4 3 1
6 3 3
8 2 6
2

R1 ok T1

4 3 1
6 3 3
8 2 6
0

R2 wait

4 3 1
6 2 4
8 4 4
1

R2 ok T3.

4 3 1
6 2 6
8 4 4
1

R2 ok. T8

4 1 0
6 2 4
8 4 4
0

R3 ok T4

4 3 1
6 3 3
8 2 6

R1 OK T1

4 3 1
6 2 4
8 2 6
0

R2 waits T2

4 3 1
6 2 4
8 4 4
1

R2 OK T3.

4 4 0
6 2 4
8 2 6

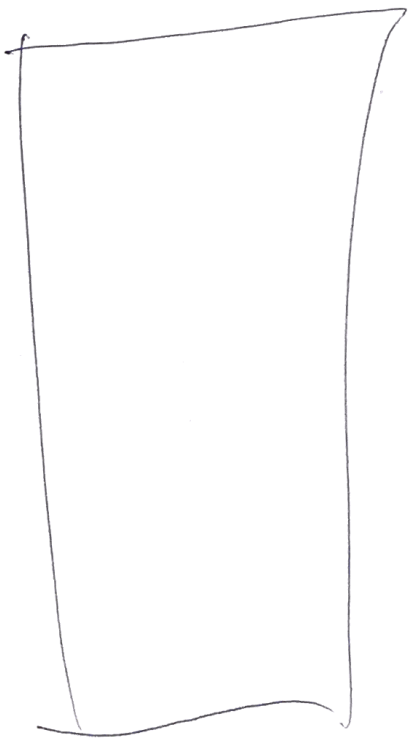
R3 OK T4.

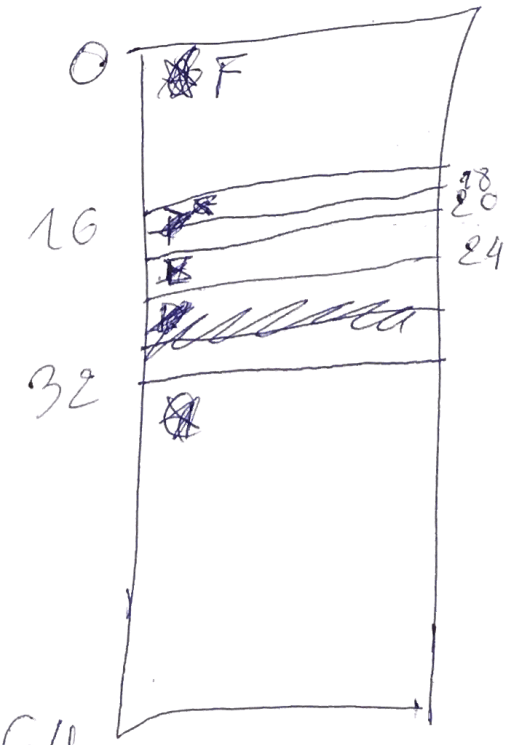
4 3 1
6 5 1
8 2 6
0

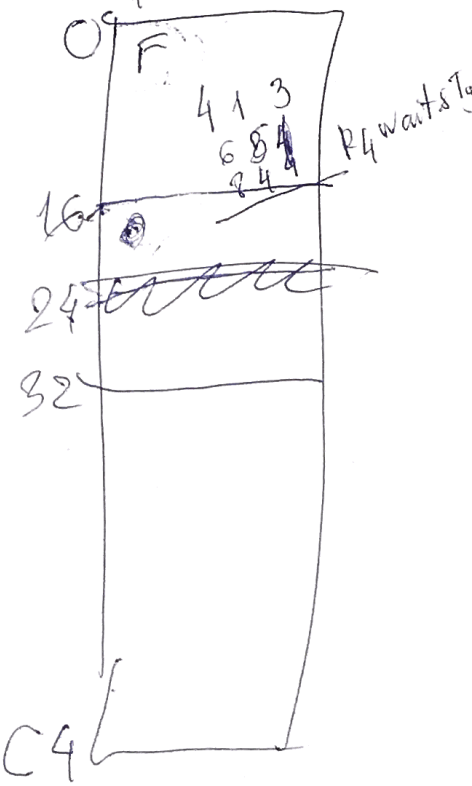
R4 waits T7

4 1 3
6 5 1
8 4 4
1

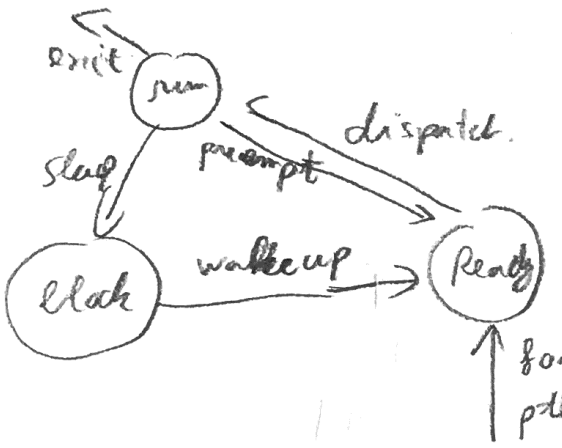
R5 OK T8
R4 waits T8

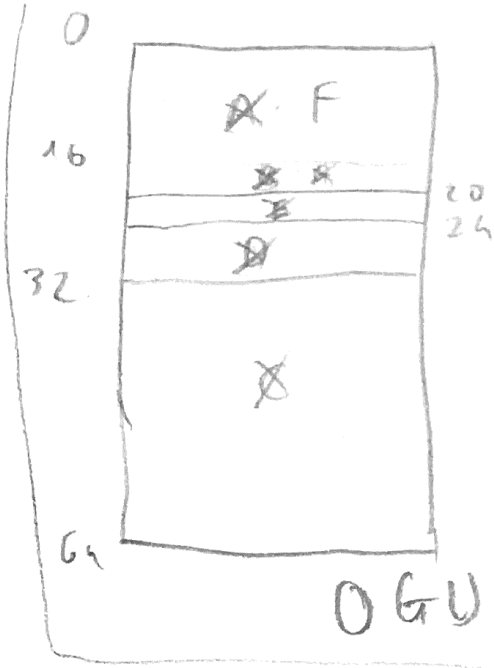






- General graph O(MN!)
- Graph reduction O(MN)
- There is more complex algo for a system of reusable b/c it already reduces all un necessary resource





A → 0
B → 16
E → 20
D → 24
C → 32

F → 0
G → 16

H → 16

20kB / 4 = 512

12 × 2k + 512 × 2k + 512² × 2k + 512³ × 2k

4 3 1
6 3 3
8 2 6
2

T1 0k (T1)

T2 wait

4 3 1
6 2 4
8 4 4
1

T2 0k (T3)

4 4 0
6 2 4
8 4 4
0

T4 0k (T4)

4 2 2
6 2 4
8 2 6

4 2 2
6 5 1
8 2 6
1

T7 0k RFA

4 0 4
6 5 1
8 4 4

5 0k T9

Fall

1) Operation available: ec_t ec_name; seq_t seq_name; void await(ec_t*, int); void advance(ec_t*); int ticket(seq_t*)
Declare your Ec(s) and Seq(s): **ec_t pEC, cEC; seq_t cSEQ;**
Void producer() { int in =0; while(1){ await(&pEC,in-10+1); ringbuff[in%10] = random();in =(in + 1); advance(&cEC);}}
Void Consumer() {int val,t; while(1){t=ticket(&cSEQ); await(pEC,t); await(&cEC,t+1); val= ringbuf[t%10]; advance(&pEC);}}
2) Void end_region (int process) { int other ; other = 1 –process; interested[process] = true; turn process; while (interested[process] = true || turn == other);} *Which line is incorrect?* **Line 4**. Why? **Mutex and bounded waiting are failed**. Fixed? interested[process] == true && turn == **thread**.
3) HPF: Highest Priority First / RR: Round Robin
What is the second major difference in the way the system treats such threads? **There is no dynamic priority adjustment in the system.**
E.20.50.2 F.10.50.5 G.20.35.1 H.15.20.4
4) In what ways do threads leave their private address space and execute in the kernel's address space?? **The way threads leave private address space and execute is by system calls or handle exeception on their CPUs.**
***B)* LEVEL 1 2 3 4 3 2 1**
5) Operation available: sem_t sem_id = initial_inter_sem_value; void p(sem_t); void v(sem_t*);
SEM/Global: **sem_t write = 1, read = 1; int count = 0;**
void reader_check_in() {p(&read); if(counter == 0) {p(&write);} ++counter; v(&read);}
void reader_check_out() { p(&read; --counter;); if(counter ==0){ v(&write); } v(&read);}.
void writer() { p(&write); // write update ; v(&write;}

6)Int main(){ int pfdout[2], pfdin[2],nreadna; char buff[81];
If(pipe(pdfout) == -1 || pipe(pdfin) ==-1); {perror("pipe"); exit(1)};
Switch(fork()){case -1: perror("fork");exit(2); Case 0; clsose(0), close (1); **Dup(pfdin[1]); Dup(pfdout[0]);** Close(pfdin[0]), Close(pfdin[1]), Close(pfdout[0]), Close(pfdout[1]);
Execlp("sort","sort","-k 1,2",NULL);
Perror("execlp") Exit(1)} Close(pfdin[1]), Close(pfdout[0]);
A=open("cs308-a2_sort_data",O_Rdonly,0); While(nread = read(a,buf,80)) write(pfdin[1],buf,nread); Close(pfdin[1]);
While(nread = read(pfdin[1],buf,80)) write(1,buff,nread); Return0;}

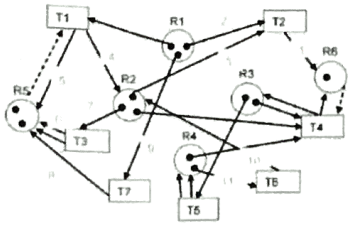
What code changes are necessary for the parent and child to run as intended.
The sort program fails because of duf() in line 18 and 19. Pfdout[0] has to be placed before pfdin[1]. So if we just swap between line 18 and 19, the program should be run correctly, and I personally also run into that problem.

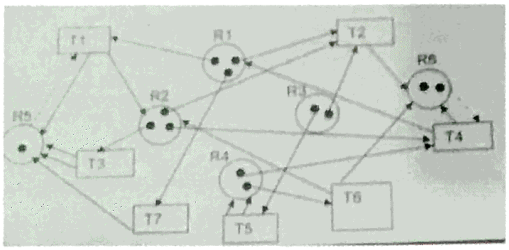
What else do we know about the actual value of the returned channel number? **Return smallest free channel in the channel table**

Spring

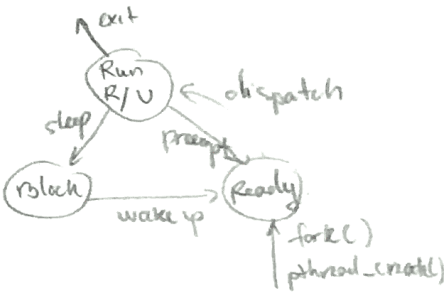
1) #define N 100; int rb[N]; int in=0 space=N objects=0; // global.. pthread_mutex_t prod, cons; pthread_cond_t pcond, ccond;
void *prod() **{ while(1){ pthread_mutex_lock(&prod); while(space==0) {pthread_cond_wait(&pcond, &prod);} rb[in%10]=random(); ++in; --space; pthread_mutex_unlocked(&prod); pthread_mutex_lock(&cons); pthread_cond_signal(&ccond); pthread_mutex_unlock(&prod);}**
void *cons() **{while(1){ pthread_mutex_lock(&cons); while(objects==0){ pthread_cond_wait(&cond, &cons);} printf("%d \n", rb[out]); out = (out + 1) % N; object--; space++; pthread_mutex_lock (&prod); pthread_mutex_unlock (&cons); pthread_cond_signal(&pcond); pthread_mutex_unlock(&cons);}**
2) Same Fall
3) A priority inversion can occur among a collection… eventually resolve these situations... sharing threads: a) explain what mechanism the kernel… recuse… among time sharing threads. **Dynamic priority adjustment is used, and as the name suggests, will dynamically adjust time sharing thread priorities**. *B)* if this situation occurs among real time threads, will the kernel… sharing threads? **No, operating systems will not perform. Dynamic Priority adjustment on real time thread, the users will often manage realtime thread Priority themselves.** ***C)*** In a single CPU system,… What aspect(feature) of time sharing… same highest priority in the READY queue? **Time slice because thread with the same priority have their order determined by their timeslice or time space running. Threads that have lower time slice are put in front of threads with larger timeslices in the reads queue.**
4) **a. System calls or handling exceptions. B). Calling fun now cnt = 3 2 1 I'm done x4.**
5) detect cars passed… Globals to observer: sem_t lock=1; int car=0;
void observer() **{ while(1) { p(&lock); car++; v(&lock);}}**
void reporter() **{while(1) {sleep (900); p(&lock); printf("current car count: %d\n", &car); car = 0; v(&lock);}**
6) grep "Feb 16"… ls generated data it reads… grep child's data.. the second
int main(){ int pchanA[2], pchanB[2], pidA, pidB, nread; char mybuf[100];
.. switch(pidA = for(){ case -1: perror("fork") exit(2); case 0: close(1); if(dup(pchanA[1] != 1){ perro("dup pchanA "); exit(3);} execlp("ls", "ls", "-1", NULL); perror("exec ls"); exit(3)….
A)… what code changes are necessary for the parent to complete:**At line 32, childA never close before execlp because this grep will wait forever for EOF. Before the execlp on line 32, close(pchanA[1]) should be inserted.** B). **The lowest available numberic channel value is returned.**

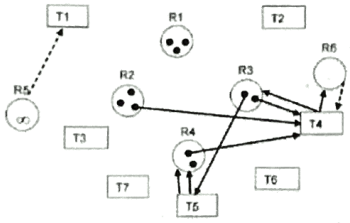
Extra: List 4 necessary conditions for deadlock to occur in CS: **Mutex Resources, Hold and Wait, No Pre-emption, Circular Wait.** B) Can deadlock occur… if not, which…: **No DL, the circular wait condition is denied by imposing strict allocation ordering**. c) Can indefinite postponement occur? **No IP, IP is a problem when denying No Pre-emption**
Void producer() {while(1){ p(&prod);buff[in]=random(); in=(in+1)%10; v(&cons)}}
Void Consumer() {int val,t; while(1){p(&cons);val=buf[out]; //print val somewhere; out=(out+1)%10; v(&prod)}

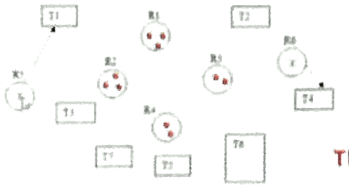




b. General graph O(mn!), more complex than reusable-only with is O(mn)







There is NO DL

LFU: *b) In our discussion of memory object… file based… explain how the pages…. Of dirty pages that must be backed up for possible reuse.* ***Dirty, but in use anonymous pages, must be backed up to the swap disk, while such pages selected for removal from a file object must be backed up to a file system disk. (OR—The anonymous objects could be saved under swap disk where is the file based objects could be saved under file system disk.)***

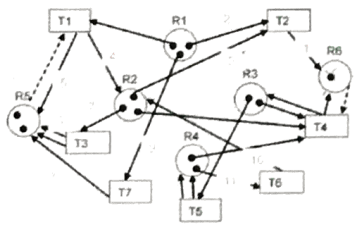
*) The inode based ….An element (or block) … implementations is 8KB. **A).** *Why are elements allocated instead of just allocating one sector at a time to a growing file* ? **Elements conserve pointers and provide better disk performance** ***B).*** *What type of fragmentation does this kind of allocation lead to*? **Internal fragmentation of ½ an allocation unit (element) per file object. C).** *If we have a UNIX type file system which uses 64KB elements, and contains a set of files that all grow sequentially over time, what can we expect the average amount of fragmentation per file to be at any given time ?* **½ an allocation unit (element) in the case of a 64 KB element leads to 32 KB internal fragmentation per file object on average**

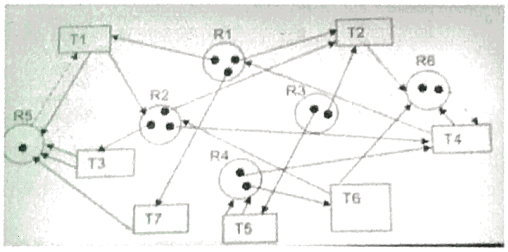
Spring 2016 1) *Consider the following resource-allocation policy for a fixed inventory of serially reusable resources of three different types: | **a)** List the 4 necessary conditions for a DL occur in a computing system.* ***Mutex Resources, Hold and wait, No Pre-emption, Circular Wait.*** *| **b)** Can deadlock occur in the system described above? If you...it can If you think it cant, describe ...that would be required for dl.* ***No, Because there is pre-emption on block resources. (Or No DL can occur, the NO-PREEMPTION condition is being denied.)*** *|**c)** Can indefinite postponement occur in the system described? Explain. (**Yes, Because there is pre-emption or blocked resources, a blocked process can remain blocked as its resources are being taken to allow other progresses. Or -- Lost work and the need to retreat and start over can lead to IP*

2.) *On an x86 single core Linux platform, when the Initial Thread (IT) of a newly created process (a process just created by a fork() call)....| **a)** Explain why thread encounters an immediate TLB miss (**New Thread is result of HW context switch, all user space TLBs have been invalidated. Or-- The TLB is tied to a process address space and when operating off a forked process in user space, it becomes separated from that address space and it will fault. b)** Explain how it's possible for the thread to get an L1 hit after filling the missed TLB entry.* ***A new process is created in a fork() call, so code and data are in cache from parent. Or –Because the process is duplicate (more or less) created via fork(), the target of the walk could still be there despite the initial fault. c)*** *Is there a page fault or context switch involved in the sequence described above??* ***No, if the page is in memory and in cache, the TLB table walk cannot fault or context switch. Or—No because even if the walk has an L1 hit, because the TLB missed the hit is ignored.***

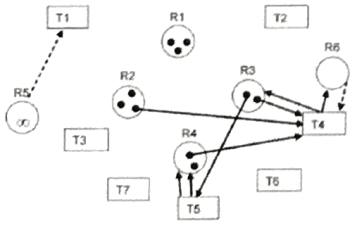
Spring 2015. 1) ***th0() and *th1(), Color!. **a)** Show what output is produced by this process: **Color Initialized to red (line 16) and line 29, org - green, green - org. b)** Although some progress is made in this process the process never finishes.* 1.***The condition var is never signalled by th0, so th1 cannot leave its cond wait and IT is stuck in join.*** Using the line numbers... *what code you add to allow process to come to normal termination.* ***(pthread_cond_signal(&color_condx); must add before return NULL in *th0 and *th1, after line 44 and 54, must signal condx var:***

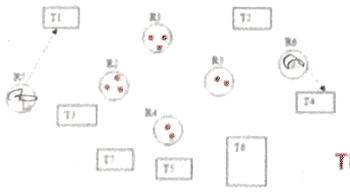
Fall 2016. 1) *The following simple program name th_run compiles and links with no errors, but when its executed it produces the follwing output but never completes. **a)** Provide a detailed explanation of why this program never finishes:* ***The main thread is hung in join, since it joins in create sequence, but threads may finish in any sequence (here 1 is followed by 5). If it can't join, it cannot unlock to the next thread. b)*** *If we run this program repeatedly, could we ever expect a particular execution to complete? Explain:* ***Yes, if in a given run threads could finish in any order, we could expect that they would occasionally complete in create sequence and satisfy the join loop.***





b. General graph O(mn!), more complex than reusable-only with is O(mn)





LFU: *b) In our discussion of memory object... file based... explain how the pages.... Of dirty pages that must be backed up for possible reuse.* ***Dirty, but in use anonymous pages, must be backed up to the swap disk, while such pages selected for removal from a file object must be backed up to a file system disk. (OR—The anonymous objects could be saved under swap disk where is the file based objects could be saved under file system disk.)***

Spring 2016 1) *Consider the following resource-allocation policy for a fixed inventory of serially reusable resources of three different types: |* ***a)****List the 4 necessary conditions for a DL occur in a computing system.* ***Mutex Resources, Hold and wait, No Pre-emption, Circular Wait.*** **| b)** *Can deadlock occur in the system described above? If you...it can If you think it cant, describe ...that would be required for dl.* ***No, Because there is pre-emption on block resources. (Or No DL can occur, the NO-PREEMPTION condition is being denied.)*** **|c)** *Can indefinite postponement occur in the system described? Explain. (****Yes, Because there is pre-emption or blocked resources, a blocked process can remain blocked as its resources are being taken to allow other progresses. Or -- Lost work and the need to retreat and start over can lead to IP***

2.) *On an x86 single core Linux platform, when the Initial Thread (IT) of a newly created process (a process just created by a fork() call)....|* ***a)*** *Explain why thread encounters an immediate TLB miss (****New Thread is result of HW context switch, all user space TLBs have been invalidated. Or-- The TLB is tied to a process address space and when operating off a forked process in user space, it becomes separated from that address space and it will fault. b)****Explain how it's possible for the thread to get an L1 hit after filling the missed TLB entry.*** ***A new process is created in a fork() call, so code and data are in cache from parent. Or –Because the process is duplicate (more or less) created via fork(), the target of the walk could still be there despite the initial fault. c)*** *Is there a page fault or context switch involved in the sequence described above??* ***No, if the page is in memory and in cache, the TLB table walk cannot fault or context switch. Or—No because even if the walk has an L1 hit, because the TLB missed the hit is ignored.***

Spring 2015. 1) ****th0() and *th1(), Color!.*** ***a)*** *Show what output is produced by this process:* ***Color Initialized to red (line 16) and line 29, org - green, green - org. b)*** *Although some progress is made in this process the process never finishes.* 1.***The condition var is never signalled by th0, so th1 cannot leave its cond wait and IT is stuck in join.*** Using the line numbers... *what code you add to allow process to come to normal termination.* ***(pthread_cond_signal(&color_condx); must add before return NULL in *th0 and *th1, after line 44 and 54, must signal condx var:***

Fall 2016. 1) *The following simple program name th_run compiles and links with no errors, but when its executed it produces the follwing output but never completes.* ***a)*** *Provide a detailed explanation of why this program never finishes:* ***The main thread is hung in join, since it joins in create sequence, but threads may finish in any sequence (here 1 is followed by 5). If it can't join, it cannot unlock to the next thread. b)*** *If we run this program repeatedly, could we ever expect a particular execution to complete? Explain:* ***Yes, if in a given run threads could finish in any order, we could expect that they would occasionally complete in create sequence and satisfy the join loop.***

Final

8). Process A (Directory / d rwx r-x r-x) Process B (Directory /abc d rwx r-x ---)
Process C (File /abc/file_one - r-s rwx rwx) Process D (File /abc/file_two - --- r--- rw-)
Process D (File /abc/file_three - rw- r—rw-) **A)** Can D write on file_three ? Yes, the write bit is on for public group
B) can A write on file_one? Yes, process A has evid/vid 0 (root). **C)** can C use chmod 644 /abc/file_two shell command? Yes, it is the owner of the file. **D)** can B use ls /abc command? Yes, group ID match, has read access. **E)**assum that …file_one is program… attempts to creates a file in /abc call file_x. if B use excel() system call to load and run /abc/file_one, can B success create the file in /abc? No, it does match group ID, but that group doesn't have write permission for the file /abc. **F)** can A send process E to termination signal? A can do whatever it want dues to being as root. **G)** Can E send process C termination signal? Only process A(root) can kill other process. **H)** can B write on /abc/file_one? Yes, same group. **I)** can C write on /abc/file_two? No, the owner has no read and no write.

3). Both the unix.. windowsxp operating… some unique thread… run state…. All times. A*) if all available user thread are blocked ... enter the context switching to block... find thread to run?* **Thread priority**. *B) When unix or xp thread is run in user mode, it is... in what way do thread leave their private... execute in the kernel's address space?* **System call and exceptions.**

5). *What code changes are necessary?* **The child never close the write side of the pipe. All "close(pchan[1]) after execlp (before line 17). B)** *When a single... unblocked SIGSEGB signal,... will terminate.. arrangs for the process.... When an unblocked SIGSEGV is delivered to the process?* **Setup a single handler.**

6.) FAT16 vs UFS. *A. Explain what this memory usage problem is for FAT when compared to an i-node based system?* **The entire FAT table (indexes for all files in the file system) is loaded into RAM when a volume is initially referenced, while only the specific i-nodes associated with current open files are brought into RAM using the Ext type file systems (only i-nodes of objects in use brought into RAM.** *B) Explain why Microsoft's FAT 16 implementation limits the size of a file system to 2GB of total space. Make sure you clearly identify the limiting factor.* **Since a FAT 16 table has only 2^{16} entries, and since Microsoft limits the maximum allocation unit size (cluster size) to 2^{15} (32 KB) the maximum single file size (as well as the maximum entire file system size) is limited to $2^{16} * 2^{15} = 2^{31}$ = 2GB**

(12 direct * 2K) + (512 * 2K) + (512^2 * 2K) + (512^3 * 2K) = 275 GB
Journaling provides support for rapid reboots after power failures by limiting the file system checking to only those things active at failure.