Short Answer Questions

1. What is device independence?
2. What is thrashing?
3. What is the difference between paging and segmentation?

Short Answers

1. Device independence is where a process should not depend on a particular device. It may depend on a type of device though. An OS should be free to assign any device of the right type as appropriate. As far as possible, programs should be independent of type of device.
2. Thrashing is when a process spends more time paging than executing. This most commonly occurs when a set of pages needed to avoid page faulting for every reference will not fit into a set of frames allocated to process. Throughput plunges, processes paging increases, but processes do no work. Effective memory access time increases. If frame allocation is local, this limits the effect to one process, but the increased contention for paging device increases effective memory access time for all processes (Lecture Slides 2022-05-09)
3. There are a few things different between paging and segmentation. One thing to note would be that paging is fixed size and segmentation is variable size.
    a. For how segmentation is done: Associated with each segment table are 2 registers. Segment table base register (STBR) holds address of start of segment table. Segment table limit register (STLR) holds the highest address of segment table. Addresses are (s,d) where s is segment number and d is offset into segment.
    b. For paging (Lecture Slides 2022-04-29): Virtual address is split in two. High bits represent page number. Low bits represent page offset. Page table has the address of each page frame in physical memory. Frame is physical memory into which a page is put. Page is a unit of virtual memory put into physical memory. Both are the same size, defined by hardware. Usually 1024, 2048, 4096, or 8192 words or bytes per page. If page contains p words, virtual address va produces: page number = va / p. page offset = va % p. If p is power of 2, then page number = high order bits of va. page offset = low order bits of va. Address of page table stored in Page Table Base Register (PTBR). Add this to the page table number to get the address of the page table entry. Get the physical address of the frame. Add offset to that address to get a physical memory address corresponding to the virtual address.
        i. This was also demonstrated in Homework #2.

Answer #4

You are the president of Cheapo Computronics, Inc., and your star hardware designer has suggested a brilliant idea: Implement segmentation, but let the least significant *m* bits of a virtual address be used to select the segment, and let the other bits determine the offset. What is the problem with this idea?

Using the brilliant idea provided by the star hardware designer, a problem arises with the segment and offset. For example when using 0001 and 0010, where the segment address equates to 00, the two words are placed directly next to each other, given that they are in the same segment. Even though they are in the same segment, they have different offsets. If 0001 is added by 1, then it will be in conflict with 0010 and thus making the offset pointless. Whereas compared to the previous implementation of segmentation, there would be a larger difference between the two segment addresses and the offset would be able to determine legal memory accesses.

Answer #5

Disk-scheduling Policies

All disk scheduling policies degenerate into FCFS (First Come First Serve). SJF (Shortest-job first), LOOK and CLOOK will only reduce to FCFS when scheduling of one request at a time is done.

Given a seek takes 0.5 + 0.4T msec, where T is number of cylinders moved, while the arm is initially at cylinder 100, the disk has 200 cylinders, and the arm is moving inward, FCFS can operate more efficiently than SCAN for some job requests of cylinders: 90, 95, 105 where FCFS will first move to 90, then 95, then 105. This will produce the result for mean wait time:

$$([0.5 + 0.4(10)] + [0.5 + 0.4(15)] + [0.5 + 0.4(25)]) / 3\,ms$$
$$= (4.5 + 6.5 + 1.5) / 3\,ms = 12.5 / 3\,ms \approx 4.17ms$$

Meanwhile SCAN will first hit 105, as its arm is moving inward and will go all the way out to 200, then backtrack to 95, and 90. Its total mean wait time would be:

$$([0.5 + 0.4(5)] + [0.5 + 0.4(205)] + [0.5 + 0.4(210)]) / 3\,ms$$

$$= (2.5 + 82.5 + 84.5) / 3\,ms = 56.50ms$$

In terms of FCFS vs SSTF, if a job request consists of: 104 97 89 then FCFS will take 104, 97, and 89. This will produce for FCFS, the result:

$$([0.5 + 0.4(4)] + [0.5 + 0.4(7)] + [0.5 + 0.4(15)] / 3\,ms$$

$$= (2.1 + 3.3 + 6.5) / 3\,ms = 11.9 / 3\,ms \approx 3.97ms$$

Now calculate for SSTF, where the same job request will be ran in the order: 97, 104, 89

$$([0.5 + 0.4(3)] + [0.5 + 0.4(7)] + [0.5 + 0.4(32)]) / 3$$

$$= (1.7 + 3.3 + 13.3) / 3\,ms = 18.3ms = 6.1ms$$

For both SCAN and SSTF, these are two scenarios where FCFS operates faster.

Answer #6

Consider a file currently consisting of 100 blocks of 512 words each. Ignoring the access to update the device directory, and assuming a disk block number fits into a single word, how many disk I/O operations are involved with contiguous, linked, and indexed allocation strategies, if one block:

a. is removed from the beginning?
b. is added in the middle?
c. is added at the end?

(Parts a and b provided by Discussion 0520)

a. Removed from beginning:
    i. Contiguous: Only the position of the file directory needs to be updated, which means the beginning of the data block is from block 1 to block 2. Zero operations
    ii. Linked: The OS must read the first linked block to retrieve the second linked block's position, and then update the beginning of the data blocks. 1 read operation
    iii. Indexed: Remove the block from memory at the given index, 1 write operation
b. Removed from middle:
    i. Contiguous: For each contiguous block, the block must be read and written into (shifted)other places, then written the new data into the new block. 50 * 2(1 read + 1 shifted) + 1 written new data = 101

       However, in the worst case, we do not have sufficient space to store data. As a result, we need to delete the entire blocks and rebuild them in a new location. 100 * 2 (1 read + 1 shifted) + 1 written new data = 201

    ii. Linked: We need to read from the first linked block to find the middle one, add a new block, then adjust the pointer for the original 50th linked block. 50 read + 1 create new block + 1 write the new pointer
    iii. Indexed: We need to read the index table, write the new block's location in the index table, and write the new data into the new block. 1 read + 1 write in table + 1 write new data
c. Removed from end:
    i. Contiguous: 1 Operation. Write the new block, unless we do not have sufficient space to store data. As a result, we need to rebuild them in a new location. 100 * 2(1 read + 1 write) + 1 written new data = 101.
    ii. Linked: 3 Operations. Read the last block pointer, then write the new block into that block's pointer, rewrite the last block pointer to point to the new block.
    iii. Indexed: 1 Operation. Write the new block to the index and update the index in memory.