CS 354 Operating Systems

Study Question Solutions

1 Processes and Threads

1. Longer quanta reduce scheduling overhead, since scheduling decisions are made less frequently. Longer quanta can increase response time for interactive processes.

2.

Proc. Number	Finish Time	Response Time
1	6	6
2	14	13
3	12	9

Average response time: $\frac{6+13+9}{3} = 9.33$

3.

Proc. Number	Finish Time	Response Time
1	10	10
2	14	13
3	13	10

Average response time: $\frac{10+13+10}{3} = 11$

This solution assumes that (1) at time 3, process 3 gets put onto the ready queue before process 1, whose quantum is expiring at that time, and (2) that when a process exits during its quantum, the first ready process runs during the remainder of that quantum. Different assumptions give different answers.

4.

Proc. Number	Finish Time	Response Time
1	4	4
2	14	13
3	8	5

Average response time: $\frac{4+13+5}{3} = 7.33$

5. Response time is the time between a process' creation and its exit, i.e., it is the amount of time that a process exists in the system. For a round-robin scheduler, the solution depends on the order of the processes in the ready queue. Assuming that process k is at the front of the ready queue and that other processes appear in decreasing order of required computing time, we can write the following expression for $\bar{R}$, the average response time:

$$\bar{R} = \frac{k + (k + (k-1)) + (k + (k-1) + (k-2)) + \ldots + \sum_{i=1}^{k} i}{k}$$
$$\bar{R} = \frac{\sum_{i=1}^{k} i^2}{k}$$

For a SJF scheduler, we have:

$$\bar{R} = \frac{1 + (1+2) + (1+2+3) + \ldots + \sum_{i=1}^{k} i}{k}$$
$$\bar{R} = \frac{\sum_{i=1}^{k} i(k-i+1)}{k}$$

6. • block waiting for a device (e.g., the console)

- block waiting for another process (e.g., Join() in Nachos)
- terminate

7. To create a new process, the operating system must create a new thread of control, a new address space, and new instances of any per-process data structures (such as open file tables and page tables). It must initialize the address space and the per-process tables. It must assign a unique identifier to the process, and it must place the process on the ready queue.

   To create a new thread within a process, the operating system must create a new thread of control and must associate it with an existing address space and data structures. It must assign a unique identifier to the new thread, and it must place the thread on the ready queue.

8. If the threads are not supported by the operating system, T_{21}, T_{22}, or T_{31} will run during the next quantum. (This assumes that none of the processes are blocked and that they are scheduled round-robin by the operating system.)

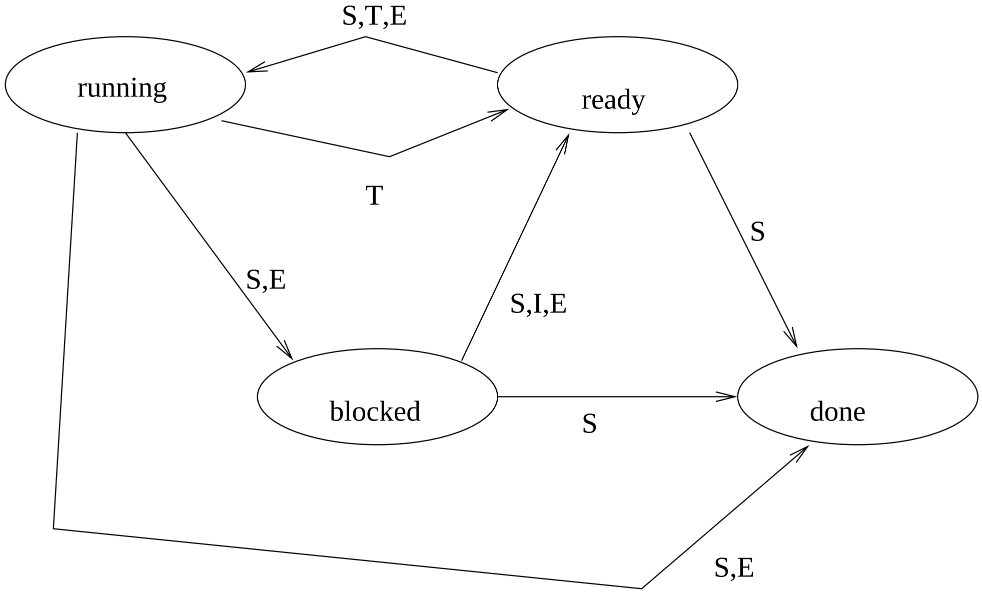
   If the threads are supported by the operating system, then any thread except T_{11} will run during the next quantum, assuming not all threads are blocked and scheduling is round-robin.

9. If the threads are not supported by the operating system, either T_{12} or T_{13} will run after T_{11} finishes.

   If the threads are supported by the operating system, any of the other threads may run.

10. A non-preemptive scheduler assigns the processor to a process until that process is finished. A preemptive scheduler may force a runnable process to yield the processor before it is finished so that another runnable process can be allowed to run.

11. If you consider the Nachos Yield() system call, an arc labeled "S" should be included from "running" to "ready".



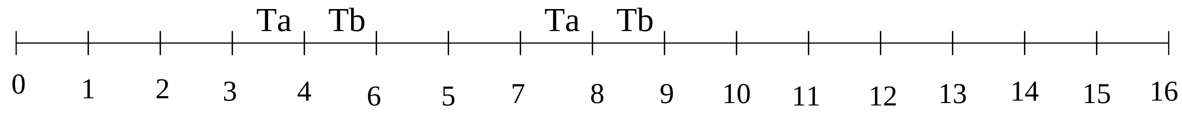
12. **a.** Process 1 will receive one time unit, process 2 will receive 2 units, and process 3 will receive three.

    b. Each process will receive two time units.

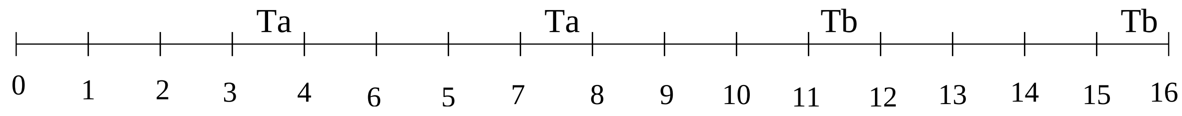
    c. Scheduler (b) is a fair share scheduler. Scheduler (a) is not. Scheduler (a) could be modified to become a two-level scheduler. The top level would select a process to run, using a fair scheduling discipline like round-robin. The bottom level would select one thread from the process that was selected by the top level scheduler.

13.

a)



b)



14. When $Q \geq T + S$, the basic cycle is for the process to run for T and undergo a process switch for S. Thus, 1. and 2. have an efficiency of $\frac{T}{T+S}$. When the quantum is shorter than $S + T$, each run of T will require $T/(Q - S)$ process switches, wasting time $ST/(Q - S)$. The efficiency here is then

$$\frac{T}{T + ST/(Q - S)}$$

For 4., by setting $Q = S$ in the formula above, we see that the efficiency is zero - no useful work is done.

15. For round robin, during the first 10 minutes each job gets 1/5 of the CPU. At the end of 10 minutes, C finishes. During the next 8 minutes, each job gets 1/4 of the CPU, after which time D finishes. Then each of the remaining three jobs get 1/3 of the CPU for 6 minutes, until B finishes, and so on. The finishing times for the five jobs are 10, 18, 24, 28, and 30, for an average of 22 minutes. For priority scheduling, B is run first. After 6 minutes it is finished. The other jobs finish at 14, 24, 26, and 30, for an average of 20.0 minutes. If the jobs run in the order A through E, they finish at 10, 16, 18, 22, and 30, for an average of 19.2 minutes. Finally, shortest job first yields finishing times of 2, 6, 12, 20, and 30, for an average of 14 minutes.

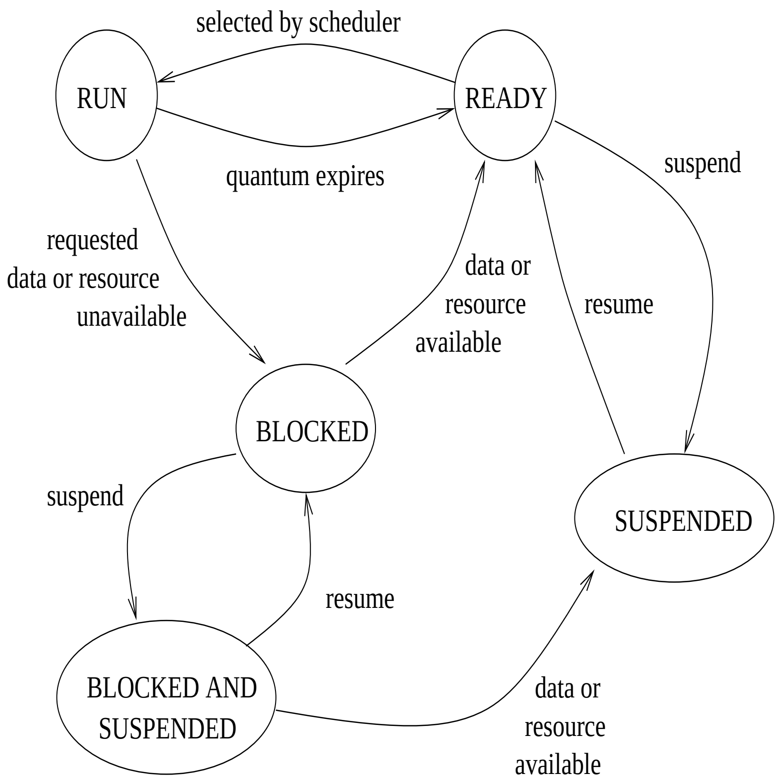
16. Once the high priority process blocks (after running for time t_c), the low priority process will run for a quantum. If the high priority process is still blocked after that quantum, the low priority process will receive another. When the high priority process unblocks, the low priority process will finish out the current quantum, at which point the scheduler will give the processor to the high priority process. Thus, for every t_c time the high priority process runs, the low priority process will run for

$$\lceil \frac{t_b}{q} \rceil q$$

units of time. The fraction of CPU time used by the low priority process is:

$$\frac{\lceil \frac{t_b}{q} \rceil q}{\lceil \frac{t_b}{q} \rceil q + t_c}$$

17.



18. **a.** Suppose that $2^{p-1} \leq c < 2^p$. Initially, the process will climb to priority $p-1$ since

$$c \geq 2^{p-1} > \sum_{i=0}^{p-2} 2^i$$

It will then drop to $p-2$ after blocking during its $p-1$ quantum. If $2^{p-2} + 2^{p-1} \leq c < 2^p$, the process will move to p_1 and then p when it next runs, and then alternate between $p-1$ and p forever. If $2^{p-1} \leq c < 2^{p-1} + 2^{p-2}$ then the process will alternate between $p-2$ and $p-1$ forever.

b. Assuming both types of processes have been in the system for a long time, the Bs will have a very low priority and will only run when A is blocked. So, they get fraction

$$\frac{b}{c+b}$$

c. If there are enough A process, the Bs will never get to run.

19. **a.** If a process uses its entire size q quantum, the scheduler will allow that process to run until either it finishes or it blocks, since it will have higher priority than all others. If the process blocks, it will rejoin the round-robin q-quantum queue. In other words, this is essentially a round-robin scheduler that may give some processes a longer "turn" than other processes. Round-robin schedulers do not cause processes to starve.

b. Suppose that a process (P_1) uses its entire size q quantum, and then a steady stream of new processes arrives in the system such that there is always at least one such process ready to run. These processes will have higher priority than P_1, and the scheduler will always choose to run them rather than P_1. P_1 will starve.

20. **a.** Because the thread implementation is user-level, the Consumer cannot run while the Producer reads from the file. Thus, the total time for the two threads will be the sum of the times for the two individual threads, or $10(t_r + t_c)$.

b. With an OS thread implementation, the Consumer can run while the Producer is blocked. There are two cases to consider:

$t_c \leq t_r$**:** In this case, the Consumer will finish consuming buffer[i] before the Producer finishes producing buffer[$i+1$]. The total time will be the time required to produce all ten buffers plus the time for the Consumer to consume the final buffer, i.e., it will be $10t_r + t_c$.

$t_c > t_r$**:** In this case, the Consumer is the bottleneck. The total time will be the time required to consume all of the buffers, plus the time the Consumer must wait while the first buffer is being produced, i.e. the total time will be $t_r + 10t_c$.

2 Virtual Memory

1. There can be at most $2^{21}/2^{10}$ frames in main memory, so a page table entry will require 11 bits for a frame number, plus protection and reference bits, etc. A page table may have up to $2^{24}/2^{10} = 2^{14}$ entries. A virtual address has 24 bits. Virtual address 1524 lies on page 1 which resides in frame 8 with an offset of 1524 modulo 1024, or 500. So, 1524 maps to physical address $8 * 1024 + 500 = 8192 + 500 = 8692$. Physical address 10020 is in frame 9, with an offset of 804. Virtual page 4 maps to frame 9, so virtual address $4 * 1024 + 804 = 4900$ will map to 10020.

2. The diagram should include a virtual address that is 22 bits wide. The most significant 3 bits of the virtual address should be used to select one of eight entries from a segment table. A base register, 21 bits wide, should point to the physical address of the first entry in the segment table. Each segment table entry should include the physical address (21 bits) of the first entry of a page table for that segment. The entry should also include the length of the page table. Seven bits are required since each page table may have up to $2^{19}/2^{12} = 2^7$ entries. Seven bits of the virtual address (the seven bits to the right of the 3 segment selector table bits) are used to select one page table entry from the page table. The PTE should include a 9 bit frame number plus protection and other bits. The remaining 12 (least significant) bits of the virtual address are used to select one of the 2^{12} offsets within the frame.

3. For LRU:

Frame	Reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2
1		2	2	2	2	2	2	3	3	3	3	4	4	4	6	6	6
2				3	3	3	4	4	4	4	2	2	2	5	5	5	1
Fault?	Y	Y	N	Y	N	N	Y	Y	N	N	Y	Y	N	Y	Y	Y	Y

The total number of faults is 11.

For clock:

Frame	Reference																
	1	2	1	3	2	1	4	3	1	1	2	4	1	5	6	2	1
0	1	1	1	1	1	1	4	4	4	4	4	4	4	5	5	5	5
1		2	2	2	2	2	2	2	1	1	1	1	1	1	6	6	6
2				3	3	3	3	3	3	3	2	2	2	2	2	2	1
Fault?	Y	Y	N	Y	N	N	Y	N	Y	N	Y	N	N	Y	Y	N	Y

The total number of faults is 9.

4. Reading virtual address v will require four memory accesses in case of a TLB miss, one for the segment table, and one for each level of page table, and one for the actual physical memory location corresponding to v. In case of a TLB hit, only one memory access (for the physical location itself) will be required. So, if h is the hit ratio of the TLB, we need $4T(1-h) + T(h) = 2T$, i.e., $h = 0.67$.

5. $2^{(a+b+c)}$, since there are $2^{(a+b+c+d)}$ total addresses in the address space, and there are 2^d on each page.

6. Swapping means moving entire address spaces between the disk and the memory. Paging means moving individual pages, so that part of an address space may be on disk while the other part is in main memory.

7. Segments are variable-sized, and are visible to user processes. Pages are fixed-size and are not visible.

8. The operating system must locate the page table for the process that is to start running. It must set the base register in the MMU so that it points to the page table in memory, and it must set the length register if present. Finally, it must clear any now-invalid cached address translations from the TLB.

9. One method is a length register, which can be used to limit the length of a single page table. Another method is to use multiple levels of page tables. A lower-level page table that is not needed is simply not allocated, and its corresponding entry in a higher level table is marked as invalid.

10. An associative memory is one that is accessed by content rather than by address. Often an "entry" in an associative memory will have two parts, a tag and a value. To access the associative memory, one supplies a tag. If that tag matches the tag of an entry in the associative memory, that entry is returned. In a hardware implementation (e.g., a TLB), the tags of all entries are checked in parallel.

11. The frame to page mapping is useful during page replacement. The operating system must know which page resides in each frame so that it can check the page table entry for that page and so that it can mark that page table entry as invalid if the page is replaced.

12. Coalescing holes means combining one or more adjacent holes to form one larger hole.

13. Advantages of larger page sizes:

    - smaller page tables
    - faster sequential access to on-disk portions of the address space, since a few large blocks can be retrieved more quickly than many small ones

    Disadvantages:

    - increased wasted space in the last page of the address space (internal fragmentation).

14. page size: 2^{10}, max segment size = 2^{16}

15.

    a. Frames 0 and 2 are selected for replacement and are added to the free list. The new page tables should look as follows:

    Process 1 Page Table

	Frame	R	M	V
0	6	0	1	1
1	4	0	1	1
2	5	1	0	1
3	0	0	0	0
4	1	0	0	1
5	9	1	0	1

    Process 2 Page Table

	Frame	R	M	V
0	0	0	0	1
1	7	1	0	1
2	8	0	0	1
3	2	0	1	1
4	0	0	0	0
5	3	0	1	1

    b. Page 4 of process 2 must be moved from disk to memory. If you choose to replace frame 0, no pages are moved to the disk from memory, and the new page table for process 2 is as follows:

	Frame	R	M	V
0	0	0	0	0
1	7	1	0	1
2	8	0	0	1
3	2	0	1	1
4	0	1	0	1
5	3	0	1	1

    If you choose to replace frame 2, page 3 of process 2 must be copied from disk to memory, and the new page table for process 2 is as follows:

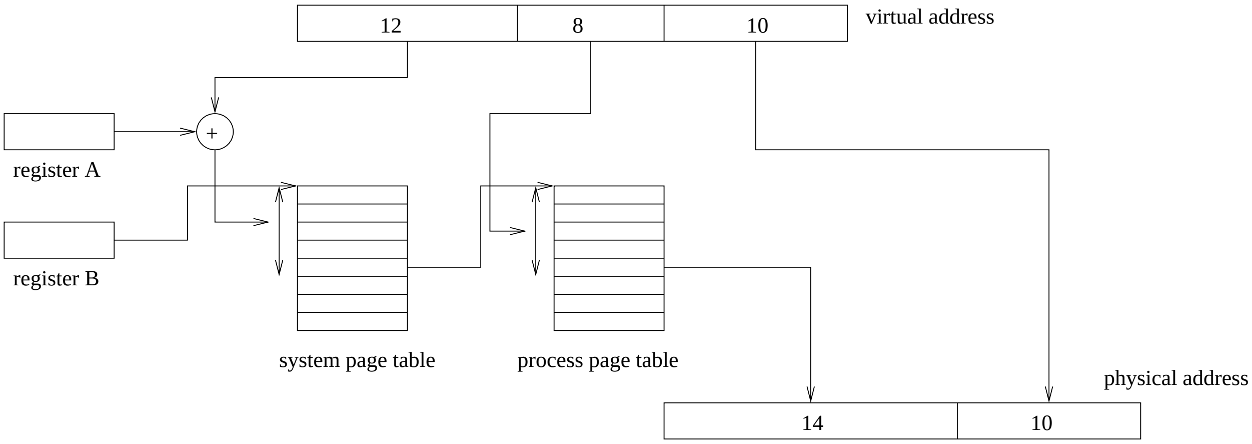
	Frame	R	M	V
0	0	0	0	1
1	7	1	0	1
2	8	0	0	1
3	0	0	0	0
4	2	1	0	1
5	3	0	1	1

16. **a.** $\boxed{100{,}650} \rightarrow \boxed{1000{,}50} \rightarrow \boxed{4000{,}700} \rightarrow \boxed{4900{,}100}$

    b. $\boxed{100{,}500} \rightarrow \boxed{700{,}50} \rightarrow \boxed{800{,}25} \rightarrow \boxed{1000{,}200} \rightarrow \boxed{3800{,}900} \rightarrow \boxed{4900{,}100}$

17. **a.** Register A requires only 20 bits, since the page tables are page-aligned in the system's virtual address space. Register B requires 24 bits. The max. page table size is 2^{22} bytes.

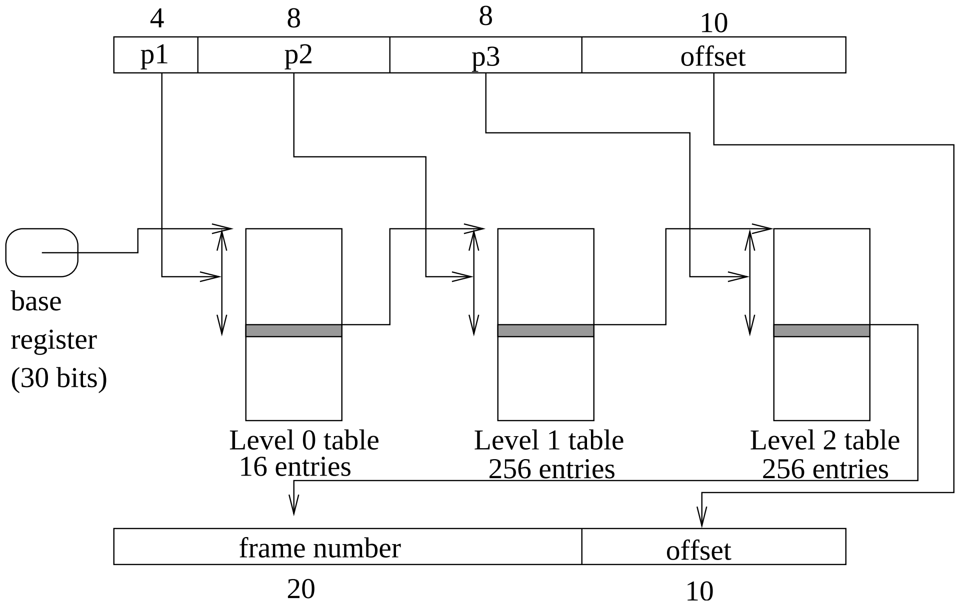
   b. This is a tricky question! The value in register A is combined with 12 bits from the virtual address to produce an offset into the system's page table. If register A contains a complete virtual address, it should be divided by the page size before being combined with the bits from the virtual address. (This division operation is not shown in the diagram.) If register A holds a page number, no division is needed.



18. • The TLB is used as a cache for the page table, which resides in primary memory.
    • Primary memory is used as a cache for pages, which reside in secondary memory.

19. • The code segment contains the executable program.
    • The data segment contains initialized and uninitialized global variables and dynamically-allocated variables (the heap).
    • The stack segment contains the program stack.

20. The lowest level of page tables requires up to $2^{30}/2^{10} = 2^{20}$ page table entries. At most 2^8 page table entries can fit on one frame. So, at least three levels of page tables will be required. This diagram assumes maximum page table sizes of 2^4 at the top level and 2^8 at the two lower levels.



21. **a.** (a) Any virtual address in the 300's. This will cause page 3 to be paged in and page 1 to be paged out.
    (b) Any virtual address in the 100's. This will cause page 1 to be paged in and page 4 to be paged out.

(c) Any virtual address in the 400's.

b. Any sequence of three addresses in which all are less than 500 and none are in the 300's.

22. **a.** physical address 130

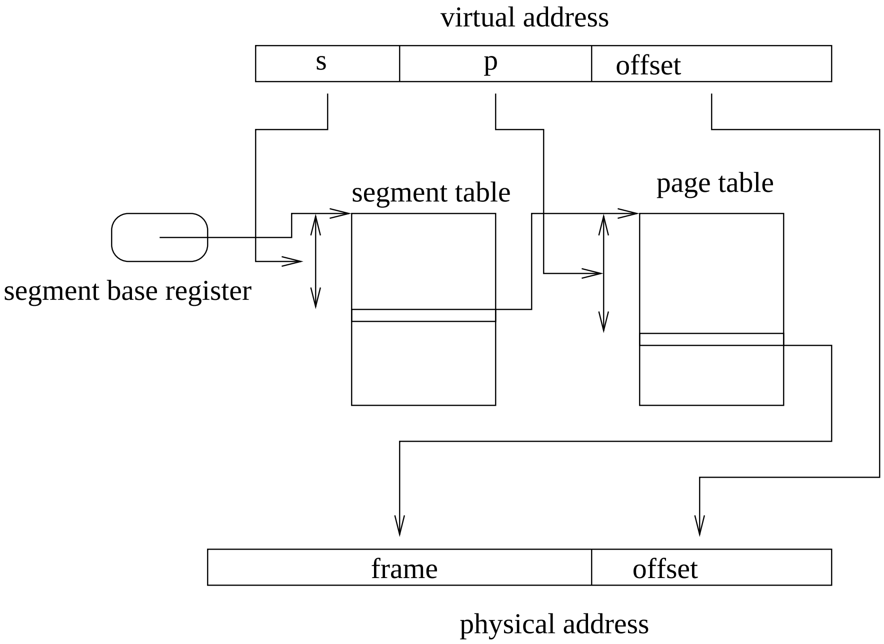
    b. does not map

    c. virtual address 250 of process 1

23. **a.** 2998,2999,0000,0001

    b. 1998,1999,3000,3001

24.



25. The effective instruction time is 100h + 500(1-h), where h is the hit rate. If we equate this formula with 200 and solve for h we find that h must be at least 0.75.

26. The aging algorithm is not a stack algorithm unless some mechanism is provided for breaking "ties" among pages with the same age. The mechanism should order tied pages the same way regardless of the size of the memory. For example, ordering based on page number will work.

27. LRU requires that recency information be updated every time a page is referenced. Since one or more references may occur each time an instruction is executed, a software implementation would, at best, be extremely inefficient.

28. Under a global policy, a page fault by process P_1 may cause replacement of a page from P_2's address space. Under a local policy, a fault by P_1 can only cause replacement of P_1's pages - the total number of frames allocated to each process remains constant.

29. The software can maintain a R bit in each PTE. It can use V and R as follows:

V	R	Meaning
0	0	page is in memory, but not referenced
0	1	page is not in memory
1	0	not used
1	1	page is in memory and referenced

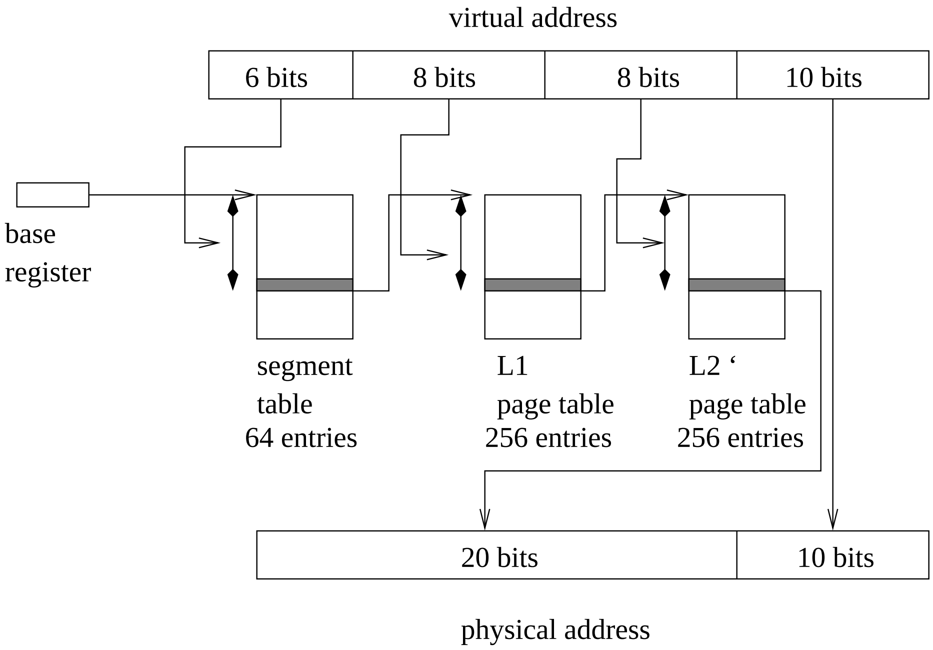
- When the OS wishes to clear the R bit, it should clear the V bit as well. (This is a 11 to 00 transition.) This will cause a page fault to occur the next time the page is referenced.
- When a page fault occurs (V is 0), the OS checks the R bit to determine whether the page is actually in memory or not. If R is 0, the page is in memory, and the OS sets both R and V to one. This indicates that the page is referenced, and it prevents further page faults. (We only need to get a fault on the first reference to the page that occurs after the R bit has been cleared). If R is 1, the page is not in memory. This is a regular page fault. The OS must bring the page into memory. It sets V and R to one for the new page (since it is about to be referenced). The page table entry for the replaced page gets V set to one and R to zero, to indicate that it is no longer memory-resident.

30.

a. Two levels are required, because there are 2^{16} pages per segment and 2^8 page table entries can fit in a single page table.

b. 2^{26} bytes.

c.



31. The graph should have page fault rate on the dependent (vertical) axis and page frames assigned on the independent (horizontal) axis. The page fault rate should be low and (roughly) flat if the number of page frames assigned is greater than W, and should increase as the number of frames drops below W. The graph should be non-increasing with as additional frames are assigned because the replacement algorithm is a stack algorithm.

32. The MMU hardware causes an exception to occur. (This transfers control to the operating system's exception handler.)

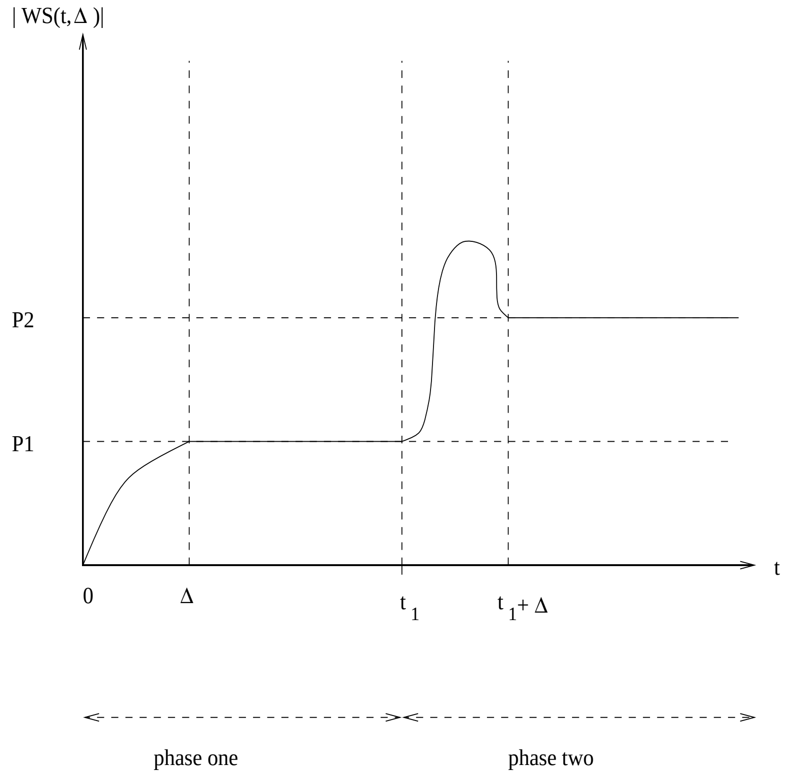
33. This solution skips lots of details. The general idea is to use the P bit to force an exception to occur when a shared page is about to be updated.

    When the a child process is created, its page table should be created as a duplicate of the parent's table, i.e., both parent and child point to the same in-memory copy of each page. The P bit should be set for all entries of both tables (parent and child) so that an attempt to update any page will result in an exception.

    When a protection fault occurs, the offending page should be duplicated. One process's page table should be made to point to the duplicate and other other to the original. The P bit can be cleared for both processes so that further updates do not cause exceptions.

This solution assumes that the only thing the protection mechanism is used for is to implement copy-on-update. If the protection mechanism is used for other things, then the OS needs a way to distinguish pages that are protected for copy-on-update reasons from pages that are protected for other reasons. (In one case, a protection fault should cause the page to be duplicated, in the other it should not.) One way for the OS to handle this is to maintain a copy-on-update bit in each page table entry.

34.



35.

ref str	1	2	1	3	2	1	1	4	3
stack	1	2	1	3	2	1	1	4	3
		1	2	1	3	2	2	1	4
				2	1	3	3	2	1
								3	2
dist str	∞	∞	2	∞	3	3	1	∞	4

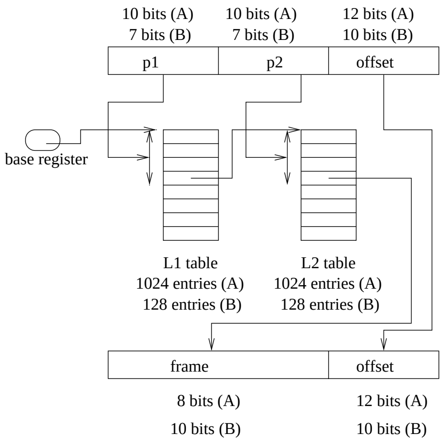
36. The optimal page replacement policy (OPT) can be used to determine the required lower bound. There will be at least five faults.

ref str	1	2	3	2	3	1	3	1	2	1
frames	1	1	3	3	3	3	3	3	2	2
		2	2	2	2	1	1	1	1	1
fault?	x	x	x			x			x	

37. **a.** This is a write into segment 0. It will fail (protection fault) because segment 0 is read-only.

b. This is a read from segment 1, offset 1. This corresponds to physical address $1000 + 1 = 1001$.

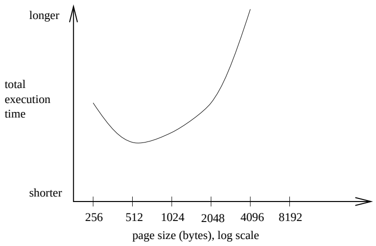
c. This is a write to segment 3, offset $4000 - 3072 = 928$. Segment 3 is only 300 bytes long, so this will result in a segmentation fault (segment length violation).



38. **a.**

b. A total of 3000 level two PTEs will be required for the code and data. These will occupy $\lceil \frac{3000}{1024} \rceil = 3$ level two page tables. 100 PTEs for the stack segment will occupy another level two page table. In addition, there is a single level 1 page table, for a total of 5 page tables (occupying one frame each).

39. **a.** Suppose that part or all of a process' working set is not in memory and must be paged in. If pages are very small, more page faults will be required to bring in the missing parts of the working set. This is what drives the curve up when the pages get too small. Large pages reduce the number of page faults required to bring in the working set. However, as pages become larger it becomes more likely that they will contain non-working-set data as well as working-set data. This unnecessary data will occupy memory space that could have been used for the working set. This drives the curve up when the page size gets too large.



b. This curve assumes that faults for larger pages take longer to handle than faults for smaller pages. This will tend to drive the minimum of the curve towards the left (where each page fault is cheaper). The curve sketched assumes this effect is strong enough to make a 512 byte page size better than an 1024 byte page size, though the latter results in fewer faults. If it is assumed that all faults take the same amount of time to handle, the minimum of the curve should remain at 1024.

40. The minimum is zero memory accesses, since on a TLB hit the entire translation is found in the TLB. The maximum is 3 access in case of a TLB miss, since 3 page tables will need to be queried to complete the translation.

41. Clock is not a stack algorithm. Clock can degenerate to FIFO - FIFO is subject to Belady's anomaly and so is not a stack algorithm.

42. **a.** The OS will need to read in the filename parameter to the Open system call. In the worst case, this may span two pages of the address space.

b. As a result of the Read call, the OS will need to update "buf", which is 1500 bytes long. In the worst case, this could span three 1024-byte pages.

c. The buffer resides at virtual addresses 2100 through 3599, i.e, on pages 2 and 3 of the address space. Virtual 2100 is offset 52 on page 2, which is in frame 1. Virtual 3599 is offset 527 on page 3, which in in frame 4. So, the buffer occupies physical 1076 to 2047, and physical 4096 to 4623.

3 Disks and Other Devices

1. - bytes/cylinder = TSB
   - to reduce rotational latency, increase the rotational velocity
   - to reduce transfer time, increase the rotational velocity, or increase the quantity SB, which represents the number of bytes per track

2. Since 500 sectors fit on each cylinder, disk block 596 resides on cylinder one (the second cylinder).
   - seek distance is $20 - 1 = 19$ cylinders
   - seek time is $3 + 0.025 * 19 = 3.5$ms
   - transfer time is $16.7/50 = .3$ms
   - expected rotational latency is $16.7/2 = 8.3$ms
   - expected service time is $3.5 + .3 + 8.3 = 12.1$ms
   - worst case service time is as above, but with the expected rotational latency replaced by the worst case latency of 16.7 ms, so service time is $3.5 + .3 + 16.7 = 20.5$ms
   - best case service time is as above, but with the expected rotational latency replaced by the best case latency of 0ms, so service time is $3.5 + .3 + 0 = 3.8$ms.

3. - SSTF: 74,20,19,400,899
   - SCAN: 400,899,74,20,19 (assuming the initial direction is up)
   - CSCAN: 400,899,19,20,74 (assuming the initial direction is up)

4. Block devices store and retrieve data in units of fixed-size blocks. Character devices supply (or consume) streams of characters, i.e., bytes.

5. **a.**
   - seek time = 0
   - transfer time = $\frac{1}{S}\frac{1}{\omega}$
   - rotational latency = $\frac{1}{\omega} - d$

   service time = $\frac{1}{S}\frac{1}{\omega} + \frac{1}{\omega} - d$

   b. the smallest integer k such that $\frac{k}{S}\frac{1}{\omega} \geq d$, i.e., $k = \lceil dS\omega \rceil$

6. **a.**
   - seek time = $5 + 0.05 * 10 = 5.05$
   - rotational latency = $\frac{1}{2}\frac{1}{\omega}$
   - transfer = $\frac{1}{S}\frac{1}{\omega}$

   service time = seek time + rotational latency + transfer time

   b. This rotational latency given below assumes that $d + t_{seek} <= 1/\omega$. If this is not the case, then the subtracted term should be $d + t_{seek}$ modulo $1/\omega$
   - seek time = $5 + 0.05 * 1 = 5.05$
   - rotational latency = $\frac{1}{\omega} - (d + t_{seek})$
   - transfer = $\frac{1}{S}\frac{1}{\omega}$

   service time = seek time + rotational latency + transfer time

7. DMA is direct memory access, the direct transfer of data between a disk controller and memory without passing through the processor. An interrupt is generated to notify the processor of completion of a DMA operation.

8. **a.** For the first request, $t_{seek} = 100(0.1) + 5 = 15$, $t_{rot} = (1/2)10 = 5$, and $t_{trans} = (1/10)10 = 1$, giving $t_{service} = 21$. For the second request, $t_{seek} = 0$ and t_{rot} and t_{trans} are as for the first request. This assumes that the delay between requests is unknown so that the disk is equally likely to be at any rotational position when the second request arrives. The sum of the service times is 27 milliseconds.

   b. We have $t_{seek} = 100(0.1) + 5 = 15$, $t_{rot} = (1/2)10 = 5$ and $t_{trans} = 2/10(10) = 2$, giving $t_{service} = 22$.

9. The seek plus rotational latency is 40 msec. For 2K pages, the transfer time is 1.25 msec, for a total of 41.25 msec. Loading 32 of these pages will take 1.32 seconds. For 4K pages, the transfer time doubles to 2.5 msec, so the total time per page is 42.5 msec. Loading 16 of these pages takes 0.68 seconds.

10. (a) 10 + 12 + 2 + 18 + 38 + 34 + 32 = 146 tracks = 730 msec
    (b) 0 + 2 + 12 + 4 + 4 + 36 + 2 = 60 tracks = 300 msec
    (c) 0 + 2 + 16 + 2 + 30 + 4 + 4 = 58 tracks = 290 msec

11. Interleaving means skipping sectors when laying data out on a disk so that DMA can occur while the skipped sectors spin beneath the heads. Its purpose is to reduce rotational latency.

12. Video RAM is memory in which addresses correspond to locations on a display screen. It serves as an interface between the CPU and the display.

13. A device driver is software: that part of the OS that is responsible for interacting with a particular device. A device controller is a piece of hardware that controls the device and implements an interface between the device and the rest of the system.

14. Increasing the number of sectors per track will decrease the transfer time since more sectors will pass under the read/write head in a given amount of time. Seek times and rotational latencies are unaffected, so the total service time will decrease

    Increasing the number of tracks per surface will not affect the transfer time or the rotational latency. Assuming the surface itself remains the same size, it is possible to make an argument that it may reduce seek times slightly (since track i and track j will be closer together on the disk with more tracks per surface), but this depends on the sequence of data requests and on how the data is distributed across the disk tracks.

15. An asynchronous device does not require that the CPU block while the device handles a request.

16. **a.** For the first request, the seek distance is 100 cylinders and the seek time is 15 milliseconds. Max rotational latency is 10 milliseconds, expected is 5 milliseconds, and the min is 0 milliseconds (this solution will assume expected time). Transfer time is 5/25 of 10 or 2 milliseconds, for a total expected service time of 22 milliseconds. The second request requires no seek time, but has the same rotational latency (expected) and transfer time as the first, for a total service time of 7 milliseconds. In sum, the two requests require 29 milliseconds.

    b. The first request requires a 15ms seek. Rotational latency is zero, and 10ms a required to transfer the whole track into the buffer, for a total time of 25ms. The second request requires zero time to service since the required data is in the buffer. Total service time for the two requests is 25ms.

17. A serial interface is used to exchange data one bit at a time through a single address or port. A memory-mapped interface uses one address per location (character or pixel) on the terminal screen.

18. **a.** Read Order: $40, 37, 27, 90$
    Distance: $2 + 3 + 10 + 63 = 78$

    b. Read Order: $90, 40, 37, 27$
    Distance: $48 + 50 + 3 + 10 = 111$

    c. For SCAN, the worst case distance is twice the number of disk cylinders (200 for version A, 400 for version B), since for a request at one edge, the heads may need to travel all the way across the disk and back. For SSTF, the worst case is unbounded. Requests may starve if new requests arrive continuously near the currently location of the heads.

19. **a.** Disk head scheduling can reduce seek times by changing the order in which requests are serviced. Track buffering completely eliminates seek times when a request hits the buffer.

    b. Interleaving can reduce rotational latencies by leaving space between logically consecutive sectors, in anticipation of delays between requests. Track buffering can completely eliminate rotational latencies, as above.

    c. Track buffering can eliminate transfer times. (Data must still be delivered from the disk controller to memory, but it need not be read from the platters.

4 File Systems

1. The per-process table is used to ensure that a process cannot access file table entries created for other processes.

2. File systems implement objects (files) of arbitrary size which can grow and shrink over time. Files can have names and other attributes. The file system also controls shared access to a storage device.

3. $10(2^{13}) + ((2^{13})/4)2^{13} + ((2^{13})/4)((2^{13})/4)2^{13}$ bytes

4. Open() and close() are provided so that the costs of locating a file's data and checking access permissions can be paid once, rather than each time a file is accessed. When a UNIX open() call is performed, the file's i-node is cached and new entries are created in the open-file table and in the per-process descriptor table of the process that issued the open().

5. An i-node's reference count keeps track of the number of hard links to that i-node.

6. Random access to file blocks is faster with an indexed layout scheme. Chained layout does not require a separate index.

7. One technique is to use a bit map, another is to maintain a list of the block numbers of free blocks.

8. Fragmentation refers to space that is wasted when files are laid-out on a disk.

9. Directory entries contain a pathname component and an i-number. The i-number is assumed to refer to a file in the same file system as the directory.

10.
```
int translate(string pathname) {
int inum = 0;
int i;
    FOR i FROM 1 TO num_components(pathname) {
        IF is_regular(inum) THEN return(INVALID);
        inum = dsearch(inum,get_component(i,pathname);
        IF inum < 0 THEN return(INVALID);
    }
    return(inum);
}
```

11.
```
int symlinkcount = 0;
int translate(string pathname) {
int inum = 0;
int i;
    FOR i FROM 1 TO num_components(pathname) {
        IF is_regular(inum) THEN return(INVALID);
        inum = dsearch(inum,get_component(i,pathname);
        IF inum < 0 THEN return(INVALID);
        IF is_symlink(inum) DO
            symlinkcount = symlinkcount + 1;
            if (symlinkcount > 8) return(INVALID);
            inum = translate(get_symlink(inum));
            if (inum == INVALID) return(INVALID);
        ENDWHILE
    }
    return(inum);
}
```

12. reference count, ID of file owner, access permissions, direct and indirect pointers to file data blocks, file size, file type

13. Two disk operations are performed, both because of the read() request. The first operation retrieves the second file data block from the disk, and the second operation retrieves the third file data block.

14. lseek(3,0);
    read(3,buffer,1024);
    /* modify buffer here */
    lseek(3,0);
    write(3,buffer,1024);

15.

symlink(/b/y, /a/d/x)	NO - /a/d/x already exists
link(/b/y, /a/d/x)	NO - /a/d/x already exists
symlink(/c, /b/z)	OK
link(/c, /b/z)	NO - /c does not exist
symlink(/a/c/f, /b/f)	OK
link(/a/c/f, /b/f)	OK
symlink(/a/c, /b/g)	OK
link(/a/c, /b/g)	NO - cannot hard link to a directory

16. Hierarchical name spaces allow files to be organized into groups, e.g., by user. This also helps to avoid name conflicts. If the name space is structured as a DAG, users have more flexibility. For example, two users working as a team can place their work in a single shared subdirectory.

17. /, /a, /a/b

18. /, /a, /a/b, /, /d, /d/e

19. 3 blocks: two data blocks plus one pointer (indirect) block

20. 203 blocks: 200 data blocks, plus the single indirect pointer block, plus two double indirect pointer blocks

21. The max number of modified blocks is three: one data block plus two double indirect pointer blocks. The max cost will be paid when $L_G = 110,000$. This is the largest file that can be represented using only the direct and single indirect pointers. The next byte added to the file will cause two double indirect blocks to be allocated, along with a new data block.

22.
```
rename(string oldpath, string newpath) {
int i;
int old, oldp, newp;
    old = 0;
    for i from 1 to num_components(oldpath) do
        oldp = old;
        old = dsearch(old,get_component(i,oldpath))
        if (old < 0) return(ERROR)
    endfor;
    newp = 0;
    for i from 1 to num_components(newpath)-1 do
        if (newp == old) return(ERROR)
        newp = dsearch(newp,get_component(i,newpath))
        if (newp < 0) return(ERROR)
    endfor;
    newp = dinsert(newp,get_component(num_components(oldpath),oldpath),old);
    if (newp < 0) return(ERROR);
    old = ddelete(oldp,get_component(num_components(oldpath),oldpath));
    return(SUCCESS);
    }
```

23.
- advantages:
  - fewer pointers required in index nodes
  - improved performance on sequential file access, since one large block can be retrieved more quickly than several smaller blocks
- disadvantage:
  - poor space utilization because of increased internal fragmentation

24.
- advantage:
  - fewer disk operations are required to access a file's attributes
- disadvantage:
  - hard links (multiple pathnames for the same file) are difficult to implement

25. B - (F MOD B)

26.
```
link(string oldname, string newname) {
int oldi,newpi,i;
    /* find the i-number of oldname */
    oldi = 0;
    for i from 1 to num_components(oldname) {
        oldi = dsearch(oldi,component(i,oldname));
        if ( oldi < 0 ) return(INVALID);
    }
    /* make sure that oldname is not a directory */
    if ( NOT is_regular(oldi) ) return(INVALID);
    /* find the i-number of the parent of newname */
    /* first make sure that newname is not the root, since the root has no parent */
    if ( num_component(newname) < 1 ) return(INVALID);
    newpi = 0;
    for i from 1 to num_components(newname) - 1 {
        newpi = dsearch(newpi,component(i,newname));
        if ( newpi < 0 ) return(INVALID);
    }
    /* insert a new entry in the parent of newname */
    if ( dinsert(newpi, component(num_component(newname),newname), oldi) < 0)
        return(INVALID);
    else
    return(OK);
}
```

27. $2^{13} + 2^{18} + 2^{26}$

28.
- if $0 \leq N < 8$, then one operation is required
- if $8 \leq N < 256 + 8$, then two operations are required
- if $256 + 8 \leq N < 2^3 + 2^8 + 2^{16}$, then three operations are required

29.
- if $0 \leq N < 8$, then one operation is required
- for $i \geq 1$, if $8 + 255(i-1) \leq N < 8 + 255i$, then $i + 1$ operations are required

30.

	single indirect pointer	access permissions	file size	file type	reference count
`link("/a/b","/c")`	-	-	-	R	W
`symlink("/a/b","/c")`	-	-	-	-	-
`open("/a/b")`	-	R	-	- or R	-
`write(d,buf,amount)`	W	-	W	- or R	-
`read(d,buf,amount)`	R	-	R	-	-

31. The global index must hold 1024/4 = 256 pointers, each of which occupies 2 bytes, for a total of 512 bytes. Four sectors are required to hold these 512 bytes.

32. string

```
showpath(int i) {
    string path,s;
    int i,p;
    path = NIL;
    while ( i ≠ 0 ) {
        p = dsearchByName(i,“..”);
        s = dsearchByNumber(p,i);
        path = concat(s,“/”,path);
        i = p;
    }
    return(path);
}
```

33. **seek:** requires no block operations

write: bytes 10100 through 11099 of the file must be updated. These bytes partially occupy blocks 10 and 11. These blocks must be read in, modified, and written back to the disk. Reading them in requires 3 block reads, since an indirect block must be read, in addition to the two data blocks. Writing them out requires 2 block writes.

read: bytes 11100 through 12899 must be read. These bytes fall on blocks 11 and 12. Block 11 is already in the cache, so it need not be written. One block read is required for block 12.

34. To create the new file, the root directory must be updated to add an entry for the new file, the free list must be updated to reflect the space used for the new file's header, and the new file's header itself must be created.

- Directory:
  - read the header for the root directory file (1 read)
  - create new data block (in memory) containing directory entry and write it to the disk (1 write)
  - update the header to reflect new file length, and possibly new timestamp (1 write)
- Free List:
  - read in data block(s) containing free list (1 read, or more)
  - update free list to reflect new file header and new directory data block (1 write, or more)
  - update file header (optional), which is already in memory (1 write)
- New File:
  - create new file header in memory and write it to the disk (1 write)

35.

- Link Count: 4
- Size: 2
- Read Timestamp: 1,2,3

- Write Timestamp: 2

36. In general, the blocks may not be contiguous, so three disk read operations will be needed. The final block is only partially used - the answer may assume that it is read in completely or partially. In the first case the answer should be $3(50 + 4) = 162$. In the second case it should be $2(50 + 4) + (50 + 2) = 160$.

37. Since the disk is not full, it should be possible for the file to be laid out as a single extent. This can be read in a single disk read operation requiring $50 + 10 = 60$ milliseconds.

38.

    a. The Seek requires no disk read or write operations. The Write partially updates blocks 5 and 6 of the file. Blocks 0 through 5 are located through the direct pointers in the i-node, while block 6 is located through the single indirect block. A total of three disk reads are required for reading in the two blocks plus the indirect block. Eventually, the updated versions of blocks 5 and 6 need to be written back out to the disk, but this need not happen right away because of the copy-back policy. Similarly, timestamps in the i-node may be updated but this need not be written immediately to the disk. The above assumes that the Write does not cause the file to grow. If the file can grow, additional activity is required to find free space for a new block, to update the free list or free map, and to update the indirect block to point to the newly allocated space.

    b. This is the same as in part (a) except that no indirect block is involved, so only two data blocks (5 and 6) need to be read in. The global index is normally kept in memory and is not updated unless the file grows, in which case it may be written back to the disk.

39. In a log-structured file system, the location of an i-node changes each time it is updated. The i-node map is needed to record the currently location of the i-node. In a non-log-structured system, i-node locations are static and can be calculated from the i-number.

40. In a write-through cache, updates to cached data are reflected immediately in the backing (secondary) store. In a copy-back cache, they are not.

41. For very large files, almost all data blocks are accessed through the double indirect pointer, so consider only those blocks. For a block size of B and a pointer size of p, the total size of the double-indirect data blocks is

    $$\left(\frac{B}{P}\frac{B}{P}\right)B = \frac{B^3}{P^2}$$

    So, if B is doubled, the maximum file size will increase by approximately a factor of 2^3, or eight.

42. **a.** The worst case is a file that is just large enough to require one byte of the first double-indirect data block. Such a file would have size

    $$10B + \frac{B}{4}B + 1$$

    where the first time accounts for the direct data blocks, the second term for the single indirect data blocks and the third term (1 byte) for the data in the double indirect block. The total internal fragmentation for such a file is

    $$(B - 4) + (B - 4) + (B - 1) = 3B - 9$$

    where the two terms $(B - 4)$ account for space wasted in the two almost-empty double-indirect pointer blocks and the $(B - 1)$ term account for space wasted in the last data block of the file, which holds only one byte of file data.

    b. The file system with the global index will require exactly B disk operations to read the data blocks. The other file system will also require B disk operations to read the data blocks. In addition, another disk operation will be needed to read the single-indirect pointer block, and four additional disk operation will be required to read double-indirect pointer blocks. Thus, a total of five extra disk operations are required.

43. **a.** In this case the file system must do many (64) block writes. The total time required, t_{total} is the sum of the service times for each block write.

$$\begin{array}{rc} t_{seek} & 10 \\ t_{rot} & 5 \\ t_{xfer} & \frac{8}{64}10 = 1.25 \\ t_{service} & 16.25 \\ t_{total} & 64t_{service} = 1040 \end{array}$$

b. In this case, only one (large) disk operation is required to write the log segment to the disk. The time for this is

$$\begin{array}{rc} t_{seek} & 10 \\ t_{rot} & 5 \\ t_{xfer} & \frac{512}{64}10 = 80 \\ t_{service} & 95 \end{array}$$

44. Note that this solution does not include any error checking.

```
int L = num_components(pathname);
int n = 0;
int i;
for i from 1 to L − 1 do {
    n = dsearch(n,get_component(i,pathname);
}
f = ddelete(n,get_component(L,pathname));
remove_link(f);
```

45. Two reasons are:

- a block may be updated more than one time in the cache before being written to disk, if copy-back is used. This would result in multiple I/O's if write-through was used, but only one if copy-back is used.
- a block may be deleted (from its file) after update and before being copied-back to the disk. Copy-back does no I/O, but write-through needed an I/O for the updated block.

46. **a.** A total of six disk reads will be performed: the first three blocks of the file will be read by calls to Read, and the last three will be read by calls to Write. The latter three reads are needed because each Write call only partially updates a file block. If it is assumed that the i-node has very few direct pointers, additional disk read(s) may be needed for indirect blocks.

b. A total of six writes will be performed - each of the last three blocks of the file will be written to the disk twice. Each disk write is caused by one of the six calls to Write made by the program.

c. If the block size is 500, a total of six disk reads are also needed. However, all six disk reads are caused calls to Read. No disk reads are induced by the Write calls, since each Write completely updates a single disk block.

d. If copy-back is used, a total of three disk Writes will be needed. Each of the last three file blocks is written to disk once. One disk write is caused by a Write system call, which forces a previously-updated block from the cache. The remaining two disk writes are caused by the call to Close.