QUESTION 1 6 points ✓ Saved

In CPU scheduling, which of the following are true?

- [x] FIFO and SJF are for non-preemptive scheduling, and RR and SRJF are for preemptive scheduling
- [x] Compared to FIFO, RR has shorter responsive time but larger turnaround time.
- [ ] RR and SRJF are starvation free, because they are preemptive.
- [ ] SRJF has better job throughput than RR.
- [ ] For RR, a smaller time slice means shorter response time as well as shorter turnaround time
- [ ] Compared to SJF, SRJF has shorter response time and larger turnaround time

QUESTION 2 7 points ✓ Saved

Which of the following are true for FFS and NFS?

- [x] NFS can co-exist with other file systems.
- [ ] NFS's Close-to-Open consistency model guarantees that there is no inconsistency.
- [x] In NFS, not all requests from clients can be made idempotent.
- [x] Generation # in NFS will not be decremented.
- [ ] The major reason for FFS to be faster is that it treats disks like RAM.
- [x] Groups in FFS can make inodes closer to data blocks, and leads to better space efficiency.
- [x] Smart allocation policy in FFS is to improve runtime performance.

QUESTION 3 6 points ✓ Saved

Which of the following are true?

- [ ] Integrity property in security prevents sensitive information from unauthorized access.
- [x] RAID-5 is strictly better than RAID-4.
- [ ] Access control list of a file should be stored only in memory.
- [ ] RAID-1 has the best redundancy among RAID-0, 1, 4, 5.
- [x] Salting in password storage can make attacks harder.
- [x] Demand paging in page selection will trigger page fault for every newly accessed page.

QUESTION 4 8 points ✓ Saved

Assume you run “sleep 3” and “exec sleep 3” in your shell respectively. Describe what happens, and explain why it happens this way. (Hint: t how “fork” and “exec” work)

For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

When we run "sleep 3" the shell will temporarily pause before returning the prompt back. Then running "exec sleep 3" will also temporarily pauce the shell however it will not return the prompt back and the teminal session ends.

The difference between the two commands is the fact that when you run a command the shell forks this command into a new subshell process and terminates this subshell when the command completes which allows the parent shell to run without issues. When you use the "exec" command it runs the command directly on the parent shell and therefore when the command terminates it terminates the parent shell which ran the process.

P 111 WORDS POWERED BY TINY

QUESTION 5 9 points ✓ Saved

Here is an example of a web server using multi-threading. It creates a new thread to serve every request. Suppose you like to limit the resource consumption by allowing no more than 100 active threads simultaneously, how do you complete the code to realize this limit? (Hint: use semaphore(s). pseudo code is enough.)

```
1).
web_server() {
        2).
        while (1) {
                int sock = accept();
                3).
                thread_cirete(handle_request, sock);
        }
}

handle_request(int sock) {
        …process request
        close(sock);
        4).
}
```

1). (nothing needed)

web_server(){

2). sem_init(threads_left, 99)

while (1){

int sock = accept();

3). sem_wait(threads_left)

thread_cirete(handle_request, sock);

}

}

handle_request(int sock){

...process request

close(sock);

4). sem_post(*threads_left)

}

QUESTION 6

10 points Saved

Consider that requests to read the following set of logical block numbers are enqueued to be served from a disk that has 100 logical blocks laid out sequentially from block 0 to block 99.

{1, 22, 14, 72, 86, 32, 11, 66, 45, 80}

Assume that the seek time in moving the disk arm head from logical block i to block j is proportional to |i – j|.
Given that the arm head is currently positioned at block 75 and is in the midst of moving in the direction towards block 0, what is the sequence in which the enqueued blocks will be read with the i) SCAN, and ii) C-SCAN algorithms (picks up requests on way down only)?

i) Scan - roates full way through

Arm Head Starting at Block 75 moving towards Block 0

75 - > 72 -> 66 -> 45 -> 32 -> 22 -> 14 -> 11 -> 1 -> 80 -> 86

i) C-Scan - Gets from starting to end then restarts from beginning

75 - > 72 -> 66 -> 45 -> 32 -> 22 -> 14 -> 11 -> 1 -> 86 -> 80

QUESTION 7

10 points Saved

Consider a UNIX-style inode with 10 direct pointers, one single-indirect pointer, and one double-indirect pointer only. Assume that the block size is 8K bytes, and the size of a pointer is 4 bytes.

a. What is the largest file size that can be indexed in this system?
b. How many blocks (including indirect blocks) are needed to address a file of size 100 bytes, 10K bytes and 4G bytes?

7 a. What is the largest file size that can be indexed in this system?
10 Direct Pointers which points to 8K data blocks: 10 * 8K = 80K
1 Single Direct Pointer which points to 8K index block, index block holds: 8K/4 = 2K pointers to data blocks
Single Indirect Direct Pointer indirectly points to: 2K * 8K = 16M
Double Indirect Pointer indirectly points to 2K * 2K * 8K = 32G
Sum all pointers: (80K + 16M + 32G) = Maximum File Size

b.

100 B = 0.1 KB

1 Block is enough

10KB

2 Blocks are enough

4G = 4194304 KB

First 80KB stored in 10 blocks, so remaining 4194304 - 80 = 4194224 KB

4194224 KB / 8KB = 524278, since a single-indirect block can store 8KB/4B = 20K, we only need 1 single-indirect block.

So total # of blocks =10+1+ 524278= **524289 Blocks are enough**

QUESTION 8

10 points ✓ Saved

Explain the first 10 steps in the timeline below for file write.

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data [0]	bar data [1]	bar data [2]
create (/foo/bar)			read							
						read				
				read						
							read			
		read								
		write								
							write			
					read					
					write					
				write						
write()					read					
	read									
	write									
								write		
					write					
write()					read					
	read									
	write									
									write	
					write					
write()					read					
	read									
	write									
										write
					write					

For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

Paragraph Arial 10pt

(1) read root inode to locate root data
(2) read root data to lookup foo and find foo's inode
(3) read foo inode to locate foo data
(4) read foo data to find bar's inode
(5) read inode bitmap data to find a free inode
(6) write inode bitmap to initialize the inode to identify that the inode bitmap is used
(7) write foo data to update entries due to addition of D entry of bar
(8) read bar inode to retrieve bar's metadata as part of file open operation
(9) write bar inode to update its last access timestamp
(10) write foo inode to update metadata of directory's inode, size, and data

P 113 WORDS POWERED BY TINY

QUESTION 9

10 points ✓ Saved

Assume a single-level page table system with 4KB page size, 64-bit address and 8-byte PTE.

a. How many pages are needed?

b. How much space would the page table take up? Hint: think about how big the address space is; use power-of-two math.

For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

Paragraph Arial 10pt

a) Offset Bits = 12 because 2^12 = 4096 = 4KB Page Size

64 - 12 = 52 bits for page numbers

How many pages are needed?: 2^52 pages

b) 2^52 * 8 Bytes = 36 (PB) Petabytes

How much space would the page table take up? 36PB

P » STRONG 52 WORDS POWERED BY TINY

QUESTION 10

9 points Saved

For a hard disk, assume the rotational rate is 5400 RPM, average seek time is 7 ms and the average # sectors per track is 500.

a. What is the average rotation time?
b. What is the average transfer time?
c. What is the disk access time?

a) It takes 1 / 5400RPM for one revolution = (1000 ms/sec * 60 sec/min) / 5400 RPM = 11.11 ms

The average rotation time is 1/2 the time for a complete revolution so 11.11 ms / 2 = 5.56ms

What is the average rotation time? 5.56ms

b) We have 500 sectors per track. So we use 1/500 revolutions of the disk to read the full sector. This is 11.11ms / 500 = 0.002222ms

What is the average transfer time? 0.002222ms

c) The disk access time is the sum of average seek time, average rotation time, and average transfer time. 7ms + 5.56ms + 0.002222ms = 12.56ms

What is the disk access time? 12.56ms

QUESTION 11

15 points Saved

Consider a processor with 16-bit address space and 4KB page size. Page Table is at 0x2000 and each PET is 4 bytes. Given a page table as follows,

VPN:0	0x0
	0x0
PageTable	0x1
	0x9
	0x7
	0x8
	0
	⋮
VPN:15	0

Assume the CPU is about to execute the following instructions starting at virtual address 0x3000

0x3000: load 0x5320, %eax
0x3004: load 0x4004, %ebx
0x3008: mul %ecx, %eax, %ebx

a. Assuming no TLB, how many memory accesses are needed? What are the physical addresses to be accessed?
b. Assuming a TLB with no valid entry is in use at the beginning of the execution, how many memory accesses are needed?

a)
Offset is 12 bits (because 4KB = 2^12 bytes)
VPN is 20 bits
PFN: 3 = 0x9000

Physical Memory Accesses with Paging
Fetch instruction at virtual addr 0x3000
Mem ref 1: 0x2000 -> VPN 3 -> 1
Fetch instruction at 0x1000 (Mem ref 2)

Exec, load from virtual addr 5230 into %eax
Access page table to get VPN 5->7
Mem ref 3: 0x2014
load 0x7320 into %eax

Fetch instruction = 2
Fetch val = 2

The next two instructions will have 4 references and the final instruction will have 6 references

The total references this is 10 references.

b)

Offset is 12 bits (because 4KB = 2^12 bytes)
VPN is 20 bits
PFN: 3 = 0x9000

Physical Memory Accesses with Paging
Fetch instruction hits TLB at virtual addr 0x3000
Mem ref 1: 0x2000 -> VPN 3 -> 1 [2 Memory References]
Fetch instruction at 0x1000 (Mem ref 2)

Exec, load from virtual addr 5230 into %eax hits TLB
Access page table to get VPN 5->7 [2 Memory References]
Mem ref 3: 0x2014

Fetch instruction = 2
Fetch instruction hits TLB at virtual addr 0x3000 [Note every time we load the references will go from 2->1]

Fetch val = 2

The next two instructions will be 3 and the final instruction will be 1 references

The total references this is 6 references.