sbrk() 和 brk()的区别：
brk通过传递的addr来重新设置program break，成功则返回0，否则返回-1。而sbrk用来增加heap，增加的大小通过参数increment决定，返回增加大小前的heap的program break，如果increment为0则返回program break。

Week 1
1. Process: the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity.

2. Program: a group of instructions to carry out a specified task; Process: a program in execution
程序是指令的集合，是进程运行的静态描述文本。而进程则是程序在系统上顺序执行时的动态活动。

3. Yes

4. Process will run directly on the cpu. Limited: some limits to what a process can and can not do

5. Restricted operations

6.
Kernel mode: the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address.

User mode: the executing code has no ability to directly access hardware or reference memory.

7. Context switch: the process of storing the state of a process or of a thread.
允许多个进程分享单一CPU资源的计算过程

8. Cooperative multitasking: known as non-preemptive multitasking, a style of computer multiple tasking, the os never initiates a context switch from a running process to another process.
协作式多工，每一个运行中的程序，定时放弃自己的执行权力，告知操作系统可让下一个程序执行。

9. Caused busy waiting and extensive calculations. Make environment unacceptably fragile.

10. True multitasking: the capacity of an OS to carry out two or more tasks simultaneously rather than switching from one task to another. Each core could be performing a separate task at any given time.
Difference with Cooperative multitasking: True multitasking could let each task run separately, but CM has to work with others by sharing rss.

11. PCB: process control block, a data structure in the OS kernel containing the info needed to manage the scheduling of a particular process

12. Process state, Program Counter, Stack Pointer, Status of opened files, Scheduling algorithms

13. Process ID (PID) uniquely identify an active process

14. Von-Neumann Model: Control Unit, Arithmetic, Logical Memory Unit(ALU), Registers and Inputs/Outputs.
Six fundamental phases of the instruction cycles:

- Fetch instruction (pre-fetch)
- Decode instruction
- Evaluate address (address generation)
- Fetch operands (read memory data)
- Execute (ALU access)
- Store result (writeback memory data)

15. A hardware feature allows a single processor to act as if it was multiple individual CPUs. It allows os to more effectively & efficiently utilize the CPU power in the computer to run faster.

16. Run in virtual memory allocated by OS. Like Linux, each process has its own address space and uses virtual memory. Even same address usually refers to different physical memory cells in two processes

17.
Thread is “Lightweight process”, an execution stream that shares an address space
//A thread of executions is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the OS.
Short for thread of execution

18.
Virtualizing the CPU:
Given each process the impression it alone is actively using CPU. Resources can be shared in time and space

19. The memory that a program or process can access. The memory could be either physical or virtual and is used for executing instructions and storing data.

20.
Running: On the CPU (only one on a uniprocessor)
Ready: Waiting for the CPU

Blocked: Asleep: Waiting for I/O or synchronization to complete
//READY: the process is waiting to be assigned to a processor
//RUNNING: Instructions are being executed
//WAITING: the process is waiting for some event to occur
//TERMINATED: the process has finished execution

21.

22.

23. Depends on the prog design

24. Wait

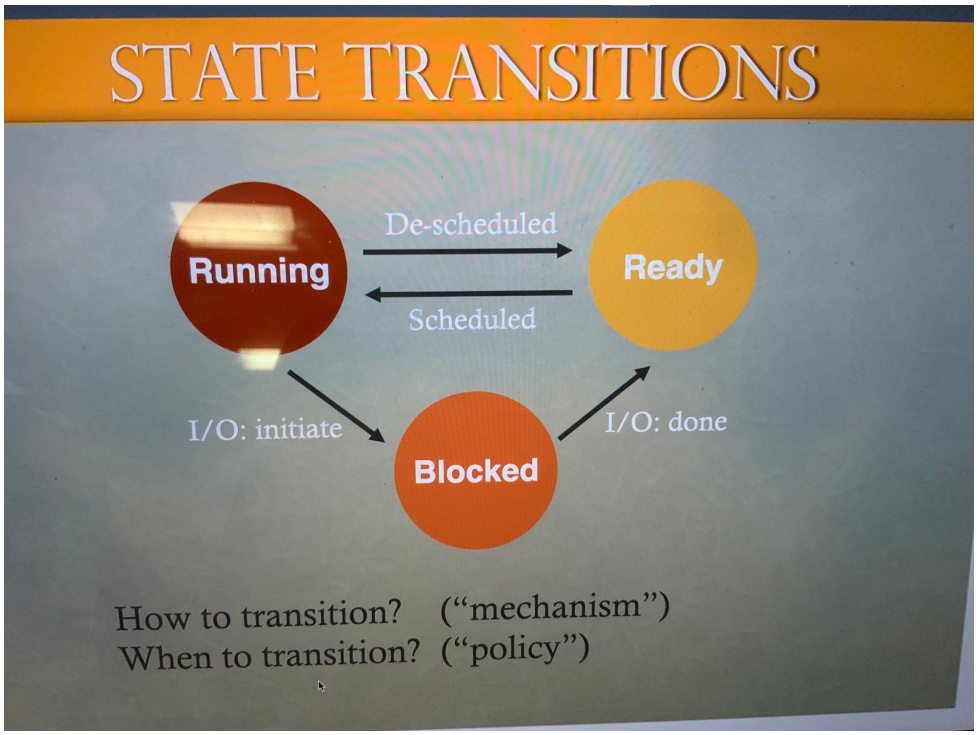
25. All code and data in the process is lost and replaced with the executable of the new program

26. OS instruction?

Week 2

1. fork(), wait()

2.



3. Workload: the amount of processing that computers have been given to do at a given time. It consists of some amount of application programming running in the computer and usually some number of users connected to and interacting with the computer's applications.

Workload: set of job descriptions

4. Scheduler: determine how to move processes between the ready and run quests which can only have one entry per processor core on the system. A special system software which handle process scheduling in various. It can select the jobs to be submitted into the system and to decide which process to run

Scheduler: logic that decides which ready job to run

5.
Minimize turnaround time
Minimize response time
Minimize waiting time
Maximize throughput
Maximize resource utilization
MInimize overhead
Maximize fairness

6. FIFO: First in, First out (aka FCFS, first come first served)
Simply queues processes in the order that they arrive in the ready queue.

7. SJF (Shortest job first)
Choose job with smallest run_time

8. STCF (Shortest Time-to-Completion First)
Always run job that will complete the quickest

9. RR (Round Robin scheduler)
Alternate ready processes every fixed-length time-slice

10. Problems for FIFO:
Turnaround time can suffer when short jobs must wait for long jobs

11. True

12. Preemptive Scheduler:
Potentially schedule different job at any point by taking CPU away from running job

The strategy of allowing processes that are logically run able to be temporarily suspended.
Contrast to “run to completion”

FIFO and SJF are non-preemptive, STCP is preemptive
Preemptive: 抢先

13.
Response time = first_run_time - arrival_time

14.
“Interactive” programs care about response time
“Batch” programs care about turnaround time

15.
MLFQ: Multi-Level Feedback Queue
General-purpose scheduling
How does it work???

```
response_time
=
first_run_time
-
arrival_time
```

16.  Yes???

17. It will starve out the CPU intensive jobs

18. Lottery Scheduling
Proportional share
- Approach:
  - Give processes lottery tickets
  - Whoever wins runs
  - Higher priority => More tickets

Week 3

1. Standard segment for Unix user:
Code segment, data segment, BSS (block started by symbol), stack segment

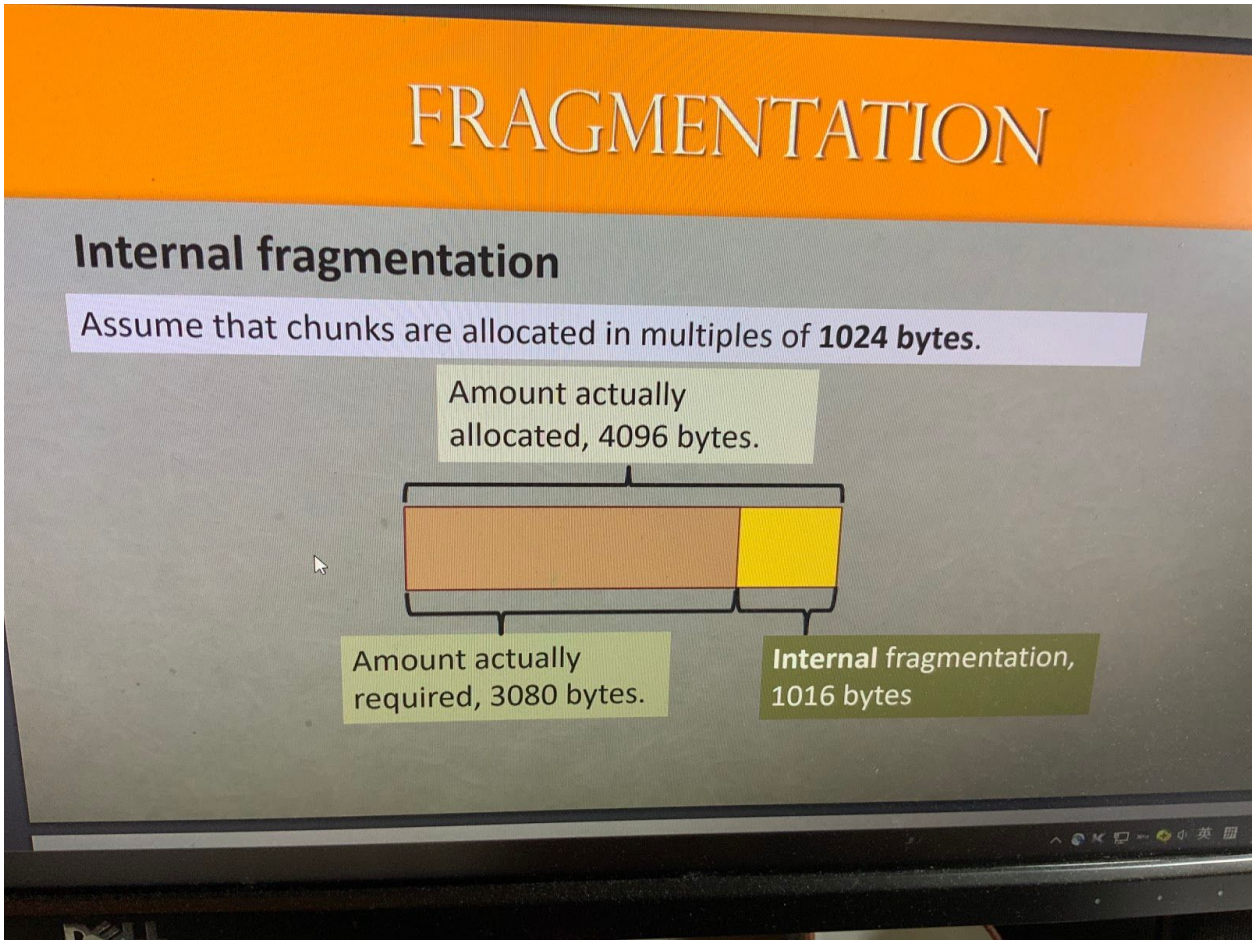
2.
Transparency
- User processes are not aware that memory is shared
- Works regardless of number of and/or location of processes

Protection
- A user process cannot corrupt OS or other processes
- Privacy: Cannot read data of other processes

Efficiency
- Do not waste memory resources (minimize fragmentation)

Sharing
- Cooperating processes can share portions of address space

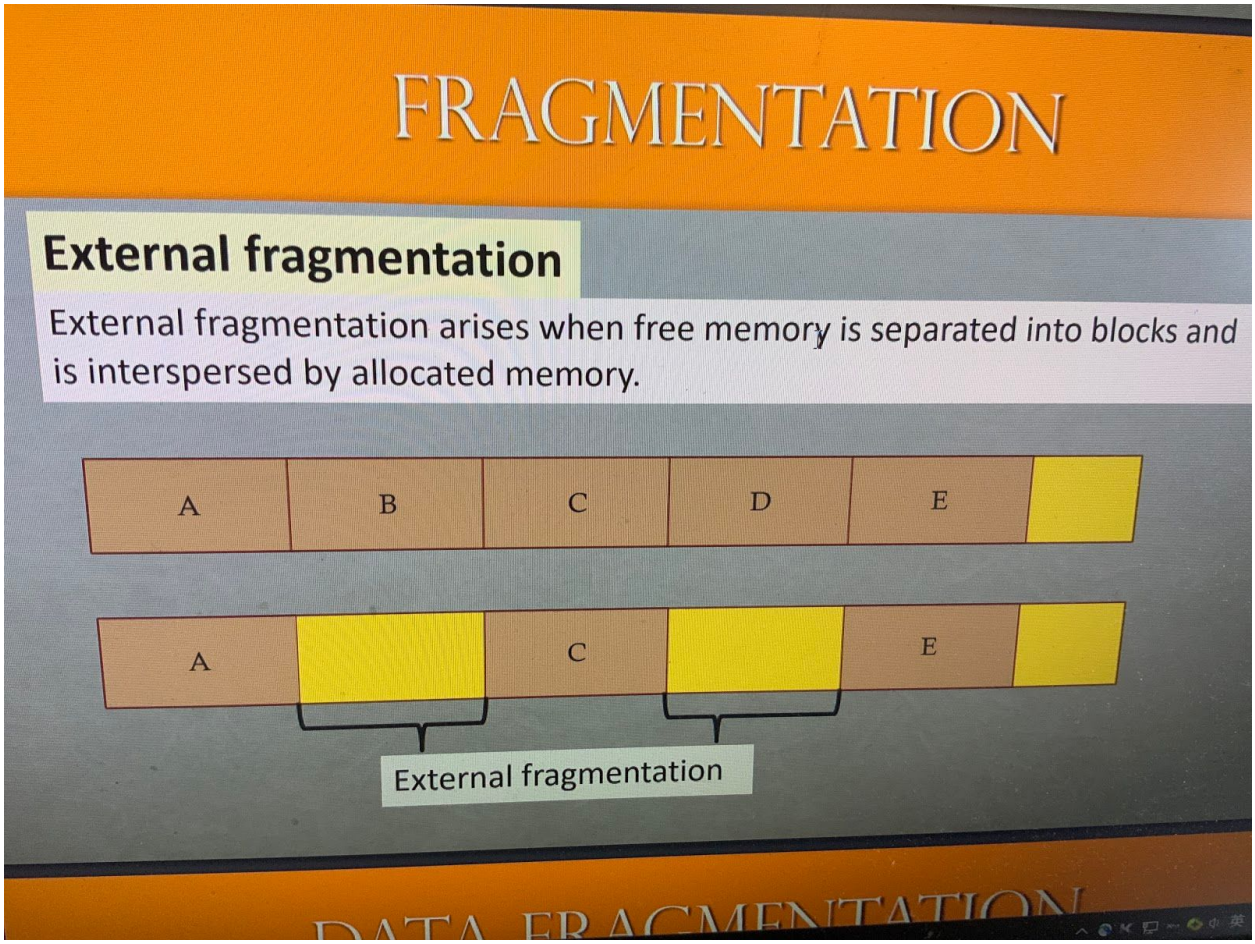
3.
Internal Fragmentation:
- More memory is sometimes allocated than is actually necessary




4.

External Fragmentation:

Free memory is separated into blocks and is interspersed by allocated memory.

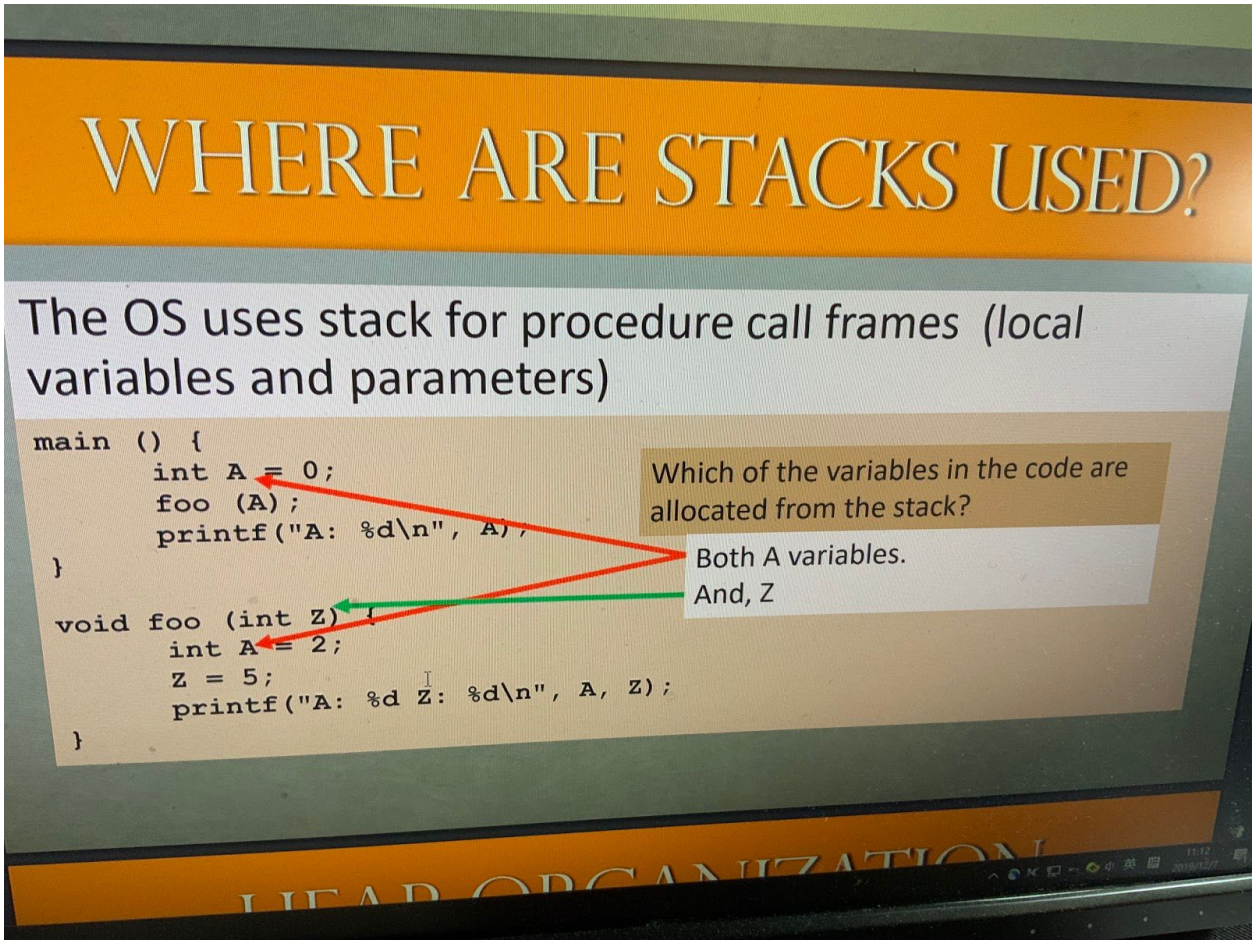



5.
Static segments: code and some global variables
Dynamic segments: stack and heap

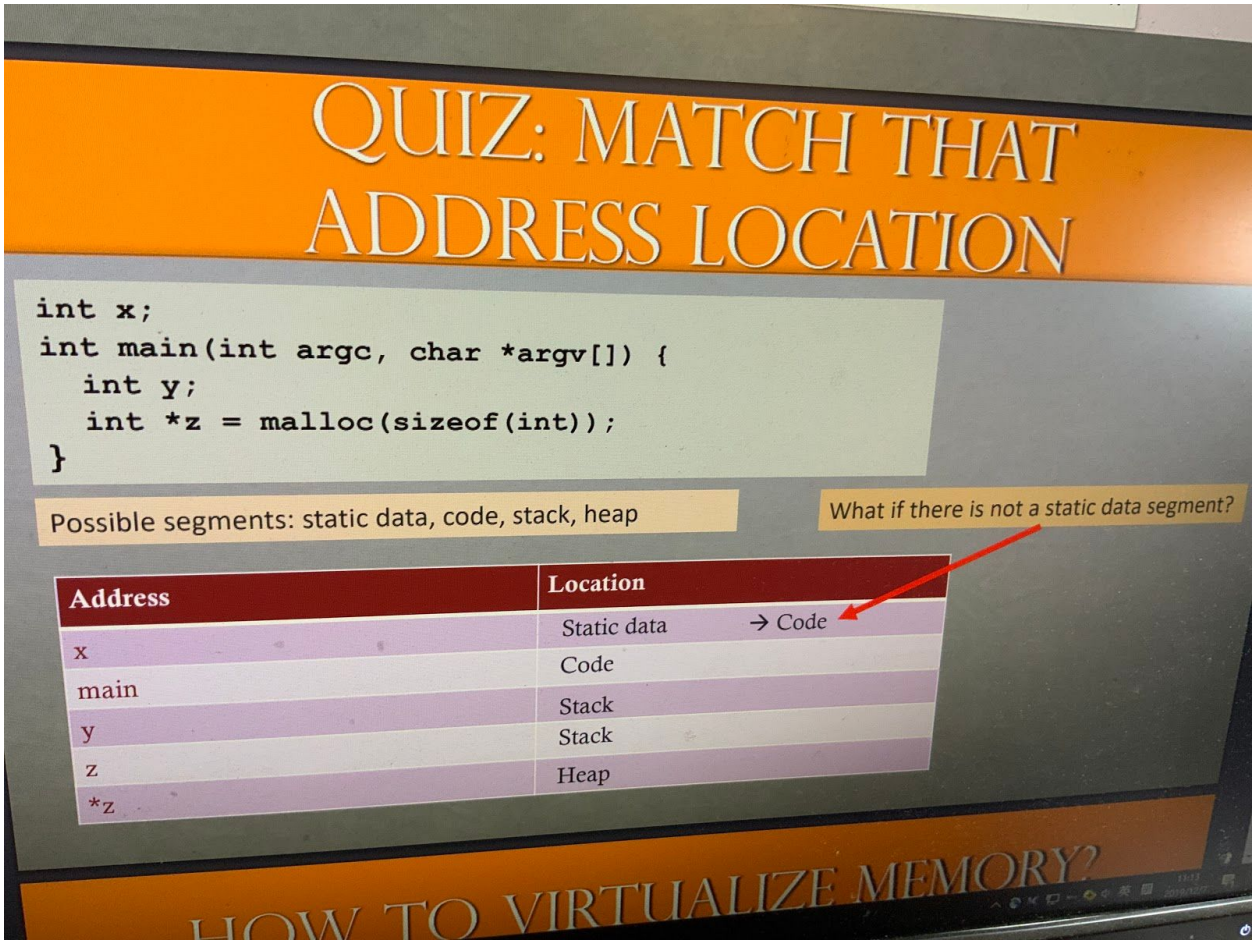
6.
Automatic  variables = local variables

7.
Created on stack on call to a function.
Deallocated on return from function.

8.


WHERE ARE STACKS USED?
The OS uses stack for procedure call frames (local variables and parameters)
main () {
int A = 0;
foo (A);
printf("A: %d\n", A);
}
void foo (int Z) {
int A = 2;
Z = 5;
printf("A: %d Z: %d\n", A, Z);
}
Which of the variables in the code are allocated from the stack?
Both A variables.
And, Z

9.




10.
Poor performance

11.
Static relocation:
- OS rewrites each program before loading it as a process in memory.
- Each rewrite for different process uses different addresses and pointers
- Change jumps, loads of static data

12.
No protection
- Process can destroy OS or other processes
- No privacy

Cannot move address space after it has been placed
- May not be able to allocate new process - fragmentation

13.
Dyn relocation:

Protect processes from one another

Required new hardwares:

Memory Management Unit (MMU)

MMU dynamically changes process address at every memory reference

14.

False. Each process has different value in base register

15.

What entity should do the translation of addresses with base register?

HW

16.

What entity should modify the base register?

OS

17.

Bound register?

18.

Yes. Add base and bounds registers to PCB

19.

Each process must be allocated continuously in physical memory

Must allocate memory that may not be used by process

No partial sharing: can not share limited parts of address space

20.

Process segmentation:?

Segmentation is the process of dividing a heterogeneous market into smaller markets or sub-segments on the basis of some homogeneous characteristics. Segmentation yields segments which are similar in some aspects.

21.

Enables sparse allocation of address space

Stack and heap can grow independently

Heap: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls sbrk())

Stack: OS recognizes reference outside legal segment, extends stack implicitly

Different protection for different segments

Read-only status for code

Enables sharing of selected segments

Supports dynamic relocation of each segment

22.
Each segment must be allocated contiguously
May not have sufficient physical memory for large segments

Week 4

1. Instead of splitting up … , each of which we call a PAGE

2. We'll view physical memory as … called PAGE FRAMES, …

3. Paging over segmentation: speed. Reloading segment registers to change address spaces is much faster than switching page tables. It consume less memory than page tables.

4. High-order bits: address designate page number, translated into a frame number

5. Low-order bits: address designate offset within page

6.

| Page Size | Low Bits (offset) | Virtual Address Bits | High Bits (VPN) |
|---|---|---|---|
| 16 Bytes | 4 | 10 | 6 |
| 1 KB | 10 | 20 | 10 |
| 512 Bytes | 9 | 16 | 7 |
| 1 MB | 20 | 32 | 12 |
| 4 KB | 12 | 32 | 20 |

7.

| Page Size | Low Bits (offset) | Virtual Address Bits | High Bits (VPN) | Virtual Pages |
|---|---|---|---|---|
| 16 Bytes | 4 | 10 | 6 | 64 = 2^6 |

| | | | | |
|---|---|---|---|---|
| 1 KB | 10 | 20 | 10 | 1 K =2^10 |
| 512 Bytes | 9 | 16 | 7 | 32 = 2^5（这里应该是老师写错了）2^7 |
| 1 MB | 20 | 32 | 12 | 4 K = 2^12 |
| 4 KB | 12 | 32 | 20 | 1 M = 2^20 |

8.
Advantage of paging:

No external fragmentation
    Any page can be placed in any frame in physical memory
Fast to allocate and free
    Alloc: No searching for suitable free space
    Free: Doesn't have to coalesce with adjacent free space
    Just use bitmap to show free/allocated page frames
Simple to swap-out portions of memory to disk
    Page size can disk block size
    Can run process when some pages are on disk
    Add "present" bit to PTE

9.
Disadvantage of paging:

Internal fragmentation: Physical page size may not match size needed by process
    Wasted memory grows with larger pages
Additional memory reference to page table --> Very inefficient
    Page table must be stored in memory
    MMU stores only base address of page table
    Solution: Add TLBs (a lecture in your near future)
Storage for page tables may be substantial
    Simple page table: Requires PTE for all pages in address space
        Entry needed even if page not allocated
    Problematic with dynamic stack and heap within address space
    Page tables must be allocated contiguously in memory
    Solution: Combine paging and segmentation (future lecture)

简便版：
Pros and Cons of Paging Table

Advantages
No external fragmentation
    don't need to find contiguous RAM
All free pages are equivalent
    Easy to manage, allocate, and free pages

Disadvantages
Page tables are too big
    Must have one entry for every page of address space
Accessing page tables is too slow [focus for this lecture]
    Doubles number of memory references per instruction

10.
Cheap 1. Extract VPN from VA
Cheap 2. Calculate address of PTR (page table entry)
Expensive 3. Read PTE from memory (Don't always do)
Cheap 4. Extract PFN (page frame num)
Cheap 5. Build PA (Physical address)
Expensive 6. Read contents of PA from memory into register

11.
TLB: Translation lookaside buffer

12.
Yes. Smaller size, higher hit rate

13. Highly random, with no repeat accesses pattern will be slow.
Repeated Random Accesses pattern is slower

14.
Spatial Locality: Future access will be to nearby addresses
Spatial:
    Access same page repeatedly; need same vpn->ppl translation
    Same TLB entry re-used

15.
Temporal Locality: Future access will be retreats to the same data
Temporal:
    Access same address near in the future
    Same TLB entry re-used in near future

16.
Flush TLB on each switch

Very costly; lose all recently cached translations

Track which TLB entries are for which process
Address Space Identifier
Tag each TLB entry with an 8-bit ASID (Address Space ID)

Context switch are expensive

Week 5

1. Threads like processes, except multiple threads of same process share a single address space

2.
False. Others are able to execute

3.
Parallelism: perform many tasks simultaneously; improve throughput

Concurrency: mediates multi-party access to shared resources; decrease response time

4.
Processes have independent memory space
Threads have shared memory space

5.
Atomic:
”All or nothing” - it should either appear as if all of the actions you group together occurred, or that none of them occurred, with no in-between state visible

6.
Critical Region:
Piece of code that accesses a shared resource, usually a variable or data structure

7.
Mutual exclusion
A guarantee that only a single thread ever enters a critical section, thus avoiding race

8.
Race condition:
The results depend on the timing execution of the code

9.

Transaction:
A sequence of steps that occur atomically

10.
Threads share page directories T

11.
Threads not share a single instruction pointer F

12.
Threads not share Stack Pointer F

13.
User-level threads: Many-to-one thread mapping
- Implemented by user-level runtime libraries
  - Create, schedule, synchronize threads at user-level
- OS is not aware of user-level threads
  - OS thinks each process contains only a single thread of control

Adv
- Does not require OS support; Portable
- Can tune scheduling policy to meet application demands
- Lower overhead thread operations since no system call

Disadvantage
- Can not leverage multi processors
- Entire process blocks when one threads blocks

14.
Kernel-level threads: One-to-one thread mapping
- OS provides each user-level thread with kernel thread
- Each kernel thread scheduled independently
- Thread operations (creations, scheduling, synchronization) performed by OS

Adv:
- Each Kernel-level thread can run in parallel on a multiprocessor
- When on thread blocks, other threads from process can be scheduled

Disadvantage:
- Higher overhead for thread operations
- OS must scale well with increasing number of threads

15.
Mutex = Mutual exclusion

16.
2 states:  Acquire && Release

17.
False. Only the thread that owns the lock can unlock it

18.
A thread may not unlock a mutex that is doesn't currently own.
False. It turned undefined behavior

19.
Threads failed to join with it when completes.
If enough zombie threads are created, then you will not be to create more threads and your process will be unstable.

20.
Detached Threads run independently of the thread that created it.
Creating threads will not wait for a detached thread to complete to join back with it.
Detached threads will be cleaned up when they terminate without calling a join on them, they do not turn into Zombies.

Week 6

1.
Spin lock performance:
Fast when:
- Many CPUs
- Locks held a short time
- (adv) avoid context switch

2.
Slow when:
- One CPU
- Locks held a long time
- (disadv) spinning is wasteful

3.
A deadlock occurs when the waiting process is still holding on to another resource that the first needs before it can finish.
EX:
Resource A and resource B are used by process X and process Y
- X starts to use A.
- X and Y try to start using B

Y 'wins' and gets B first
now Y needs to use A
A is locked by X, which is waiting for Y

4.
Deadly embrace = deadlock

5.
Dining Philosophers Problem
The dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him.

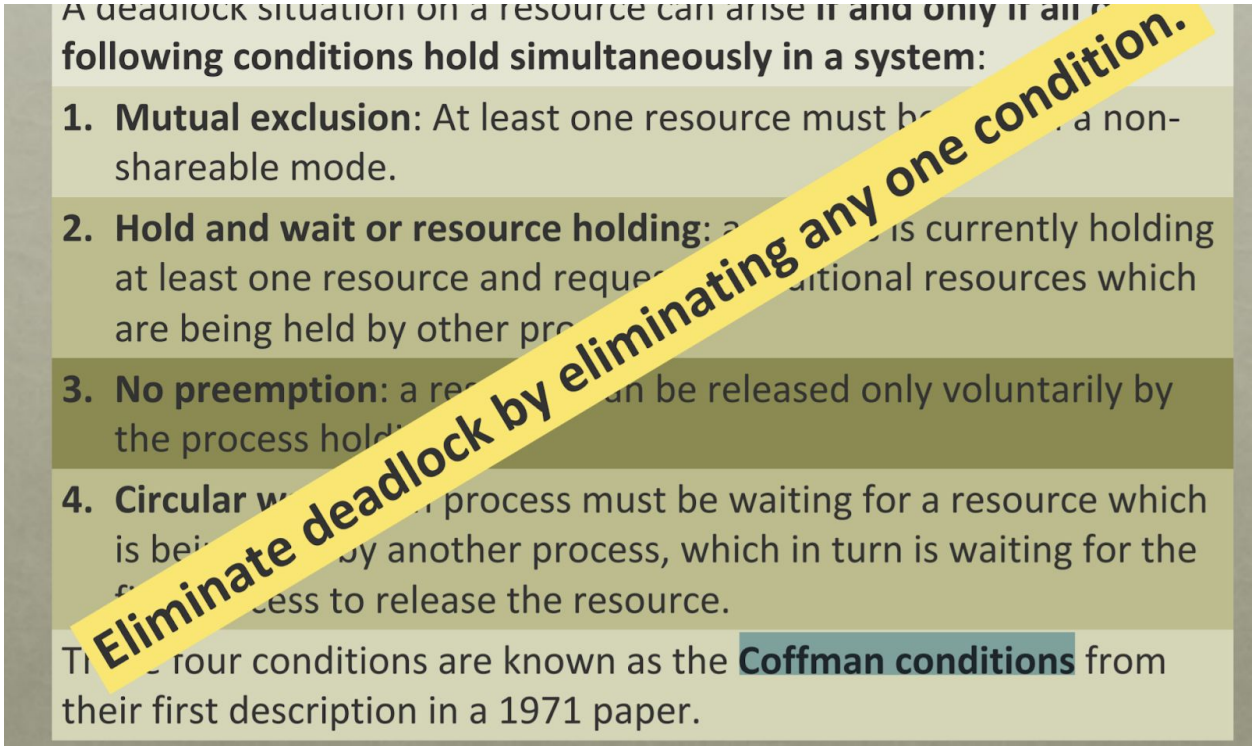
6.
Also known as four conditions of deadlock:
Mutual exclusion
Hold and wait or resource holding
No preemption
Circular wait or resource waiting

7.
Eliminate deadlock

8.
Deadlock prevention or avoidance:
Do not allow the system to get into a deadlocked state
Deadlock detection and recovery
Abort a process or preempt some resources when deadlocks are detected
Ignore the problem all together
If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection

9.
Ignore the problem all together

Coffman conditions:




Week 7

1.

True. There is normally a brick wall between the memory of different processes. Process A cannot access or modify the memory of Process B. Shared memory allows you to change that.

2.

True.

Once processes maps a shared memory segment into the process address space, there are no system calls necessary to share data between the attached processed.

3.

Shared memory does not provide any means of synchronization or mutual-exclusion for multiple or concurrent access.

These are useful tools to allow processes to exchange large amounts of information very quickly.

4.

ftruncate(): size a shared memory segment

5.

True.
A POSIX shared memory segment will exist until explicitly removed(with the shm_unlink() call) or the system is rebooted

6.
A semaphore is an integer that cannot be decremented below zero

7.
If the value of semaphore is currently zero, then a request to decrement will block until the value the value becomes greater than zero

8.
The initialization value represents the number of available resources controlled by the semaphore.

9.
True.
Threads can be used with either threads or processes.

10.
True.
And process or thread can decrement a semaphore and any process or thread can increment a semaphore.

11.
False.
A process or thread to exit, it could hold a semaphore resource

12.
POSIX semaphores come in 2 types:
Named semaphores && Unnamed semaphores

13.
sem_post() increment semaphore's value
sem_wait() decrement a semaphore's value

Week 8

1.True.
A file can go by multiple names, but two files can't have the same name.

2.
Inode:

Each file has exactly one inode number, it is unique within file system

3.
NM$L

4.
False.
Each file only have a unique inode
5.
Soft link:
The reference by name, it points to a file by file name
Hard link:
Links the files and directories in the same file system, could not traverse file system boundaries; but soft link can.

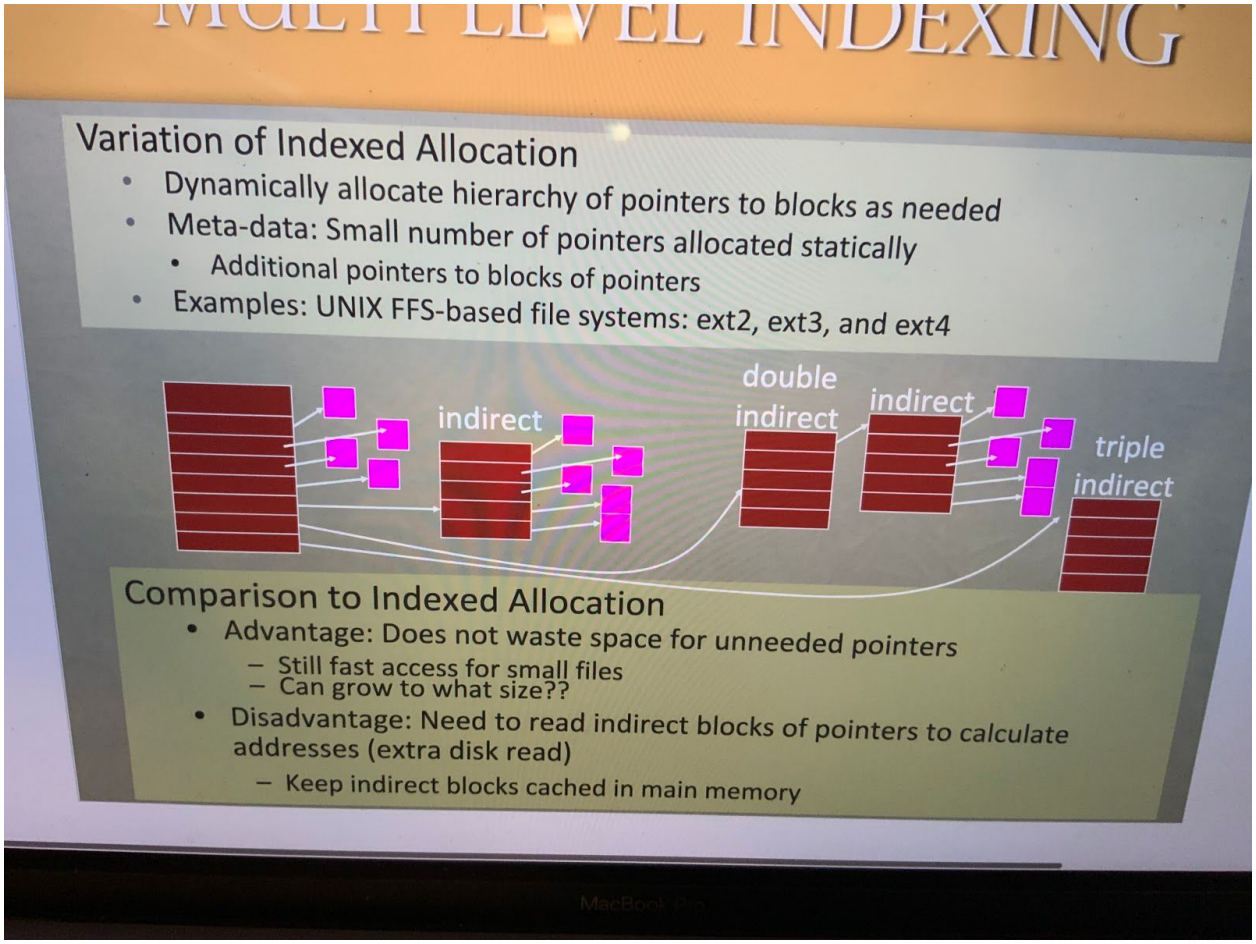
False.
Soft link point to a directory entry,
Hard link points to the inode belonging to the file; links to the directory is not allowed.

Week 9
What's Putin's favorite for Thanksgiving dinner? Ans

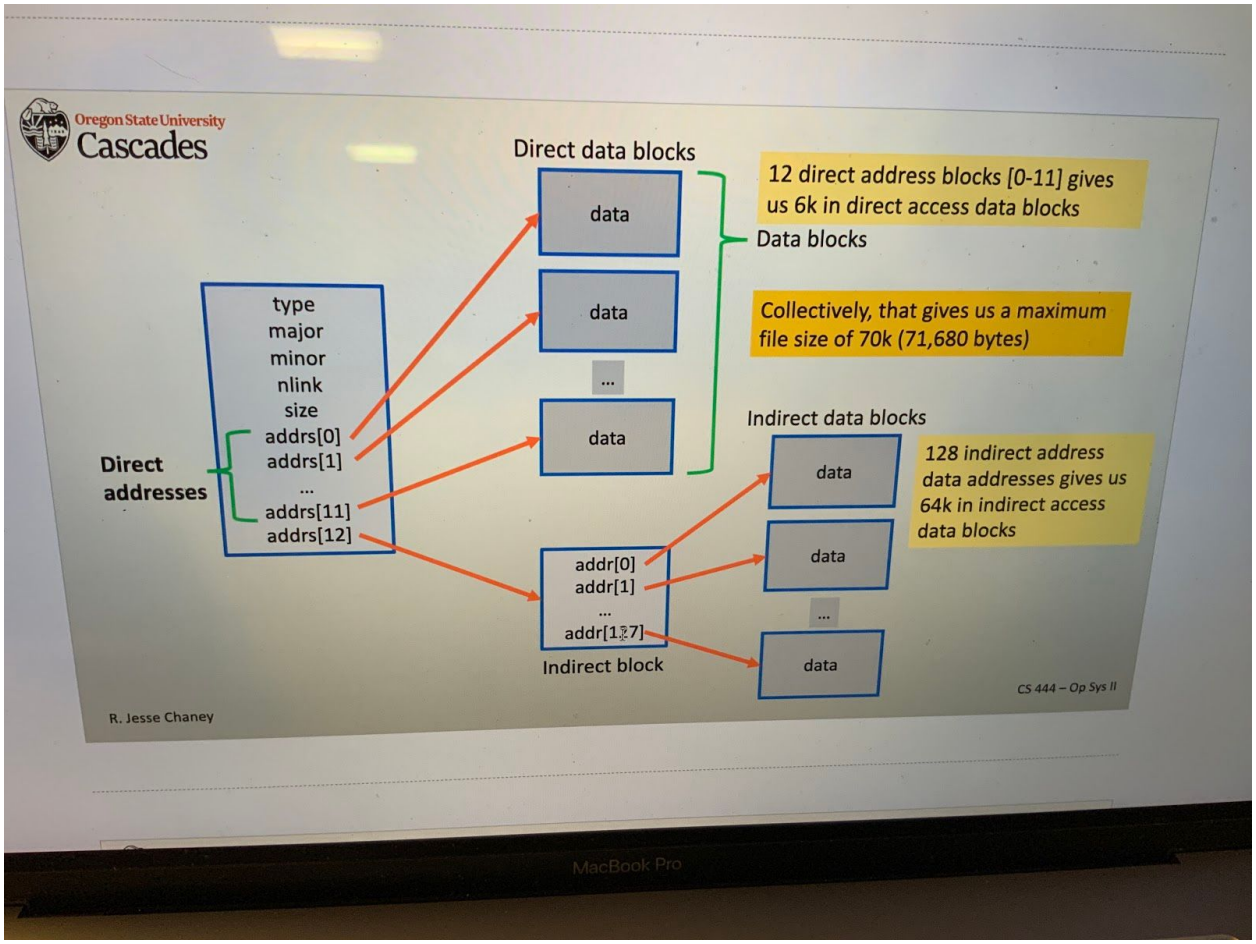
Week 10

1.
Inode:
Unique name for file system to use
Records meta-data about file: file size, permissions, etc.

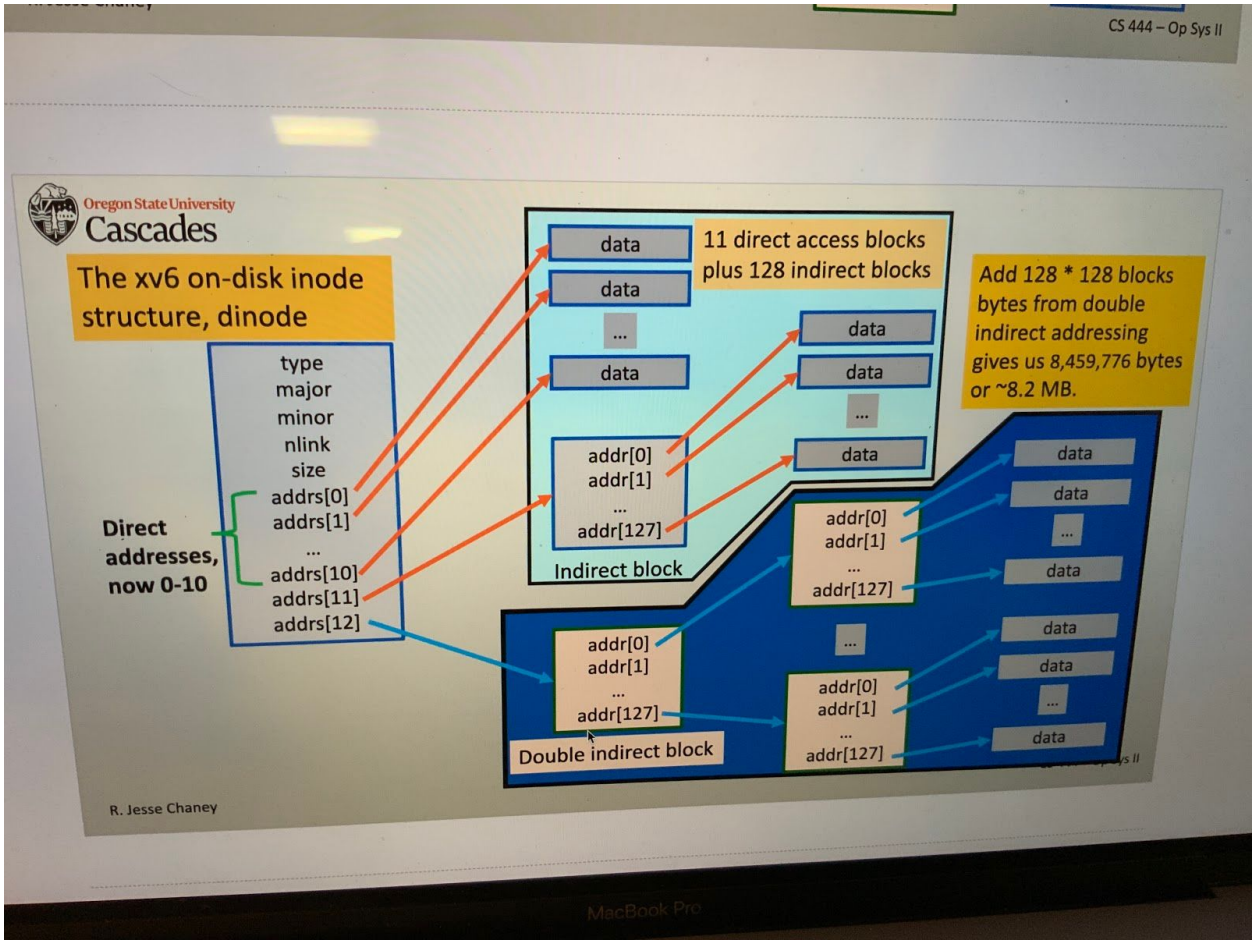
2.


MULTI LEVEL INDEXING
Variation of Indexed Allocation
• Dynamically allocate hierarchy of pointers to blocks as needed
• Meta-data: Small number of pointers allocated statically
• Additional pointers to blocks of pointers
• Examples: UNIX FFS-based file systems: ext2, ext3, and ext4
indirect
double indirect
indirect
triple indirect
Comparison to Indexed Allocation
• Advantage: Does not waste space for unneeded pointers
– Still fast access for small files
– Can grow to what size??
• Disadvantage: Need to read indirect blocks of pointers to calculate addresses (extra disk read)
– Keep indirect blocks cached in main memory

3.


Oregon State University
Cascades
Direct data blocks
data
data
...
data
12 direct address blocks [0-11] gives us 6k in direct access data blocks
Data blocks
Collectively, that gives us a maximum file size of 70k (71,680 bytes)
type
major
minor
nlink
size
addrs[0]
addrs[1]
...
addrs[11]
addrs[12]
Direct addresses
Indirect data blocks
data
data
...
data
128 indirect address data addresses gives us 64k in indirect access data blocks
addr[0]
addr[1]
...
addr[127]
Indirect block
R. Jesse Chaney
CS 444 – Op Sys II
MacBook Pro

4.




5.
System utility fsck (file system consistency check)

A tool for checking the consistency of a file system in Unix and Unix-like OS, such as Linux, macOS, and FreeBSD.

6.
super block

A record of the characteristics of a file system, including its size, the block size, the empty and the filled blocks and their respective counts, the size and location of the inode tables, the disk block map and usage information, and the size of the block groups.

7.
Beginning

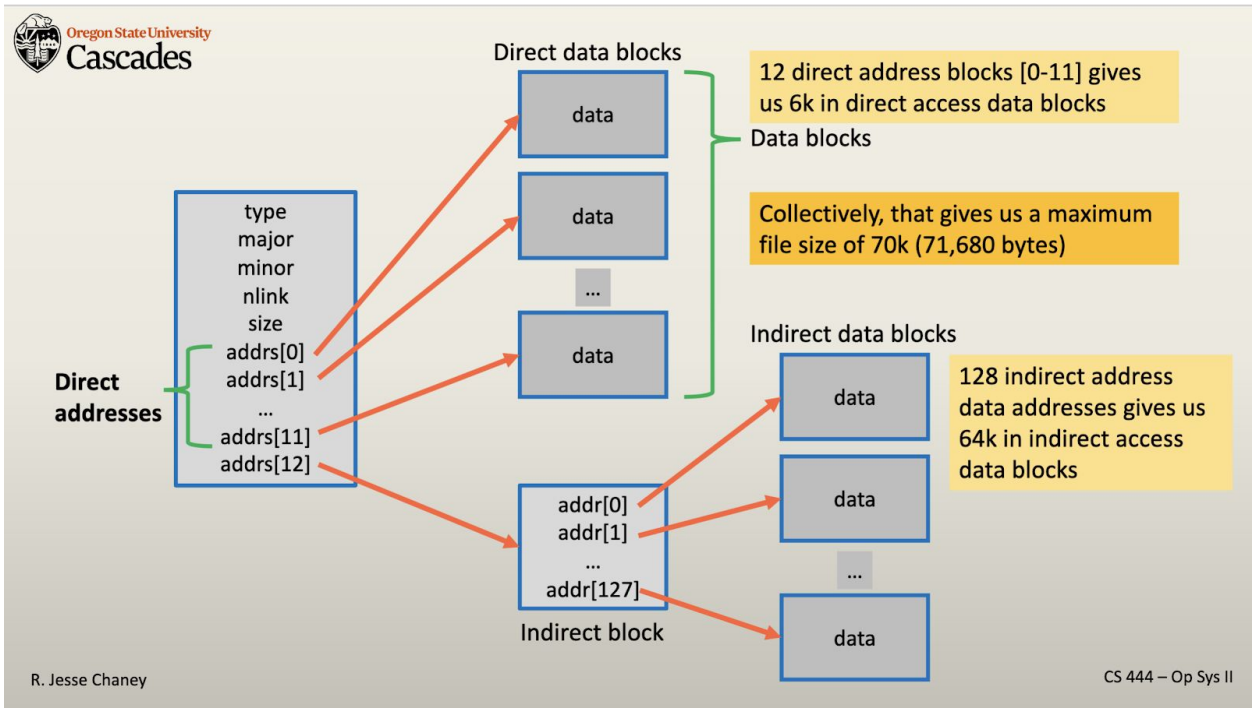
8.
?

9.

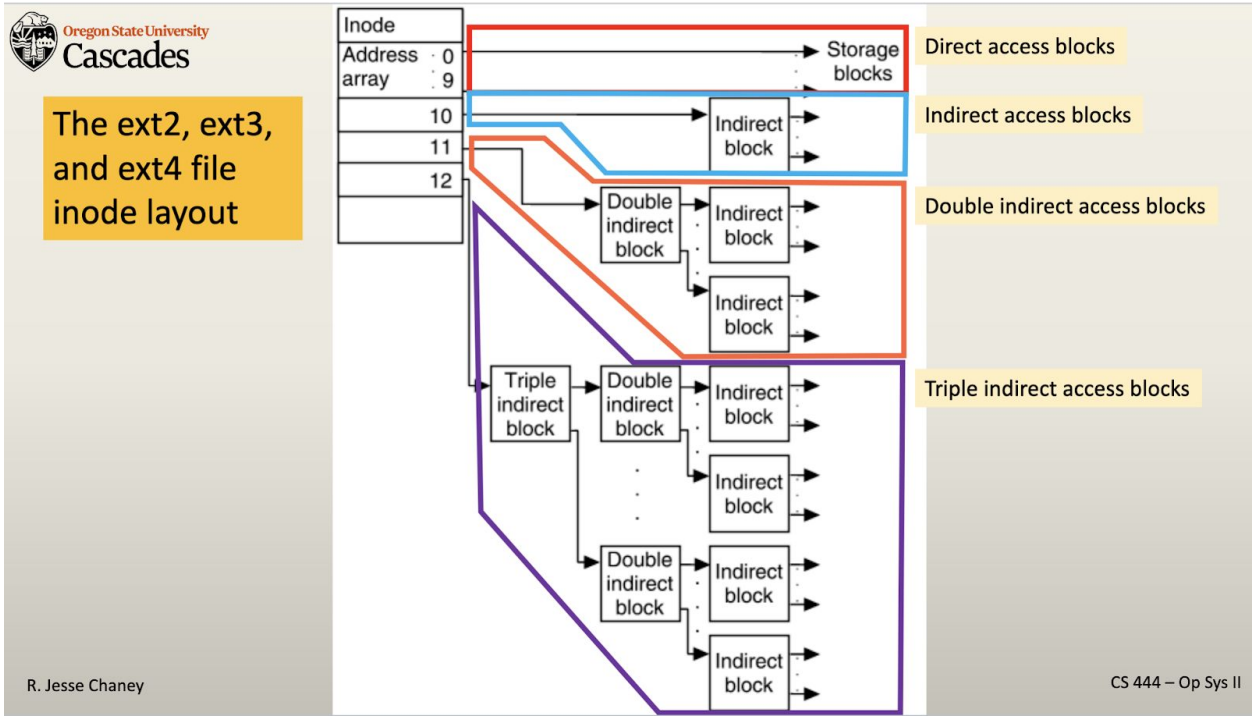
The number of inode represents the maximum number of files that a file disk can contain.

10.
Free list, Bitmaps

11.




12.




Direct access blocks,
Indirect access blocks,
Double indirect access blocks,
Triple indirect access blocks.