Synchronisation Properties
Mutual Exclusion: if Pi in CS, all processes cannot enter CS
Progress: if none in CS, one waiting enters CS
Bounded Wait: upper bound on waiting time
Independence: Process not in CS should not block other processes

Incorrect Synchronisation
Deadlock: all processes blocked
Livelock: processes running, but changing state to avoid deadline, doing useless
Starvation: wait forever

TestAnd Set Reg, MemLocat Atomically:
Load [MemLoc] into [Reg]
[MemLoc] <- 1

Peterson's Algo
- Busy Waiting
- Need variable setting
- Only 2 processes
- Turn=Want[0]=Want[1]=0

```
Want[0] = 1;                    Want[1] = 1;
Turn = 1;                       Turn = 0;
while (Want[1] &&               while (Want[0] &&
       Turn == 1);                     Turn == 0);
Critical Section                Critical Section
Want[0] = 0;                    Want[1] = 0;
Process P0                      Process P1
```

Semaphore
sem_t a;
sem_init(&a, process=1, val=1)
(process=1, thread=0)
sem_wait(&a); (--, else block)
sem_post(&a); (++/signal)
Scurr = Sinit + #signal - #wait

Producer Consumer
Producer produces if buffer <K
Consumers eat if >0
count=in=out=0
mutex=S(1), notFull=S(K)
notEmpty=S(0)

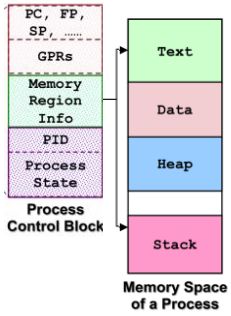
Producer	Consumer
wait(notFull)	wait(notEmpty)
wait(mutex)	wait(mutex)
CS	CS
signal(mutex)	signal(mutex)
signal(notEmpty)	signal(notFull)

Reader Writer
Reader must have X lock
Writer must have S lock
all mutexes = S(1)

Reader-preference
```
writer()
wait(roomEmpty);
Modifies data
signal(roomEmpty);
reader()
wait(mutex);
nReader++;
if (nReader == 1)
  wait(roomEmpty);
signal(mutex);
Reads data
wait(mutex);
nReader--;
if (nReader == 0)
  signal(roomEmpty);
signal(mutex);
```

Writer-preference
```
writer()
wait(writeMut);
writeCount++
if(nWriter == 1)
  wait(readTry);
signal(writeMut);
wait(roomEmpty);
Modifies data
signal(roomEmpty);
wait(writeMut);
nWriter--;
if(nWriter==0)
  signal(readTry);
signal(writeMut);
reader()
wait(readTry);
wait(readMut);
nReader++;
if(nReader == 1)
  wait(readMut);
signal(readMut);
signal(readTry);
Reads data
wait(readMut);
nReader--;
if(nReader == 0)
  signal(roomEmpty);
signal(readMut);
```

Pthread Mutex
pthread_mutex
pthread_mutex_lock()
pthread_mutex_unlock()
pthread_cond
pthread_cond_wait()
pthread_cond_signal()
pthread_cond_broadcast()



Memory Usage
Transient Data eg params, local vars
Persistent Data eg global vars

OS Handles:
- Allocate memory space to new processes
- Manage memory space for processes
- Protect memory space of processes from each other
- Provides memory related system calls to processes
- Manage memory space for internal use

Main Topics:

Memory Abstraction
Presenting a logical interface for memory accesses

Contiguous Memory Allocation
Allocating and managing continuous chunk of memory

Disjoint Memory Allocation
Allocating and managing memory in disjoint areas

Virtual Memory Management
Use of secondary storage as an extended memory region

Memory Abstraction
- Cannot fix addr during comp time
- Need to protect address space when 2 processes have same phys mem

Address Relocation (not used)
- diff to distinguish mem ref from int ref

Base + Limit Registers (not used)
- Base reg added to all refs
- Limit reg needed for mem integrity
- Have to add and compare every inst

Multitasking
- Allow multiple processes at same time to allow switching
- Free up memory by removing terminated processes, swap blocked processes to sec storage

Contiguous Mem Mgmt
- Contiguous process space, phys mem can store whole memory

Fixed-Size Partitions
- Physical memory is split into fixed number of partitions of equal size
- A process will occupy one of the partitions
- Internal Fragmentation

Variable-Size Partitions
- Partition is created based on the actual size of process
- OS keep track of the occupied and free memory regions
- Perform splitting and merging when necessary (no internal partition)
- Need to store more info
- Need to locate regions

Dyn Algorithms: FF, BF, WF, NF

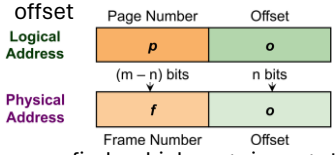
Buddy Algorithm
- Free blocks split in half to find smallest block of 2^S >= N
- Buddies have 0..S-1 bits equal
- kth level buddy has kth bit flipped
- 32B=2^5 root is 00000, 10000 (16B buddies)

FF, BF, WF, NF: EF, no IF
Fixed: IF, no EF
Buddy: IF and EF

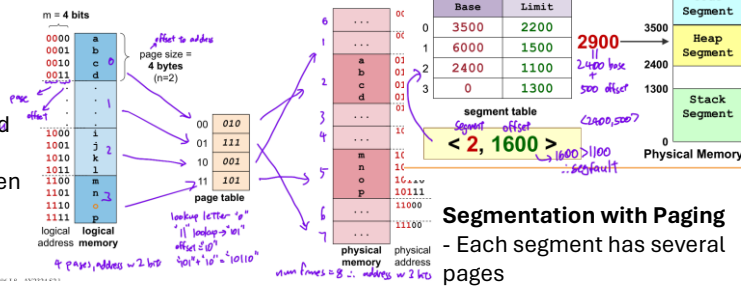
Disjoint Mem with Paging
- Logical Memory split into logical pages of same size
- Physical memory split into physical frames of same size
- During runtime, pages are loaded into available memory frames

Paging Scheme
- needs to know starting address and size of process
- EF not possible. IF insignificant

Phys frame size== logical page size==2^k
PA=F*sizeof(physical_frame)+ offset



p finds which page in page table
f is what's inside page table
o is num bits of a page (1page is 2^oB)
p ++ o is num bits of LogMem
f ++ o is num bits of PhysMem



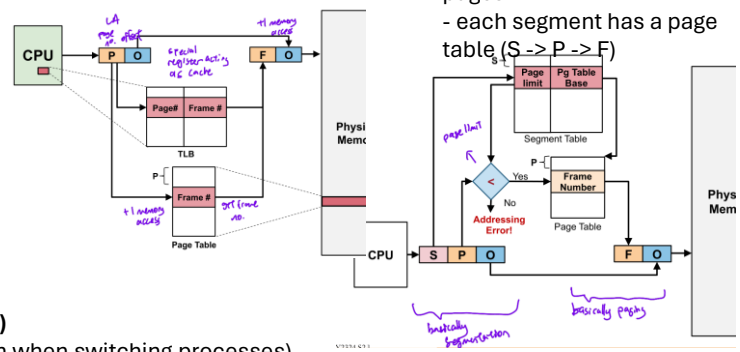
Paging Scheme Protection
Access-Right Bits – RWX. Every memory access checks (not every process uses all the pages)
Valid Bit – whether is valid to access by process

Page Sharing
Several processes can share the same physical memory frame

Copy-on-Write

Segmentation
Some regions may grow / shrink at execution time
Each memory segment is specified with **SegmentName** + **Offset**, and specifies a limit
LA <**SegID**, **Offset**>:
SegID looks up <**Base**, **Limit**> in segment table
PA = **Base** + **Offset**
Offset < **limit**

Segmentation with Paging
- Each segment has several pages
- each segment has a page table (S -> P -> F)



Translation Lookaside Buffer (TLB)
Caches for Page Table Entries (flush when switching processes)

Average memory access time (AMAT):
= TLB hit + TLB miss = 90% x (1ns + 50ns) + 10% x (1ns + 50ns + 50ns)
= P(TLB hit) x latency(TLB hit) + P(TLB miss) x latency(TLB miss) = 56ns

Virtual Memory
- If processes Logical mem > phys mem
- Have a section of memory to map to phys mem to map to sec storage

Extended Paging Scheme
Memory Resident (pages in phys mem
Non-Mem Resident (pages in sec storage)
CPU accesses only MemRes pages, else page fault (OS loads page into phys mem)

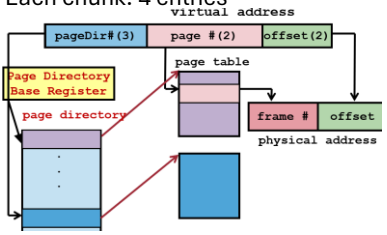
Locality: Temporal, Spatial
Cache thrashing

Demand Paging
- fast startup time, but many page fault
Page Replacement Algo
Frame Allocation

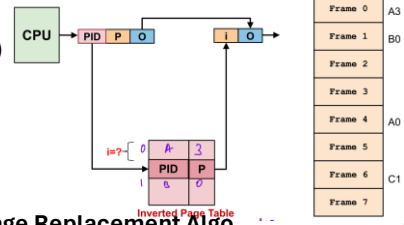
Direct Paging
Keep all entries in single table
- 2^p pages in logical memory space
- p bits to specify one unique page
- 2^p page table entries (PTE) with phys frame number + info bits (valid, read only)

Example:
- Page size: 4KiB (12 bits for offset)
- VA 64-bit -> 16 ExaBytes of VA ($2^4 * 2^{60}$)
- PhysMem16GiB -> PA 34 bits
- Virtual Pages: $2^{64}/2^{12} = 2^{52}$
- 2^{52} PTE entries
- Phys pages? $2^{34}/2^{12} = 2^{22}$
- PTE size = 8B (with other flags)
- Page table size = $2^{52} * 8B = 2^{55}B$ per process!
- physical memory size $2^{34}B$!

2-Level Paging
Split page table into chunks. E.g.
VA: 7 bits
Page dir: 2^3 entries
Page size: 4 bytes (2 bit offset), so 4 entries
#page: $2^{7-2} = 32$ pages
Each chunk: 4 entries



Inverted Page Table
- quickly find process of evicted frame to mark page table as nonresident
- physical frame to <**pid**, **page#**>
- pid = process id , page# = logical page number in the corresponding the process
- page# is not unique among processes
- pid + page# can uniquely identify a memory page
- In normal page table, the entries are ordered by page number
- To lookup page X, simply access the Xth entry
- In inverted page table, the entries are ordered by frame number
- To lookup page X, need to search the whole table



Page Replacement Algo
$T_{access} = (1-p) * T_{mem} + p * T_{page\_fault}$

Belady's Anomaly
More frames, page faults increases
FIFO does not have temporal anomaly
LRU does not suffer, but needs Counter or Stack (complicated)

Clock
- Equivalent to FIFO when all refbit=1

while True:
  if refbit = 0, evict. Done
  if rebit =1, set to 0 and advance

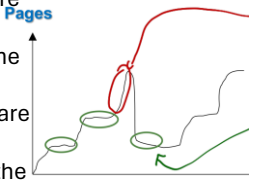
Frame Allocation
- Equal allocation
- Proportional Allocation: total available / ratio mem of process

Page Replacement
Local Replacement (evict within proc)
- Stable performance between runs
- Page thrashing if insufficient
- Thrashing might use I/O bandwidth, degrade other process' perf

Global replacement (evict any)
- Allow adjust between processes
- Page stealing / cascading thrashing / unpredictable performance

Working Set
- set of pages referenced by a process is relatively constant in a period of time e.g. local vars, params. code
- Transient, Stable Regions

Working Set Model
- W(t, Δ) = active pages in the interval t
- Allocate enough frames for pages in W(t, Δ) to reduce possibility of page fault



File Storage
- Self-contained
- Persistent
- Efficient

Metadata
Name, ID, Type, Size, rwx, Time, Date,Owner

Access Control List
getfacl()

File Data Structure
- Array
- Fixed Length
- Variable Length

Access Methods
- Sequential
- Random
- Direct (fixed only)

Details
File pointer
File descriptor (id)
Disk loc
Ref count

File Block Allocation
Contiguous
- Fast access
- External Fragmentation
- Know file size in advance

Linked List
- Disk Blocks store data+next disk
- No EF
- Part of disk block lost for ptr
- Low random access

File Allocation Table (FAT)
FAT16 -> 2^16 disk blocks
FAT entry contains either:
- FREE code (block is unused)
Block number of next block
- EOF code (i.e., NULL pointer)
- BAD block (block is unusable, i.e., disk error)

Details
- Faster random access (RAM traversal)
- But need all disk blocks in partition

Partition Free Space
(how to tell disk block is full)
- Free space list

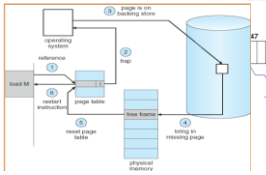
Bitmap (need in mem)
Linked List (high overhead)

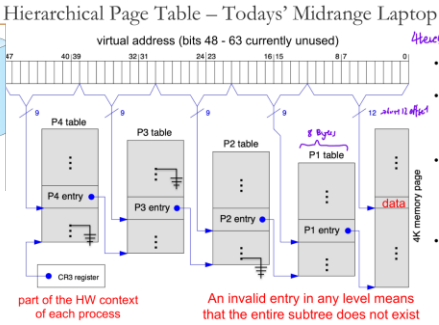
Directory Impl
- Hash table / Linear List

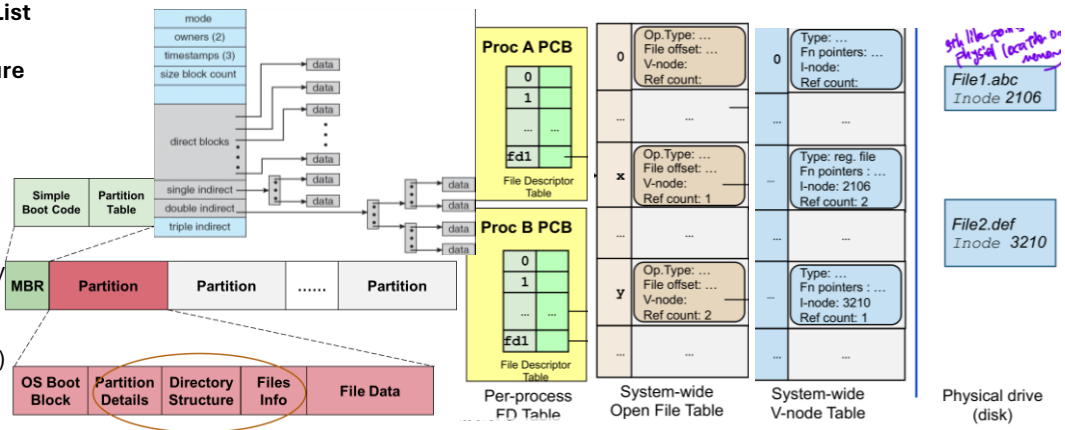
File Info
- File name, metadata, disk blk info
- either store in dir entry, or point to another ADT

Indexed Allocation (EXT2)
Each file has index blk, IndexBlk[N] == Nth Blk Addr
- Less overhead
- Fast direct Access
- Limited max file size by blks

Direct blocks (12x of 1KiB):
Single Indirect block (1x):
#entry in disk block = $2^{10}/4$ (blkAddr) = 256 entries
Double Indirect block (1x): 256^2 x 1 KiB = 64 MiB
Triple Indirect block: 256^3 x 1 KiB = 16 GiB



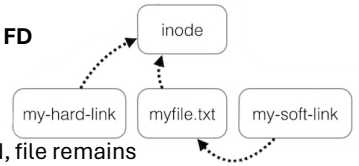




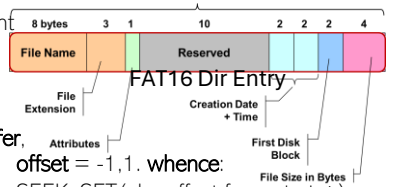
Fd in Per-process FD
Separate fd
Fork: copy fd

Soft Link
- If symlink deleted, file remains
- Dangling links possible

Hard Link (Disallowed for dir)
- separate pointers to actual file (only pointers added)
- creates DAG




```
int rfd = open(char* f1, int O_RDONLY);
int wfd = open(f2, O_WRONLY|O_CREAT, 0600)
int bytesRead = read(rfd, void*buffer, int bytesToRead=1) (returns >=0)
int bytesWritten = write(wfd, buffer, int BytesToWrite) (returns >= -1)
off_t seeked = lseek(fd, off_t offset, int whence) (returns >= -1).
```



offset = -1,1. whence:
SEEK_SET(abs offset from start +),
SEEK_CUR (relative offset from curr +/-)
SEEK_END (relative from end +/-)
int closed = close(int fd)