CS3104 Exam

200007413

December 12, 2022

1

(a) I would argue this approach is not the best approach. This is because 46 cores and 192G of ram are wasted at any given time when researchers are using the server.

Instead, I would suggest that at least 4 researchers use the server simultaneously to run experiments.

As there are 64 processor cores, each of these could be working on a different task. Given the 8 core maximum requirement, a minimum of 8 researchers could be using the server without impacting each other. However, as there is only 256GB of RAM, there are only a maximum of 4 concurrent researchers that could use the server without potentially impacting each other at all.

However, I would propose that potentially more than 4 researchers should work simultaneously utilizing memory swapping to find the optimal balance of CPU use and RAM usage. Given 5 researchers working, a potential maximum RAM use is above the 256GB the server has, but because the 64GB is a maximum requirement is a maximum requirement, if it is probable that researchers are not going to be always using all 64GB at one time, it is unlikely that all 256GB will be used. If this does occur, then memory not used in a while can be swapped onto storage, which would only increase the CPU usage which can be done on one of the spare cores.

This would allow the research to be done much faster as research is being done simultaneously, whilst not impacting the researchers, just potentially slowing each researcher down by a small amount due to memory swapping.

(b) (i) Infrastructure-as-a-service (IaaS) would be sufficient for this use case as they just require an increased networking and processing capacity during certain times which can be achieved. This keeps all the proprietary applications already setup used and user data manageable by the company.

(ii) Software-as-a-Service (Saas) would be most applicable here as they don't have to worry about the expense of setting up the servers or infrastructure, as well as the applications and databases.

(c) (i) These protection mechanisms protect the device from malicious apps the user may download. This is because apps will run in user mode, and must request OS permission for privileged operations. They also will not prevent hardware from working during heavy usage, and cannot access other apps memory maliciously.

(ii) Dual-mode operation works by splitting the process into user-mode and kernel-mode. This is done all by the OS.

Hardware interrupts are initially made by the hardware when they have data they need to send to the OS. This interrupt is received by the OS which then manages that data.

Memory protection is initially done by the OS which sets the protection information for each part of memory (such as protection bits), and tells the hardware what memory is accessible to a process. The hardware then actually gives or rejects access to these parts of memory.

(iii)
- Without dual-mode operation, a malicious program would be able to run any privileged code it wanted. This could be used to modify the kernel code itself in any manner, thus leading to obvious security implications. It could also access any kernel memory space it wanted to by changing the memory permissions, and so could change any IO input values too.

  An example of this would be adding in spyware into the kernel code which would send data about usage and key-presses to a server.

- Without hardware interrupts, a malicious program could take control of a computer. This would mean taking up enough resources to prevent the hardware input from being processed. This could be used to run ransomware software which prevents hardware interaction whilst encrypting all of the information so a user cannot stop it half way through.

  Hardware interrupts allow a user to (for example) press the power button to shut the computer off preventing the process from continuing.

- Without memory protection the malicious program could effectively change kernel code. This is because when the kernel runs, it loads information into memory which the malicious program could change if no memory protection was implemented. For example, you could change a kernel memory pointer of a piece of code to a new part of memory storing a malicious piece of code which will then be ran in kernel mode.

  This would then counter the dual-mode operation leading to the same security concerns.

2

(a) The SSTF algorithm will give the order: [55,52,69,74,94,129,162] with a total head movement of 107.

The LOOK schedule will give the order: [69,74,94,129,162,52,55] with a total head movement of 215.

Therefore, the SSTF has the least head movement.

(b) Linked allocation means that a files data is spread across blocks, where each block contains a pointer to the next block. This requires storing a pointer to the first block related to that piece of memory, then going through the list to get to a certain position.

When adding data to a file, you just add a pointer from the last block connected to the new block.

Indexed allocation requires allocating an index block for each file , then the data is spread across blocks which are pointed to from the index block.

To add data to a file, you just add the pointer to the new block to the list in the index block. To reference, you just require the pointer to the index block.

(c) (i) The pointers require 6 bits to reference each block uniquely.

(ii) To address block 55, the whole list needs to be traversed by accessing each previous block to get the next pointer. Therefore, 4*4ms = 16ms is required to get the block address 55, then another 4ms to read the data totalling 20ms.

(iii) The size of the index block for the above sequence would be 512 bytes. This is because the index block just uses a block which would otherwise be used for data. However, only 36 bits (9 bytes) are actually used, the rest remains allocated but unused.

(iv) An index block can store 341 pointers to blocks. This is because to reference each block 6-bits are required, and with a 512-byte block you have 2048-bits which can store 341 pointers.

This file system is probably not good for most use cases as the maximum size of a single file is roughly 174 KB which would only be useful for very specific and small-scale applications.

(d) (i) RAID 4 will allow 4 of the 5 disks to be used for data storage, with the fifth storing interleaved parity blocks. Each block of disks 1-4 will run through a parity algorithm which is then stored on disk 5. Therefore, 4*100GiB = 400GiB is usable here.

RAID 5 uses the same interleaved parity blocks, but here they are spread across the disks. Therefore, the same size and number of parity blocks will be created using the same amount of storage in total. Therefore, 400GiB is usable here too.

(ii) If you are a hospital with a large budget and can afford a lot of storage, but data reliability is vital then RAID-6 is preferable as it has two parity calculations using more space for parity, but enabling 2 disk failures to be recovered.

Another scenario would be when your storage devices are less reliable, you should use RAID-6 as more of the storage is used for parity to prevent data loss as it can recover from more disk failures which can then be replaced.

3

(a) Effective access time of no page faults is 250ns. Therefore, to keep the slowdown less than 80% requires an effective access time of $250/0.8 = 312.5$ns.

(i) $(1-p)\cdot 250 + p\cdot 8000000 = 312.5$

$250 - 250p + 8000000p = 312.5$

$7999750p = 62.5$

$p = \frac{1}{127996}$

(ii) $(1-p)\cdot 250 + p\cdot 2000000 = 312.5$

$250 - 250p + 2000000p = 312.5$

$1999750p = 62.5$

$p = \frac{1}{31996}$

(iii) $(1-p)\cdot 250 + p\cdot 20000000 = 312.5$

$250 - 250p + 20000000p = 312.5$

$19999750p = 62.5$

$p = \frac{1}{319996}$

$4KiB = 4096$ bytes requires 12-bits for byte offset.

(b) (i) a0202c = 101000000010000000101100.

Offset bits are 12-bits = 000000101100 = 02c. Page number is the remaining 12-bits = 101000000010 = a02.

The hardware would address page a02, and detect that the page is invalid. It would then cause a page fault which would make the OS load the page into memory and restart the operation. The frame number would then be returned and added to the offset giving the physical address number which would be referenced as read-only by the hardware.

(ii) First, the hardware would search the page table to check if any pages are already allocated to frame 100. This would return page a03. The hardware would then set the page to invalid and swap the data in the frame out into storage. The frame number associated with a03 would be reset and then the frame number associated with b12 would be set to 100.

(iii) The hardware would search the page table and find a04 pointing to ca7. It would swap out the frame ca7 out to storage, then unset the valid bit for a04, and reset the frame number. It would then associate the frame number c13 to the frame number ca7.

(c) (i) First, j will be referenced, which will get 0001d, this checks if j > 0. Then i is referenced setting i=0 getting 702dc. Size is referenced which also uses 702dc to check if i < size.

Then i is referenced to get the array offset also using 702dc, then array is referenced to get the element in the array using 0001b,

For each subsequent for loop call, i and size are referenced both accessing 702dc, then a repeat of the index and array accesses are referenced. The for loop runs 4 times in total ending with a final check of i and size getting 702dc.

After the for loop, j is referenced using 0001d, and then it exists as j becomes 0.

Therefore, the reference string from the while loop is:

0001d,702dc,0001b,702dc,0001b,702dc,0001b,702dc,0001b,702dc,0001d.

(ii) The page replacement process would begin by storing 0001d and setting it's reference bit to 1. The queue then moves to the next position which then stores 702dc, then moves on to the final frame setting to 0001b and the pointer loops back to the first item containing 0001d.

When 702dc is then referenced next, the queue sets 0001d's reference bit to 0 and then moves to the frame storing 702dc. The reference bit remains at 1 then the queue moves to pointing at 001b.

Then, 0001b is referenced and the reference bit remains 1 and the queue moves back to the start.

When this repeats in the loop, the reference bit of 0001d is kept at 0, but the reference bits of 702dc and 0001b remain at 1.

Then, when the final 0001d is referenced, the queue pointer is at 0001b. The reference bit is set to 0, and the queue advances to the start with 0001d where the reference bit is set to 1.

There is no page replacement occurring as only 3 pages, and so 3 frames are required to store the array, j, i, and size as long as local page replacement is used. If global replacement is used for other programs, then the 0001d frame would most likely be swapped out first as it's reference bit remains at 0 during the for loop.