

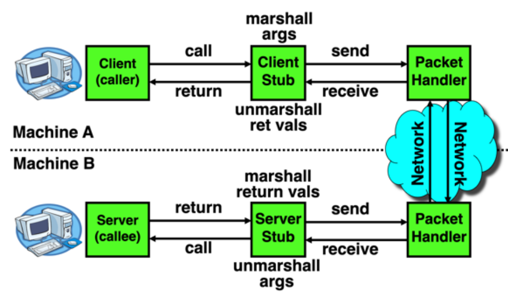
## Lecture 12: Transaction

Data Model: collection of entities and their relationships;
Schema: an instance of data model; Relational data model: relation (table), every relation has a schema which describes the fields in the columns
Database Management System (DBMS): a software system designed to store, manage and facilitate access to DB, provide Data Definition Language (DDL, relation, schema) and Data Manipulation Language (DML, queries)
Key Concepts: Queries, Query plans and operators
Consistency based on our knowledge of semantics, DBMS provide automatic enforcement via IC
Transaction: atomic sequence, take DB from one consistent state to another
ACID properties: Atomicity, Consistency, Isolation, Durability (Journaling file system)
Log: write/append a basic item is atomic, use that solidify commitment to a series of action
Creating a file: find, write in log, commit log, copy changes to disk
Crash during log: scan log, detect transaction without commit, discard, redo
Crash after commit: find matching commit, redo as usual
Locks: higher-level construct than in OS, a reader-writer monitor
Transaction Schedule: max concurrency, semantically equivalent to some serial schedule; Equivalent schedule: the effect and output of two schedules are the same; Serializable schedule: equivalent to some serial schedule    conflict: belong to different transactions, access same data item, at least one is write; conflict equivalent: every conflicting operation is ordered the same way; conflict serializable: conflict equivalent to some serial schedule -> serializable (iff dependency graph is acyclic)
Lock: shared(S) lock, exclusive lock(X); S or X before reading, X before writing
2-Phase locking(2PL), once it releases any locks, it can't request any locks, guarantee conflict serializable (otherwise deadlock)

| Method | Comment |
|---|---|
| UNIX semantics | Every operation on a file is instantly visible to all processes |
| Session semantics | No changes are visible to other processes until the file is closed |
| Immutable files | No updates are possible; simplifies sharing and replication |
| Transactions | All changes occur atomically |

## Lecture 13: Distributed system

Definition: collection of independent computers that appears to its user as a single coherent system
Message sharing, no shared memory
Goal of distributed system: Resource sharing, transparency, openness, scalability
Inter-process communication: fork, signal, file, pipes, shared memory, socket(network)
Remote Procedure Call: look like a local procedure call on client `file.read(1024)`, translate into a procedure call on remote machine




Marshall: convert values to a canonical form, serialize objects, copy arguments passed by reference. Use stub to glue piece together
Call Semantics: exactly / at most / at least once
Cost of Procedure call << same-machine RPC << network RPC
Simple DFS: every read and write gets forward to server, performance low
NFS: server stateless, each request provides all arguments for execution, performing requests multiple times equals to exactly once (Idempotent)
NFS Pro: simple, highly portable; Con: sometimes inconsistent, don't scale to large number of clients
Cache consistency: 1. Write-through caching: modified data committed to server's disk before return
2. Client pool periodically (30s) to check for change
AFS: assume client machine has disk, most update by one user / machine, access local is faster, cheaper
Cell / Volume Arch: cell – administrative groups (afs/andrew.cmu.edu), broke into volumes (mini file system)
AFS caching: cache whole file on disk when open, register with server that they have a copy of file; server tell "invalidate" if file changes
AFS: Session semantics, write visible to all process in same machine; when file closed, changes visible to new opens, but not visible to old opens.
Ideal: One Copy Semantics