CS 354 Operating Systems

Study Questions

1 Processes and Threads

1. What is an advantage of a large scheduling quantum? What is a disadvantage?

2. An operating system uses a preemptive scheduler with a scheduling quantum of two time units. The running process is preempted only if its quantum expires or it receives its required computing time and exits. Suppose that the following processes must be scheduled:

Proc. Number	Creation Time	Required Computing Time
1	0	4
2	1	6
3	3	4

   What is the response time for each process? What is the average response time?

3. Repeat the problem above assuming a scheduling quantum of 3 time units.

4. Repeat the problem above assuming that the scheduler is non-preemptive and uses the shortest-job-first (SJF) heuristic.

5. Suppose that a scheduler has k ready processes at time 0, and that no new processes are created after time 0. Process i $(0 < i \leq k)$ requires i units of computing time. For a premptive, round-robin scheduler with a scheduling quantum of one time unit, what is the mean response time for these processes? For a non-preemptive, shortest-job-first scheduler, what is the mean response time for these processes? (If you have trouble doing this for arbitrary k, try it for a small fixed value of k, such as $k = 3$.) If one of these schedulers has a lower mean response time than the other, does that mean that it is a better scheduler? Why or why not?

6. List at least three things that a running process might do that would cause the scheduler *not* to move it to the ready state when it stops running.

7. Some operating systems provide support for concurrent processes, which may contain more than one thread. Such an operating system must provide a way for new processes to be created, and a way for new threads to be created within a process. Sketch, in words, what the OS must do to implement each of these operations.

8. Suppose that three concurrent processes exist in a system, as described in the following table:

Process	Threads within the Process
P_1	T_{11}, T_{12}, T_{13}
P_2	T_{21}, T_{22}
P_3	T_{31}

   Suppose that the system uses preemptive, round-robin scheduling, and that T_{11} is running whan the scheduling quantum expires. If the threads are implemented entirely at the user level, with no support from the operating system, indicate which threads might possibly be executing at the beginning of the next quantum. If the threads are supported by the operating system, indicate which threads might possibly be executing at the beginning of the next quantum.

9. This question assumes the same processes and threads as the previous question. Suppose that thread T_{11} finishes executing and terminates. If the threads are implemented entirely at the user level, with no support from the operating system, indicate which threads might possibly run immediatly after T_{11} finishes. If the threads are supported by the operating system, indicate which threads might possibly run immediately after T_{11} finishes.

10. What is the difference between a preemptive scheduler and a non-preemptive scheduler?

11. The following is a list of "events" which may cause the operating system to run:

Label	Event	Description
S	System call exception	exception handler executes because of syscall instruction
E	Other exception	exception handler executes because of page fault, arithmetic error, etc.
T	Timer interrupt	timer interrupt handler executes
I	Other interrupt	non-timer interrupt (e.g. console interrupt) - handler executes

The system uses a preemptive round-robin process scheduler, and supports a set of system calls, devices, and exceptions similar to those supported by the Nachos system. (The Nachos system calls are Exec, Exit, Join, Create, Open, Read, Write, Close, and Halt.) Assume that a process can be in one of four states: **running, ready, blocked, done**. The **done** state indicates that the process has finished execution.

Draw a state transition diagram with one state corresponding to each of the four process states given above. Your diagram should include a directed arrow from one state to another, labeled with one of the events from the list above, if it is *possible* that the event could cause a process to move between those states. For example, you should have an arc from **running** to **done** labeled with "S" because it is possible that a system call will cause a running process to finish executing. (This will occur if the system call is Exit or Halt.)

12. An operating system is running three concurrent processes, P_1, P_2, and P_3 for three different users. Each process requires six units of computing time. Process P_1 consists of a single thread. Process P_2 consists of two threads, each requiring three of P_2's six total units of computing time. Process P_3 consists of three threads, each requiring two of P_3's six total units of time. Assume that the threads never block and that the processes are entirely resident in primary memory.

a. Suppose that the threads are kernel-supported, and that the kernel implements preemptive, round-robin scheduling of threads, with a quantum on one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

b. Suppose that threads are implemented entirely at the user level, and that the kernel implements preemptive round-robin scheduling of processes with a quantum of one time unit. Of the first six units of time, how many will be devoted to each of the three processes?

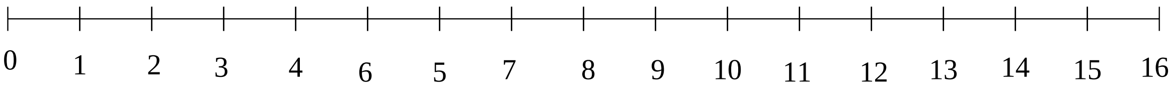
c. Suppose that we define a *fair share* scheduler to be one that devotes $1/n$ of the processor's time to each process when there are n processes in the system. Are the schedulers described in parts (a) and (b) fair share schedulers? If not, suggest how they might be modified to become fair share schedulers. Your suggestion(s) should require no more than a sentence or two.

13. A concurrent process has two threads, T_a and T_b. Each thread is executing the following program fragment:

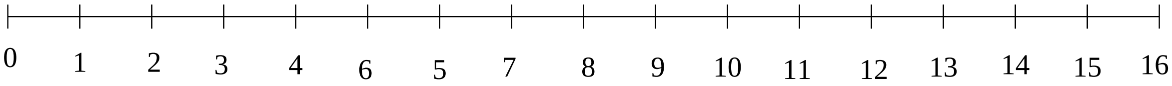
```
char x[1024];
int i;
// assume f is a descriptor referring to a large
// file that has already been opened
read(f,x,1024); // read from the file
compute(x); // call a function that performs some computation using x
read(f,x,1024); // read from the file
compute(x); // call a function that performs some computation using x
thread_done(); // terminate the thread
```

Assume that each call to `read()` causes blocking for three time units (even if several `read()` calls are concurrent) before returning. The function `compute()` requires one time unit to execute, and does nothing that would cause blocking.

a. Assume that threads are kernel-supported. Round-robin scheduling is used, with a quantum of two time units. The scheduler never leaves the processor idle if there are runnable threads. On the following time line, indicate which thread, T_a or T_b, will be executing during each quantum. If no thread is executing during a quantum, indicate this by marking the quantum with "–".



b. Repeat part (a) assuming that threads are not kernel-supported, i.e., that they are implemented entirely at the user level.



14. Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). The CPU's efficiency is the fraction of its time its spends executing user programs, i.e., executing user processes. For round robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following:

    (a) $Q = \infty$

    (b) $Q > S + T$

    (c) $S < Q < S + T$

    (d) $Q = S$

15. Five batch jobs (processes) A through E arrive at a computer center at almost the same time. They have estimated running times of $10, 6, 2, 4,$ and 8 minutes. Their (externally determined) priorities are $3, 5, 2, 1,$ and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

    (a) Non-prioritized round robin

    (b) Non-preemptive prioritized scheduling

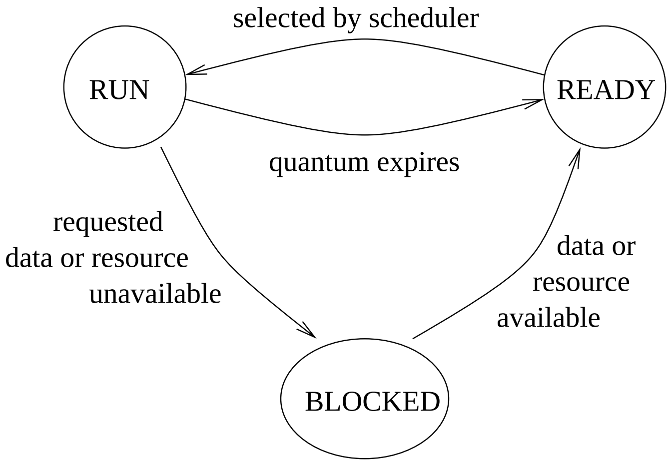
    (c) (Non-preemptive)first-come, first served (run in order A, B, C, D, E)

    (d) Shortest job first (non-preemptive)

    For round robin, assume that the scheduling quantum is much less than the running time of the shortest job. All jobs are completely CPU bound - they do not block.

16. Suppose that an operating system supports two kinds of sequential processes: high-priority interactive processes, and low-priority non-interactive processes. The behaviour of the high-priority processes is to alternate between periods of computation of duration t_c and periods of blocking (waiting for input) of duration t_b. The behaviour of the low-priority processes is to compute constantly, with no blocking. The operating system's scheduling policy is prioritized round-robin with a quantum q, where $t_c < q$. Scheduling decisions are made only when a quantum expires, or when the running process blocks. The scheduler selects a low-priority process to run only if no high-priority processes are ready. Suppose there is one high-priority process and one low-priority process in the system, and that both processes will run for a long time. For what fraction of the time does the low-priority process run?

17. The figure below shows a simple state transition diagram for sequential processes. Suppose that the operating system wishes to implement the two system calls SUSPEND and RESUME. The SUSPEND call can be used by one process to suspend the execution of *another* process. (A process cannot SUSPEND itself.) The suspended process remains suspended until it is RESUMEd by the process that originally suspended it. While a process is suspended it cannot run. Add transitions and/or states to the state transition diagram below to account for the SUSPEND and RESUME system calls. Label any new transitions to indicate what causes the transition to occur, much as the existing transitions are already labeled. Label any new states you add, and briefly describe their purpose.



18. An operating system implements prioritized round robin scheduling. For simplicity, assume that an infinite number of priority levels are allowed. Each priority level is represented by a non-negative integer, with 0 representing the *highest* priority, i.e., if two processes with priorities 0 and 1 are runnable, the scheduler will run the process with priority 0.

    The size of the quantum given to a process when it runs depends on its priority. When a process with priority p is selected to run, it gets a quantum of 2^p time units. Assume the scheduler starts a new quantum of the appropriate length whenever a new process starts to run, and that scheduling decisions are made whenever a process blocks, whenever a process unblocks (becomes ready), or when a quantum expires.

    If a running process with priority p uses its entire quantum without blocking, the scheduler changes its priority to $p + 1$. If instead a running process blocks before the end of its quantum, the scheduler changes its priority to $p - 1$ (unless its priority is already zero, in which case it remains unchanged).

    a. Suppose that there is one process in the system. Initially, it has $p = 0$. The behaviour of this process is to alternate between periods of computation of duration c and periods of blocking of duration b. After this process has been in the system for a long time, which priority (or priorities) could it have?

    b. Suppose there are two types of processes in the system. Type A processes alternate between periods of computation of length c and periods of blocking of length b. Type B processes just compute; they never block. If there is one type A process in the system and many type B processes, what fraction of the CPU's time will be spent running type B processes?

    c. This part assumes the same process types described in part (b). If there is one type B process and many type A processes, what fraction of the CPU's time will be spent running type B processes?

19. A scheduler uses a prioritized round-robin scheduling policy. New processes are assigned an initial quantum of length q. Whenever a process uses its entire quantum without blocking, its new quantum is set to twice its current quantum. If a process blocks before its quantum expires, its new quantum is reset to q. For the purposes of this question, assume that every process requires a finite total amount of CPU time.

    a. Suppose the scheduler gives higher priority to processes that have larger quanta. Is starvation possible in this system, i.e., is it possible that a process will never complete because it is neglected by the scheduler?

    b. Suppose instead that the scheduler gives higher priority to processes that have smaller quanta. Is starvation possible in this system?

20. Suppose that a process contains two threads, a Producer thread and a Consumer thread. The following pseudo-code describes the behavior of the two threads:

Variables Shared by Both Threads

```
data buffer[10]; /* array of buffers to hold produced data */
semaphore sem = 0; /* a counting semaphore to synchronize access to the buffers */
```

Producer Thread	Consumer Thread
FOR i FROM 1 TO 10 DO	FOR i FROM 1 TO 10 DO
read from file into buffer[i]	P(sem)
V(sem)	process contents of buffer[i]
ENDFOR	ENDFOR

a. Suppose that the Producer blocks for a time t_r each time it reads from the file, and the Consumer takes time t_c to process a buffer's worth of data. (Processing the data does not cause the Consumer to block.) The remaining parts of the pseudo-code for the Producer and Consumer require very little CPU time. If the threads are implemented entirely at the user level, what will be the total time required for both threads to run to completion? Your answer should be a single expression involving t_c and/or t_r.

b. Repeat part (a) assuming that the operating system supports concurrent processes and implements both threads directly.

2 Virtual Memory

1. A system implements a paged virtual address space for each process using a one-level page table. The maximum size of an address space is 16 megabytes. The page table for the running process includes the following entries:

page	frame number
0:	4
1:	8
2:	16
3:	17
4:	9

   The page size is 1024 bytes and the maximum physical memory size of the machine is 2 megabytes.

   How many bits are required for each page table entry? What is the maximum number of entries in a page table? How many bits are there in a virtual address? To which physical address will the virtual address 1524 translate to? Which virtual address will translate to physical address 10020?

2. A system implements a segmented virtual memory scheme which provides up to eight paged segments for each process. Each segment is limited in size to half a megabyte (2^{19} bytes). The page size is 4096 bytes. The physical memory size is limited to 2 megabytes. A single base register is used by the MMU. Draw a translation diagram that describes how virtual addresses can be translated in this system. The diagram should show the sizes of the virtual and physical addresses, and how they are broken down into fields. It should show the base register and any tables used for translation. It should show how each of the fields in the physical address is derived from a field(s) in the virtual address and from information in the table(s).

3. Suppose that pages in a virtual address space are referenced in the following order:

   ```
   1 2 1 3 2 1 4 3 1 1 2 4 1 5 6 2 1
   ```

   There are three empty frames available. Assume that paging decisions are made on demand, i.e., when page faults occur. Show the contents of the frames after each memory reference, assuming the LRU replacement policy is used. Repeat assuming that the clock policy is used. How many page faults occur in each case?

4. A system implements a segmented, paged virtual memory for each process using two levels of page tables and a segment table. The MMU includes a TLB. The time to access main memory is T. The TLB can be accessed very quickly (in essentially zero time). The running process needs to read the contents of virtual address v. What must the hit ratio of the TLB be if we wish this operation to take time $2T$, on average?

5. A pure paging system uses a three level page table. Virtual addresses are decomposed into four fields, a, b, c, and d, with d being the offset. In terms of these constants, what is the maximum number of pages in a virtual address space?

6. Distinguish swapping from paging.

7. Distinguish segments from pages.

8. Describe how a context switch affects the virtual memory system. What must the OS do to ensure that memory references made by the newly-running process will be properly translated?

9. Often, the actual amount of virtual memory used by a process is much less than the maximum amount of virtual memory per process that is permitted by the operating system. In this case, the operating system may not wish to maintain page table entries for those portions of the maximum possible address space that are unused. We have discussed at least two ways to limit the number of page table entries required for small processes. What are they?

10. What is an associative memory?

11. The operating system maintains a mapping from page frame numbers to virtual page numbers. Why?

12. What does it mean to coalesce holes?

13. One of the parameters of a virtual memory system is its page size. Give one advantage and one disadvantage of choosing a large page size rather than a small one. Be specific.

14. A virtual address in MULTICS is paritioned into an 18 bit segment number, a 6 bit page number, and a 10 bit offset. Each virtual address identifies a *word* (not a byte) in the virtual address space. What is the page size (in words) in MULTICS, and what is the maximum segment size (in words)?

15. In a system that supports paged virtual memory, two processes are active. The system has 10 physical page frames available to hold pages. They are numbered from 0 to 9. The size of each frame is 1024 bytes. The page tables of the two processes are shown below. Each row of each table shows the contents of a page table entry. Each page table entry includes a frame number and referenced (R), modified (M), and valid (V) bits. An entry with $V = 1$ is a valid entry.

Process 1 Page Table

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

The system uses the global clock algorithm. Replacement decisions are made periodically, rather than on demand. At the end of each period, the clock algorithm scans half of the frames. Any frames selected for replacement during the scan are placed on a free list. When page faults occur, the system chooses pages from the free list for replacement. When modified frames are added to the free list, their pages are not copied back to secondary storage until they are actually replaced by new pages.

a. Assume that the page tables are as shown above when the clock algorithm runs at the end of a period, and that the frame pointer used by the algorithm points at frame zero. (This means that the algorithm will consider frames zero, one, two, three, and four.) Show the page tables of the two processes after the algorithm runs. Indicate which frames will be added to the free list.

b. Immediately after the replacement algorithm runs, process 2 runs and attempts to read virtual address 4100, resulting in a page fault. Show the contents of process 2's page table after the page fault handler has run and the virtual address has been translated successfully by the MMU. Assume that the free list contains those frames that were placed there when the replacement algorithm was run (in part (a) of this question). Indicate which pages if any, will be copied from memory to disk, and which pages, if any will be copied from disk to memory.

16. Suppose that primary memory is managed using a free list to keep track of holes (space that has not been allocated). The best-fit heuristic is used for space allocation. At a certain time, the free list looks as follows:

$$\boxed{100{,}500} \rightarrow \boxed{700{,}50} \rightarrow \boxed{1000{,}200} \rightarrow \boxed{4000{,}700} \rightarrow \boxed{4900{,}100}$$

Each free list entry, represented here by a box, describes a hole. The first number in the box is the starting address of the hole. The second number is the size of the hole, in bytes.

a. Suppose that the following events occur in the order given:

(a) 250 bytes of space starting at address 1200 are freed

(b) 400 bytes of space are allocated

(c) 100 bytes of space starting at address 600 are freed

Show the free list after these events have occurred.

b. For this part of the question, you should start with the free list shown at the top of the page, not with the free list you produced for part (a). Suppose that the following events occur in the order given:

(a) 150 bytes of space starting at address 800 are freed

(b) 125 bytes of space are allocated

(c) 200 bytes of space starting at address 3800 are freed

Show the free list after these events have occurred.

17. A system implements virtual memory using pure paging, with a single page table for each process and one for the operating system. The maximum virtual address space size is 2^{30} bytes, and the page size is 2^{10} bytes. The size of a page table entry is 4 bytes. The system supports up to 2^{24} bytes of physical memory.

    The MMU contains two base registers. Register A contains the *virtual address* of the first entry of the page table of the running process. This is a virtual address in the virtual address space of the operating system. Each process' page table is contiguous in the virtual address space of the operating system, but is not necessarily contiguous in the physical address space. Each page table starts on a page boundary in the operating system's virtual space.

    Register B contains the *physical address* of the first entry of the page table of the operating system's virtual address space. This page table is contiguous in the physical address space.

    a. How many bits are required for each of the MMU's base registers? What is the maximum size (in bytes) of a page table?

    b. Draw a translation diagram that shows how virtual addresses can be translated in this system.

18. A *cache* is a relatively small, relatively fast memory that is used to hold a partial copy of the contents of a larger, slower memory. Give two examples of caching found in virtual memory systems. For each example, indicate what is the small fast memory, and what is the large slow memory.

19. UNIX operating systems consider the virtual address space of each process to consist of three segments. What is contained in each of these segments?

20. A paged virtual memory system uses a page size of 1024 (2^{10}) bytes. Page table entries are 4 bytes long. The maximum virtual address space size of each process is 2^{30} bytes. The size of the physical address space is also 2^{30}. What is the minimum number of levels of page tables needed in this system to ensure that each page table will fit entirely within a single frame? Draw a translation diagram that shows how virtual address translation can be performed in this system. Be sure to indicate the sizes of the registers, addresses, and page tables in your diagram. Indicate how each address is broken down into components, and indicate the size of each component.

21. Suppose that a virtual memory system uses the CLOCK replacement algorithm, which is applied on demand. The system has four physical frames (numbered zero through three), and the frame size is 100 bytes. The CLOCK algorithm's pointer points at frame zero. Only one process is running in the system. Its page table is as follows (the protection bits are not shown):

Page Number	Frame Number	Valid	Use	Dirty
0	3	1	1	0
1	1	1	0	0
2	0	1	1	1
3	0	0	0	0
4	2	1	0	1

    a. Give a sequence of three virtual addresses such that if these are the next three addresses referenced (used) by the process, each reference will cause a page fault.

    b. For this part, use the page table shown above (ignore any changes that would have been made by the memory references you have in part (a)). Give a sequence of three virtual addresses such that if these are the next three addresses referenced by the process, none of the references will cause a page fault.

22. The following is the core map of a virtual memory system which has a page size of 100.

Frame Number	Process ID	Page Number
0	1	2
1	1	1
2	2	1
3	3	0
4	1	3

a. To which physical address does virtual address 130 of process 1 map? If this virtual address does not map to any physical address, write "does not map".

b. To which physical address does virtual address 17 of process 2 map? If this virtual address does not map to any physical address, write "does not map".

c. Which virtual address of which process maps to physical address 50?

23. A user process executes a program that begins as follows:

```
int fd; // file descriptor
char buf[4]; // buffer to hold data from the file
fd = Open("foo");
Read(fd,buf,4); // read 4 bytes from "foo" into buf
```

When the operating system handles the exception caused by the `Read` system call above, it receives the parameter values 3, 998, and 4, corresponding to the parameters `fd`, `buf` (the address of the buffer), and `4`. Assume that the first four bytes of the file "foo" are the characters a,b,c,d (in that order). The virtual memory system uses a page size of 1000 bytes. At the time of the `Read` call, the page table for the process looks like this:

Page Number	Frame Number	Valid	Use	Dirty
0	2	1	1	0
1	0	1	0	0
2	3	1	1	1
3	4	0	0	0
4	7	1	0	1

a. Assume that that operating system runs in physical memory, i.e., it does not have its own virtual address space. To which four physical addresses will the operating system write the characters a, b, c, and d so that they will be placed properly in the process' buffer when the `Read` system call returns?

b. Suppose instead that the operating system has its own virtual address space. When the operating system is ready to pass the characters a, b, c, and d to the process, its (the operating system's) page table looks like this:

Page Number	Frame Number	Valid	Use	Dirty
0	6	1	1	0
1	2	1	1	1
2	9	1	1	1
3	0	1	0	1
4	1	1	0	1

To which four operating system virtual addresses will the operating system write the characters a, b, c, d so that they will be placed properly in the process' buffer when the `Read` call returns?

24. Briefly sketch the address translation process of a system that utilizes a combination of segmentation and paging. Your diagram should include all relevant components (such a segment table, page table, virtual address, resulting physical address, etc.).

25. A computer whose processes have 1024 pages in their address spaces keeps its page tables in memory. The overhead required for reading a word from the page table is 500 nsec. To reduce this overhead, the computer has an associative memory, which holds 32 (virtual page, physical page frame) pairs, and can do a lookup in 100 nsec. What hit rate is needed to reduce the mean overhead to 200 nsec?

26. Is the "aging" algorithm (for making page replacement decisions) a stack algorithm? Why or why not?

27. Can a true Least-Recently-Used (LRU) page replacement policy be implemented (in software) in a virtual memory subsystem? Why or why not?

28. How do local page replacement policies differ from global ones?

29. A virtual memory system is to be implemented on memory management hardware that does not support a reference (R) bit in page table entries. In other words, the hardware does not automatically set an R bit each time a page is referenced. The only page table entry bits supported by the hardware are a dirty (D) bit and a valid (V) bit.

    Show how an R bit can be simulated in software by the virtual memory system, using no hardware facilities other than the ones described above. This R bit should appear to behave just like a hardware-supported R bit would. To simulate the R bit, the software may define and use any page table entry bits it wishes to, but such bits will not be used by the hardware.

30. A virtual memory system uses both paging and segmentation. Process virtual addresses consist of a total of 32 bits. The page size is 1024 (2^{10}) bytes. A page table may occupy at most one frame, and the size of a page table entry is 4 bytes. A single process may have up to 64 (2^6) segments.

    a. How many levels of page tables are required for virtual address translation in this system?
    b. What is the maximum size of a virtual memory segment?
    c. Draw an address translation diagram that shows how a virtual address is translated in this system. Your diagram must show each component of the virtual and physical addresses, and the size of each componet. It must show any tables used in translation, and their lengths. It must show how the components of the virtual address are used to index the tables, and how the components of the physical address are determined.

31. Sketch a graph showing page fault rate for a process as a function of the number of page frames assigned to it. Assume that the working set size of the process is W pages, and be sure to label this point in your graph. Also assume that the virtual memory system's page replacement algorithm is a stack algorithm.

32. On a MIPS R2000 (the hardware simulated in Nachos), what happens if a virtual address cannot be translated using the contents of the TLB?

33. In UNIX systems, the new address space of a newly-created process is initialized so that it is an exact copy of the address space of its parent. A simple way to implement this is to make a copy of every page in the parent process' address space when a child process is created. If the address space is large, this can take a long time.

    An alternative implementation is to allow the child process to share the parent's copy of a page until either the parent or the child attempts to update the page. At that point, a separate copy of the page is made for the child, and only one of the two copies of the page (depending on which process is doing the update) is actually updated. Under this scheme, called *copy-on-update*, pages that are never updated by either process (such as code pages) remain shared by the two processes. Note that both the parent and the child have their own page tables.

    Suppose that page table entries contain the following bits, in addition to a frame number:

    - valid bit (V)- causes a page fault exception to occur if an attempt is made to access an invalid page
    - modify bit (M)- set by the hardware when a page is updated
    - read-only bit (P) - if set, causes a protection fault exception to occur if an attempt is made to update a read-only page

- use bit (U) - set by the hardware when a page is read or updated

Describe how the operating system can implement *copy-on-update* assuming that the page table entries are as described. Be specific - indicate what the operating system must do, and when it must do it.

34. Suppose that a particular program goes through two execution phases. During the first phase, which lasts for t_1 time units, the program makes frequent and heavy use of p_1 pages in its address space. During the second phase, which begins immediately after the first, the program makes frequent and heavy use of p_2 of the pages from its address space. These pages are distinct from those used during the first phase.

    Let $|WS(t,\Delta)|$ represent the size of the program's working set at time t, for a window size of Δ. In other words $|WS(t,\Delta)|$ represents the number of distinct pages referenced during the interval from $t-\Delta$ to t. Sketch a plot of $|WS(t,\Delta)|$ versus t. Assume that $0 < \Delta < t_1$, and that the number of pages referenced during a window of length Δ is much larger than p_1 or p_2. Be sure to mark the axes to indicate the location(s) of any "interesting" points in your sketch.

35. Consider the following page reference string:

    1 2 1 3 2 1 1 4 3

    Give the corresponding distance string, assuming that the page replacement policy is LRU.

36. A small virtual memory system has only two frames of physical memory, and uses demand paging. Both frames are initially empty. Consider the following page reference string:

    1 2 3 2 3 1 3 1 2 1

    Give a lower bound on the number of page faults that will result from this reference string.

37. A virtual memory system supports 12 bit virtual addresses. It uses pure segmentation with a maximum segment size of 1024 (2^{10}) bytes. Suppose that the segment table for the currently running process looks as follows:

seg. num.	V	P	start	len
0	1	1	0	200
1	1	0	1000	160
2	0	0	0	0
3	1	0	200	300

In the table, V and P are the segment "valid" and "protection" bits, "start" represents the physical address of the start of the segment and "len" is the segment's length. $P = 1$ indicates that a segment is read-only.

Consider the following read and write operations. If the specified operation would succeed given the segment table shown above, give the physical address to which the specified virtual address would translate. If it would not succeed, state the reason that it would fail.

a. A write to virtual address 150.

b. A read from virtual address 1025.

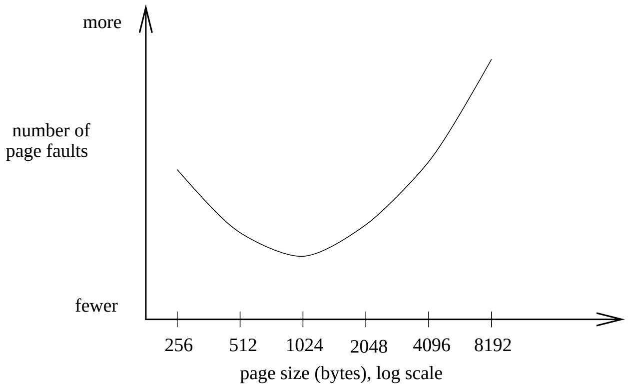
c. A write to virtual address 4000.

38. A virtual memory system uses a two-level paging scheme for address translation. Virtual addresses are 32 bits long, and the page size is 4096 (2^{12}) bytes. Each page table occupies a full frame, and each page table entry is 4 bytes long. The maximum physical memory size is 1 megabyte (2^{20} bytes).

    a. Draw an address translation diagram showing how addresses are translated in this system. The diagram should show the sizes of the virtual and physical addresses, how they are broken down into fields, and how each field if the physical address is derived. It should show the base register, any tables used for translation, and the maximum sizes of those tables.

b. Suppose that a process' virtual address space is structured as in UNIX, with program code at the "bottom" of the address space, program data (the heap) immediately following the code, and the stack at the "top". Suppose that the program code occupies 2000 pages, the data occupies 1000 pages, and the stack occupies 100 pages in the address space - the remainder of the address space is unused. How many frames will be occupied by the page table(s) for this process?

39. Suppose that a series of experiments is run to measure the number of page faults incurred by a test program as a function of the page size used by a demand paging virtual memory system. For each page size, the test program is run alone in the system and the total number of page faults that occur is measured. The results of the experiments are shown in the graph below.



a. Explain, in two or three sentences, the shape of the curve in the graph. Why is the number of page faults higher when the page size is very large or very small? Your explanation should involve the working set of the test program.

b. Suppose that the experiment described above had measured the total execution time of the test program rather than the number of page faults. Sketch the total execution time as a function of the page size, assuming the number of page faults varies as shown in part (a). State explicitly any assumptions that you make.

40. A memory management unit supports three-level paging, and has a hardware-loaded TLB. What are the minimum and maximum numbers of main memory accesses required by this MMU to translate a virtual address to a physical address, assuming that the translation does not cause a page fault?

41. Is the "clock" replacement algorithm a stack algorithm? Why or why not?

42. Suppose that an operating system supports paged virtual address spaces with a page size of 1024 bytes. A process running in this system executes the following code:

```
char buf[1500]; // a buffer to hold file data
int d; // file descriptor
d = Open("experiment.dat"); // open a file, returning a descriptor
Read(d,buf,1500); // fill the buffer with data from the file
```

a. What is the maximum number of pages of the process' address space that will be examined (read, but not changed) by the operating system as a result of the execution of the code above? Assume that system call parameters and return values are passed as in Nachos.

b. What is the maximum number of pages of the process' address space that will be updated (changed) by the operating system as a result of the execution of the code above?

c. At the time of the Read system call the process' page table looks as follows:

Page Number	Frame Number	Valid
0	0	1
1	3	1
2	1	1
3	4	1
4	0	0
5	2	1

Suppose that the virtual address of the start of "buf" is 2100. At which physical addresses is the buffer located? (Indicate where the entire buffer is located, not just the start of the buffer.)

3 Disks and Other Devices

1. Consider the following parameters describing a disk:

parameter	description
C	number of cylinders
T	number of tracks per cylinder (number of platters)
S	number of sectors per track
ω	rotational velocity (rotations per second)
B	number of bytes per sector

   In terms of these parameters, how many bytes of data are on each disk cylinder? Suppose that you are designing a disk drive, and that you hope to reduce the expected rotational latency for requests from the disk. Which of the parameters above would you attempt to change, and in what way would you change them? Suppose you wanted to reduce the disk's data transfer time - which parameters would you attempt to change?

2. Suppose that $C = 2000$, $T = 10$, $S = 50$, $B = 512$, and $\omega = 60$. Suppose further that seek times (t_s) (in milliseconds) are given by

$$t_s = 3 + 0.025d$$

   where d is the seek distance, in cylinders. The read/write heads are positioned over cylinder 20. The disk controller receives a request for device block 596. (Suppose that the disk sectors are numbered so that all sectors on cylinder i have lower numbers than all sectors on cylinder j, if $i < j$.) What is the expected time for the controller to service this request? What is the worst case time? What is the best case time?

3. Suppose that the disk request queue contains requests for sectors on the following cylinders: 400, 20, 19, 74, 899. Suppose that the current position of the disk arm is over cylinder 200. In which order will the requests be handled under the SSTF disk head scheduling policy? Under the SCAN policy? Under the C-SCAN policy?

4. What is the difference between a block-oriented device and a character-oriented device?

5. A disk drive has C cylinders, T tracks per cylinder, S sectors per track, and B bytes per sector. The rotational velocity of the platters is ω rotations per second.

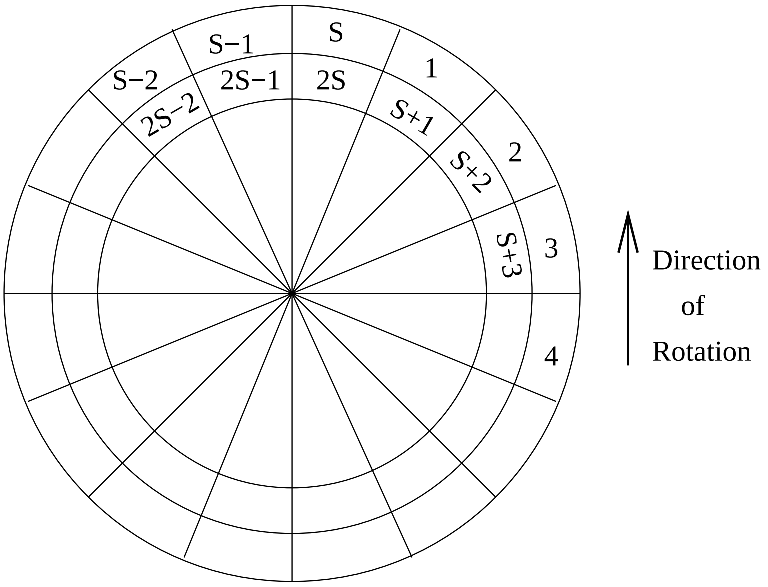
   a. Consider s_1 and s_2, consecutive sectors on the same track of the disk. (Sector s_2 will pass under a read/write head immediately after s_1.) A read request for s_1 arrives at the disk and is serviced. Exactly d seconds ($0 < d < 1/\omega$) after that request is completed by the disk, a read request for sector s_2 arrives at the disk. (There are no intervening requests.) How long will it take the disk to service the request for sector s_2?

   b. Suppose that s_1 and s_2 are not laid out consecutively on the disk. Instead, there are k sectors between s_1 and s_2. Does this change your answer from part (a)? If so, for which value(s) of k will the time to service the read request for s_2 be minimized? (Only consider $0 \leq k \leq S - 2$.)

6. A disk drive has S sectors per track and C cylinders. For simplicity, we will assume that the disk has only one, single-sided platter, i.e., the number of tracks per cylinder is one. The platter spins at ω rotations per millisecond. The following function gives the relationship between seek distance d, in cylinders, and seek time, t_{seek}, in milliseconds:

$$\begin{array}{ll} t_{seek} = 0 & d = 0 \\ t_{seek} = 5 + 0.05d & 0 < d \leq C \end{array}$$

   The sectors are laid out and numbered sequentially, starting with the outer cylinder, as shown in the diagram below.



a. Suppose the disk read/write head is located over cylinder 10. The disk receives a request to read sector S. What is the expected service time for this request?

b. Exactly d milliseconds after completing the request for S, the disk receives a request for sector $S+1$. What is the expected service time for this request?

7. In one or two sentences, define DMA (direct memory access). How is the processor made aware that a DMA has been completed?

8. A disk drive has $T = 1000$ tracks per surface and $S = 10$ sectors per track. The platters spin at a rate of $\omega = 100$ rotations per second. The following function relates seek distance, d, in cylinders, to the seek time (in milliseconds):

$$t_{seek} = 0.1d + 5$$

Sectors s_1 and s_2 are consecutive sectors on the same track of cylinder 100. (s_2 will pass under the read/write head immediately after s_1 does.)

a. The read/write heads are initially located over cylinder zero. The disk receives a request for sector s_1. After servicing that request, it is idle for a time, and then receives a request for sector s_2. What is the sum of the expected service times for these two requests?

b. The read/write heads are initially located over cylinder zero. The disk receives a single request to read sectors s_1 and s_2. What is the expected service time for this request?

9. How long does it take to load a 64K program for a disk whose average seek time is 30 msec, whose rotation time is 20 msec, and whose tracks hold 32K

(a) for a 2K page size?

(b) for a 4K page size?

Assume that the pages are spread randomly around the disk and that each page is stored in consecutive sectors on the same track, with each sector being 512 bytes in size.

10. Disk requests come in to the disk drive for tracks 10, 22, 20, 2, 40, 6, and 38, in that order. A seek takes 5 msec per track moved. How much seek time is needed for

(a) First-come, first served

(b) Closest track next

(c) Elevator algorithm (initially moving upwards)

In all cases, the arm is initially at track 20.

11. Interleaving is a technique for reducing disk service times. What is interleaving? Which component or components of the service time does it attempt to reduce?

12. What is video RAM, and what is its purpose?

13. What is the difference between a device driver and a device controller?

14. A disk manufacturer is considering two options for increasing the capacity of its disks. The first is to increase the number of sectors per track on the disk, while keeping the number of bytes per sector and the number of tracks per surface constant. The second is to increase the number of tracks per surface, while keeping constant the numbers of bytes per sector and sectors per track. What effect will these changes have the time required by the disk to service requests? Be specific. For each of the two possible changes, indicate which components of the disk's service time will increase or decrease (relative to the original disk).

15. In Nachos, the disk and the console are asynchronous devices. What does it mean for a device to be asynchronous?

16. A disk spins at 100 rotations per second and has 25 sectors per track. Its seek times, in milliseconds, are given by the formula
$$t_{seek} = 5 + \frac{d}{10}$$
where d is the seek distance measured in cylinders. The disk's read/write heads are located over cylinder 0.

    a. Suppose that the disk receives a request to read 5 sectors from a track in cylinder 100. After an arbitrary delay (during which no other requests are received), the disk receives a second request to read 5 sectors from the same track that the first request read from. What is the sum of the service times of these two requests?

    b. Suppose that the disk controller has a *track buffer*, i.e, sufficient memory to cache the contents of one track from the disk. The controller behaves as follows. If it receives a request for which the requested data is in the cache it services the request using the cached data. This takes zero time. Otherwise, it seeks to the appropriate cylinder and transfers to the track buffer the entire track that includes the requested data. (This overwrites whatever was already in the buffer.) This transfer to the track buffer begins as soon as the seek is complete - there is no rotational latency because an entire track is being transferred to the buffer. Once the track is in the buffer, the request is serviced from the buffer. This last step takes no time. Assuming that the read/write heads are initially at cylinder 0, what is the sum of the service times for the two requests described in part (a).

17. How does the structure of a serial terminal interface (e.g., RS-232) differ from the structure of a memory-mapped terminal interface?

18. For each part of this question, assume that the disk has a total of 100 cylinders.

    a. Suppose that the disk head scheduling policy is shortest seek time first (SSTF). The disk heads are currently positioned over cylinder 42 and requests are queue for data on cylinders

    27, 40, 90, 37

    What will be the total distance (in cylinders) traveled by the disk heads to service the four queued requests?

    b. Suppose instead that the disk head scheduling policy is SCAN (elevator). The disk heads are initially positioned over cylinder 42 and the current direction of travel of the heads is "up", i.e., towards higher-numbered cylinders. For the same set of queued requests given in part (a), what will be the total distance (in cylinders) traveled by the disk heads to service the requests?

c. For this part, assume that the contents of the disk queue are unknown and that new requests may be made at any time. Under each of the two head scheduling policies (SCAN and SSTF), what is the maximum (worst case) distance that the disk arm may travel between the time a new request is made and the time that request is serviced?

19. Disk head scheduling, interleaving, and track buffering (in the disk controller) are three techniques for improving the performance of disks.

    a. Which, if any, of these three techniques can reduce (or eliminate) seek times? Explain briefly.

    b. Which, if any, of these three techniques can reduce (or eliminate) rotational latency? Explain briefly.

    c. Which, if any, of these three techniques can reduce (or eliminate) transfer times? Explain briefly.

4 File Systems

1. What is the purpose of the per-process descriptor tables in UNIX file systems?

2. File systems implement files using an underlying storage device. Specify two distinct features of file systems that make access to data through the file system preferable to direct access to the underlying storage device. An example of a feature is that files may be assigned user-selected names.

3. Consider a UNIX-style i-node with 10 direct pointers, one single-indirect pointer, and one double-indirect pointer. Assume that the block size is 8 Kbytes, and that the size of a pointer is 4 bytes. How large a file can be indexed using such an i-node?

4. Why do file systems provide "open" and "close" operations? For UNIX file systems, describe the work that must be performed by the file system when an open() call is made. In particular, describe what changes are made to the file system's in-memory data structures.

5. In UNIX file systems, each i-node includes a reference count. Why?

6. What advantages does a per-file indexed file layout scheme have over chained (linked list) layout? What disadvantages?

7. One task of a file system is to keep track of free data blocks on the device. Describe at least two different techniques for doing this, and list the advantages of each.

8. What is "fragmentation"?

9. In UNIX file systems, hard links cannot span file systems. Why not?

10. Write a procedure to translate a pathname into an i-number in a UNIX-like hierarchical file system. Your procedure should take a pathname (of type string) as its only parameter. It should return the i-number (of type integer) of the file specified by the input pathname. If any error is encountered during translation, your function should return the special code *INVALID* instead of a real i-number. Your procedure may call the following functions:

    `boolean is_regular(int i-number)` This function returns the value "TRUE" if the file whose i-number is given is a regular file. It returns "FALSE" otherwise.

    `int num_components(string pathname)` This function returns the number of components in the specified pathname. For example, num_components(*"/a/b/c"*) returns 3, num_components(*"/foo"*) returns 1, and num_components(*"/"*) returns 0.

    `string get_component(int i, string pathname)` This function returns a string representing the ith component of the specified pathname. The number "i" should be a positive integer. For example, get_component(*2, "/a/b/c"*) will return the string "b", and get_component(*1, "/foo/bar"*) will return "foo". If there is no ith component, the function returns the empty string "".

    `int dsearch(int i-number, string component)` This function is used to search a directory file for an element whose name matches the "component" argument. The parameter "i-number" must be the i-number of the directory file to be searched. The function returns the i-number of the matching entry, For example, the call dsearch(*10, "foo"*) will return the i-number of the "foo" entry in the directory file whose i-number is 10. If there is no such entry in the specified directory, the function returns the value -1.

    Your code may treat strings like primitive data types. Assume that the i-number of the root directory is 0.

11. Modify the code from the previous question so that symbolic links are handled properly. Your code should report an error if more than eight symbolic links are encountered during a pathname translation. For this question, you may use the following functions, in addition to the ones shown above.

    `int is_symlink(int d)` This function returns the value "TRUE" if the file whose i-number is given is a symbolic link file. It returns "FALSE" otherwise.

`string get_symlink(int d)` This function returns the symbolic link (a pathname) from the symbolic link file whose i-number is specified.

12. List at least four different types of information that can be found in a UNIX i-node.

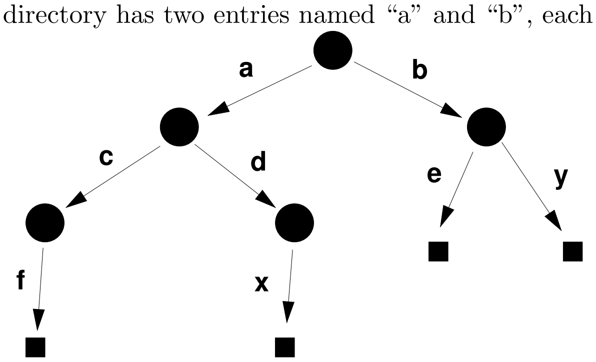
13. Suppose that a UNIX file system uses a block size of 1K bytes (i.e., 1024 bytes), which is the same as the disk block size. A process has opened a file, receiving file descriptor 3 as a return value. The total length of the file is 24K bytes. The process then performs the following two file system calls on the open file:

```
lseek(3,1500)          #seek to location 1500 in the file
read(3,buffer,1024) #read 1024 bytes of data from the file into the buffer
```

    In the worst case, how many disk operations will the file system perform to implement these two file system calls? (Each disk operation retrieves or updates one file system block.) For each disk operation, specify which of the two calls causes the operation to occur, and what the purpose of the disk operation is (i.e., what data it stores or retrieves).

14. Suppose that the same process, after performing the two operations above, wants to modify (change) the first 1024 bytes in the file. To do this, it will read the file's first 1024 bytes into a buffer in its memory, modify the buffer, and then write the modified buffer back to the beginning of the file. Give a sequence of file system commands that will do this.

15. Suppose that a UNIX file system is as shown in the figure below. In the figure, directories are represented as circles, regular files as squares. A directory's entries are represented by the labeled arrows. For example, the figure shows that the root directory has two entries named "a" and "b", each of which is a subdirectory.



    Consider each of the following commands (separately, not as a sequence). Will they result in error, or will they be performed successfully? If the command will result in an error, what is the reason for the error?

    - symlink(/b/y , /a/d/x)
    - link(/b/y , /a/d/x)
    - symlink(/c , /b/z)
    - link(/c , /b/z)
    - symlink(/a/c/f , /b/f)
    - link(/a/c/f , /b/f)
    - symlink(/a/c , /b/g)
    - link(/a/c , /b/g)

16. Why is a hierarchical file name space more useful than a flat name space? What additional advantages are there if the name space is a rooted directed acyclic graph?

17. In a UNIX file system, suppose that "/a/b/c" is a hard link to the file "/d/e/f". List, in order of search, the pathnames of the directories that must be searched by the file system when a process requests open("/a/b/c").

18. Suppose instead that "/a/b/c" is a symbolic (soft) link to "/d/e/f". Repeat part (d), i.e., list, in order of search, the pathnames of the directories that must be searched by the file system when a process requests open("/a/b/c").

19. Assume that a file F has been opened, and that F is 200,000 bytes long. Assume that the block size of the file system is 1,000 bytes, and that the size of a block pointer is 10 bytes. Ignore the effects of caching. Assume that the file system uses UNIX-like index structures (i-nodes), each having ten direct data block pointers, one single indirect pointer, and one double indirect pointer. Assume that the i-node for F is in memory but none of *F's* data blocks are. The process that opened F issues a `read()` request for bytes 9500-10300 from the file. How many blocks must the file system retrieve from the disk to satisfy this request? Justify your answer.

20. This question refers to the same file system described in the previous question. Assume that F has been opened and that its i-node is in memory, but none of *F's* data blocks are. The process that opened F issues a request to read the entire file (200,000 bytes) into memory. How many blocks must the file system retrieve from the disk to satisfy this request? Justify your answer.

21. This question refers to the same file system described in the previous question. Assume that the file system allocates only as many data blocks and indirect blocks as are necessary to accommodate the size of a file. A file G is L_G bytes long. Suppose that a process opens G and appends one byte to it, i.e., it makes G one byte longer than it was. What is the maximum, over all possible values of L_G, number of blocks that must be modified by the file system to implement the append operation? Justify your answer, and give an example of a value of L_G for which the maximum number of blocks will have to be modified.

22. This question assumes a UNIX-like file system. Write a procedure to implement the `rename` operation in this file system. This operation takes two arguments of type `string`: the existing pathname (`oldpathname`) and the desired new pathname (`newpathname`). The rename operation should simply change the old name of the file to the new name. It should not cause any new files or directories to be created. It should return an error code in case any problems are encountered, e.g., in case `oldpathname` does not exist or `newpathname` does exist. An example is `rename(/a/b,/b/c)`, which should change the name of file "/a/b" to "/b/c". For this call to succeed, "/a/b" must already exist, and "/b" must exist, and "/b/c" must not already exist.

    Assume that the i-number of the root directory is zero, and do not worry about symbolic links for the purposes of this question. You may use the functions `is_regular`, `num_components`, `get_component`, and `dsearch`, described previously. You may also using the following two functions:

    `int ddelete(int d, string path_component)` If `path_component` appears in directory file `d`, its entry is deleted from the directory. The function returns the i-number from the deleted entry. If `path_component` cannot be found in the directory, a negative value is returned.

    `int dinsert(int d, string path_component, int i_num)` If `path_component` does not match any of the entries in directory `d`, a new entry is inserted. The arguments `path_component` and `i_num` are used to create the new entry. The function returns a positive value if the insert succeeds. It returns a negative value if `path_component` matches an existing directory entry.

23. A file system that lays out data in blocks must specify the block size that it will use. Name one disadvantage and one advantage of choosing a large block size.

24. In some file systems, each directory entry contains file attributes in addition to a pathname component. Compare this scheme to the UNIX scheme in which file attributes are stored in a separate data structure (the i-node). Give one advantage and one disadvantage of storing attributes in directories.

25. A block-oriented file system has a block size of B bytes. Give an expression for the amount of space wasted due to internal fragmentation when a file of size F bytes is stored in the file system.

26. This question assumes a UNIX-like file system with a hierarchical directory structure. The file system supports a `link` operation:

    ```
    link(string oldname, string newname)
    ```

The `link` operation is identical to a UNIX hard link. The `oldname` argument should be the pathname of an existing file, and the `newname` argument is the name of the new link that is to be created. For example, `link("/a/b","/c/d/e")` should make `"/c/d/e"` a link to the file `"/a/b"`. In case of error, the `link` operation should return `INVALID`, otherwise it should return `OK`.

Implement `link` in C-like pseudo-code.

Assume that the i-number of the root directory is zero. Do not consider symbolic links for the purposes of this question. You may use the functions `is_regular, dsearch, ddelete, dinsert, num_components` and `get_component` defined in previous questions.

27. Consider a file system that maintains an unique index node for each file in the system. Each index node includes 8 direct pointers, a single indirect pointer, and a double indirect pointer. The file system block size is 1024 (2^{10}) bytes, and a block pointer occupies 4 bytes. What is the maximum file size that can be supported by this index node?

28. For the file system described in the previous question, how many disk operations will be required if a process reads data from the Nth block of a file? Assume that the file is already open, the buffer cache is empty, and each disk operation reads a single file block. Your answer should be given in terms of N.

29. Suppose that the file system referred to in the previous two questions is modified so that the index node does not contain a double indirect pointer. Instead, if a file is larger than can be represented using the direct and single indirect pointers, the last pointer in the indirect block is used to point to another indirect block containing more pointers to data blocks. If that second indirect block fills up, its last pointer is used to point to another indirect block. This chain of indirect blocks can grow as long as is necessary to accommodate a large file. Under this new indexing scheme, how many disk operations will the file system have to perform if a process reads data from the Nth block of a file? Use the same assumptions as were used in the previous question.

30. A UNIX i-node contains a number of fields, including a single indirect pointer, the file size, the file type, access permissions, and a reference count. Assume that `"/a/b"` is the pathname of an existing file in the file system, and that `"/c"` is not.

    Fill in each of the entries in the matrix below to indicate how each of the specified file system calls uses each of the fields of the inode for file `"/a/b"`. In each cell of the matrix, write one of the following three codes:

    "-" to indicate that that the call does not need to read or change the field value

    "R" to indicate that the call may need to check (read) the field value, but will not need to change it

    "W" to indicate that the call may need to change the field value (and may read it as well).

    For the `read` and `write` calls, assume that `d` is a file descriptor returned by a previous call to `open("/a/b")`.

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

31. A file system uses a global index scheme (like the File Allocation Table in DOS) to locate file data. Secondary storage is a disk with 1024 (2^{10}) sectors of 128 (2^7) bytes each. The file system uses a block size of 4 sectors. A block pointer occupies two bytes. How many disk sectors are required to hold the global index?

32. Give a pseudo-code implementation of a function `showpath(i)`, which prints the full pathname of the directory whose i-number is `i`. Assume that the file system is UNIX-like, with a hierarchical directory structure. Every

directory contains a special entry (``..'',parent-i-number), where parent-i-number is the i-number of that directory's parent in the hierarchy. The i-number of the root directory is zero. The root directory is considered to be its own parent, i.e., the root directory contains the entry (``..'',0).

You may treat strings as primitive data types, and you may use any reasonable string-related functions in your pseudo-code (don't implement the string functions). For example, you may use `print(s)` to print a string `s`, or `concatenate(s1,s2)` which concatenates `s1` and `s2` and returns the resulting string. In addition, you may use either or both of the following functions in your pseudo-code:

`int dsearchByName(int dirInumber, string component)` This function is used to search a directory file for an entry whose name matches the "component" argument. The parameter "dirInumber" must be the i-number of the directory file to be searched. The function returns the i-number of the matching entry, For example, the call dsearchByName(*10, "foo"*) will return the i-number of the "foo" entry in the directory file whose i-number is 10. If there is no such entry in the specified directory, the function returns the value -1.

`string dsearchByNumber(int dirInumber, int i)` This function is used to search a directory file for an entry whose i-number matches the "i" argument. The parameter "dirInumber" must be the i-number of the directory file to be searched. The function returns the pathname component of the matching entry For example, the call dsearchByNumber(*10, 7*) will return the string "foo" if there is an entry (``foo'',7) in the the directory file whose i-number is 10. If there is no such entry in the specified directory, the function returns an empty string.

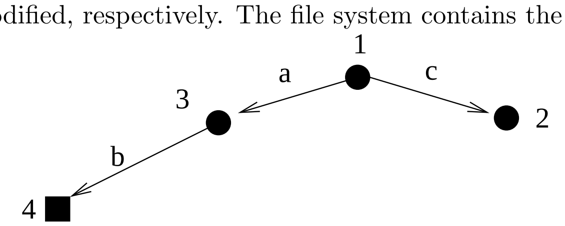
33. A file system uses i-nodes containing nine direct pointers, one indirect pointer, and one double-indirect pointer. The block size is 1000 bytes. The file system has a buffer cache that uses a write-through update policy. The buffer cache hold 10 blocks. It is initially empty. Consider the following program fragment executed by a user process:

```
char x[1800];
// assume f is a descriptor referring to a very
// large file that has already been opened
seek(f,10100);
write(f,x,1000); // write 1000 bytes to the file
read(f,x,1800); // read 1800 bytes from the file
```

For each of the three file system calls in this program fragment, indicated how many block read and block write operations the file system will perform to implement that call. Justify your answer.

34. Consider a Nachos-like file system in which every file has a file header, and in which directories and the file system's free space bitmap are themselves stored as files. Suppose that the file system caches file headers and file data, and that the cache is empty except for the file headers for the root directory and for the free space bitmap's file. The file system's cache uses a write-through update policy. The root directory is initially empty. A user program requests that a new file, initially of zero length, be created in the root directory. How many disk reads and disk writes are required to create this file?

35. A UNIX-like file system stores file attributes in per-file indices (i-nodes). These attributes include a link count, the file size, a read timestamp and a write timestamp. The read and write timestamps are updated each time the file's data is read or modified, respectively. The file system contains the following files



where circles indicate directories and squares indicate regular files. The index number (i-number) of each file is written next to the file's square or circle. A user program requests that a hard link `/c/d` to the file `/a/b` be added to the file system. For each of the four attributes listed above, give the i-numbers of those files (regular or directory) for which the value of that attribute might change as a result of this request.

36. Assume that the time required for a single disk read or write operation (in milliseconds) is $50 + n$, where n is the number of kilobytes of data read or written. A 10 kilobyte file is stored in a block-oriented file system with a 4 kilobyte block size. Assuming the file's data has already been located in the disk, how long will it take to read it all into memory?

37. Suppose that a 10 kilobyte file is stored in an extent-based file system that was almost empty when the file was created. Under the same assumptions used for the previous question, how long will it take to read this entire file into memory?

38. A user program executes a program which contains the following code:

    d = Open("filename"); // open a file
    Seek(d,5900); // move the file pointer
    Write(d,buf,200); // write 200 bytes of data to the file

    a. Assume that the file system uses Unix-like i-nodes with six direct pointers, one single-indirect pointer, and one double-indirect pointer. The file system is block oriented with a block size of 1000 bytes. It has a large cache which uses a copy-back update policy. How many disk read and write operations are required to implement the last two system calls (`Seek` and `Write`) in the program fragment above?

    b. Assume instead that the file system uses a global index. Otherwise, it is the same as the file system described in part (a). How many disk read and write operations are required to implement the last two system calls (`Seek` and `Write`) in the program fragment above?

39. Log-structured file systems have a data structure called an i-node map used to determine the location of each i-node on the disk. Regular (non-log-structured) UNIX file systems do not. What is the reason for this distinction?

40. What is the difference between a write-through cache update policy and a copy-back policy?

41. Consider a UNIX-like file system that uses i-nodes with single-indirect and double-indirect blocks and a block size of 1024 (2^{10}) bytes. If the block size of the file system is doubled, by approximately what factor does the maximum possible file size increase? Your answer should be an integer.

42. For both parts of this question, assume that the file system uses per-file indexing (like UNIX) with 10 direct pointers, a single indirect pointer, and a double indirect pointer. The block size is B, and the size of a block number (block pointer) is 4 bytes. Assume that the size of a block (B) is greater than the size of a per-file index.

    a. Define the total internal fragmentation of a file to be the amount of space (in bytes) that is allocated for that file but not used. This includes space allocated for file data but not used. It also includes space allocated for data block pointers but not used. What is the maximum total internal fragmentation that can occur for a file in this file system, and for what file size(s) does this maximum occur?

    b. Consider a file of size B^2 bytes. How many extra disk operations are required to read this file (all of it) from the file system described above, as compared to the number required to read a file of the same size from a file system that uses a global file index (like DOS)? For the UNIX-like file system, assume that only the per-file index (the i-node itself) is cached. For the other file system, assume that only the global file index is cached.

43. **a.** Consider a UNIX file system with a block size of 4 kilobytes (2^{12} bytes). The file system is stored on a disk that spins at 100 rotations per second. The disk has 64 (2^6) sectors per track, with 512 (2^9) bytes per sector. The expected seek time for each disk operation is 10 milliseconds.

Periodically, the file system writes the dirty blocks from its buffer cache back to the disk. Typically, the file system has 64 (2^6) dirty blocks at the end of each period. What is the expected time for the file system to write 64 dirty blocks to its disk?

b. Suppose that a log-structured file system is stored on a disk like the one described in part (a). Periodically, the file system must write a log segment from memory to the disk. The size of a log segment is 256 kilobytes (2^{18} bytes. How long will it take the file system to write a single log segment to the disk? Assume that the entire log segment is stored in a single cylinder, and ignore any time that might be required to switch from track to track as the segment is being written.

44. Using pseudo-code, implement `unlink(pathname)` for a UNIX-like file system. This procedure removes a link to the file specified by the *pathname* argument, which specifies an absolute pathname. After the `unlink` operation, `pathname` is no longer a valid file name. Assume that the i-number of the root directory is zero. You may use the following functions in your implementation:

`int num_components(string pathname)` This function returns the number of components in the specified pathname. For example, num_components(*"/a/b/c"*) returns 3, num_components(*"/foo"*) returns 1, and num_components(*"/"*) returns 0.

`string get_component(int i, string pathname)` This function returns a string representing the *i*th component of the specified pathname. The number "i" should be a positive integer. For example, get_component(*2, "/a/b/c"*) will return the string "b", and get_component(*1, "/foo/bar"*) will return "foo". If there is no *i*th component, the function returns the empty string "".

`int dsearch(int i-number, string component)` This function is used to search a directory file for an element whose name matches the "component" argument. The parameter "i-number" must be the i-number of the directory file to be searched. The function returns the i-number of the matching entry, For example, the call dsearch(*10, "foo"*) will return the i-number of the "foo" entry in the directory file whose i-number is 10. If there is no such entry in the specified directory, the function returns the value -1.

`int ddelete(int d, string path_component)` If `path_component` appears in directory file `d`, its entry is deleted from the directory. The function returns the i-number from the deleted entry. If `path_component` cannot be found in the directory, a negative value is returned.

`void remove_link(int i-number)` This function decrements the link count in the i-node whose i-number is specified. If the link count becomes zero, the i-node and all blocks to which it points (directly or indirectly) are freed.

45. A copy-back cache update policy may result in fewer secondary storage updates than a write-through cache of the same size subjected to the same workload. Explain why.

46. Suppose that a file system contains a file called "foo" that is 6000 bytes long. The file system is block-oriented with a block size of 1000 bytes. The file system's buffer cache holds four blocks. It uses an LRU replacement policy and a write-through update policy.

A process accesses the file using the pseudo-code shown below. Assume that the buffer cache is empty at the time the process starts running and that the i-node for file "foo" is already in memory. Note that the Read system call returns the number of bytes actually read from the file. It always reads the number of bytes requested unless the end of the file has been reached.

```
int position = 0; // the current read position
const int bufsize = 500;
char buffer[bufsize];
int r,d;

d = Open("foo");
```

```
loop
    Seek(position); // seek to the current read position
    r = Read(d,buffer,bufsize); // read bufsize bytes from "foo"
    Seek(position+3000); // seek 3000 beyond the current read position
    Write(d,buffer,r); // write buffer to the file
    position = position + bufsize;
until ( position ≥ 3000 );
Close(d);
```

a. How many disk reads does the file system perform as a result of this pseudo-code? (Each disk read reads one block of data.) Indicate which file system calls (from the code above) cause the disk reads, and how many they cause.

b. How many disk writes does the file system perform as a result of this pseudo-code? Indicate which file system calls cause the disk writes, and how many each causes.

c. Would your answer to part (a) change if the file system's block size was 500 bytes, rather than 1000, and the buffer cache holds eight of the smaller blocks? The pseudo-code remains unchanged. If so, give the new answer. If not, explain why it would not.

d. Would your answer to part (b) change if the buffer cache used a copy-back update policy? The exact policy is that dirty file blocks are only copied to disk when they are replaced in the cache, or when their file is closed. If your answer would change, give the new answer. If it would not, explain why it would not. Note that for this part of the question, the block size is again 1000 bytes, as it was for part (b).