Homework Set 3 - Virtual Memory, I/O, File Systems

Yiming Liu 2023010747

Problem 1

(1) Given that the lowest 12 bits of the virtual address are used as the offset within the page. This offset specifies individual bytes within the page. Therefore, the page size is 2^{12} bytes, or 4 kB.

(2) Since 12 bits are used for the page offset, the remaining bits 20 bits are used for the page number. So the maximum size of physical memory this system can support is $2^{20} \times 2^{12} = 2^{32}$ bytes, or 4 GB.

(3) Each page table entry (PTE) is 32 bits (4 bytes) in size. First-level page table has 2^{10} entries at most, and Second-level page tables has $2^{10} \times 2^{10} = 2^{20}$ entries at most. Therefore, the maximum size that a page table of a process can be $(2^{10} + 2^{20}) \times 4 = (1024 + 1048576) \times 4 = 4198400$ bytes.

(4) Only two entries in the page tables are actually used: one for the lowest address and one for the highest address. This means that we have two first-level page table entries pointing to a second-level page tables, each with only one valid entry. Therefore the size of the page table would be $(2^{10} + 2 \times 2^{10}) \times 4 = 12288$ bytes.

(5) Each page is 2^{12} bytes and a process uses 512 kB of physical memory in total, so we can get the number of pages used by the process is $\frac{512\text{kB}}{4\text{kB}} = 128$ pages.

The minimum size of the page table occurs when all pages are perfectly contiguous in memory, using one first-level page table entry points to one second-level page table which contains 128 entries. Therefore, the size of the page table would be $(2^{10} + 2^{10}) \times 4 = 8192$ bytes.

The maximum size occurs if each of the 128 pages is non-contiguous, potentially requiring up to 128 entries in the first-level page table, each pointing to a different second-level page table. Therefore, the size of the page table would be $(2^{10} + 128 \times 2^{10}) \times 4 = 528384$ bytes.

(6) The TLB tag is 20 bits because it needs to uniquely identify pages based on the first 20 bits of the virtual address. Each TLB entry includes the page table entry, a tag and a valid bits, so the bits of each TLB entry is $32 + 20 + 1 = 53$ bits.

Problem 2

(1)

	A	B	C	D	E	C	A	B	C	D	F
i	A				E					D	
ii		B					A				F
iii			C								
iv				D				B			

(2) When there are four physical frames, the number of faults is 9. So we need to increase the number of physical frames to reduce the number of faults.

If there are 5 physical frames, the number of faults is 6. And when physical frames are larger than 5, the number of faults is always 6 since there are only 6 different pages in total.

	A	B	C	D	E	C	A	B	C	D	F
i	A										F
ii		B									
iii			C								
iv				D							
v					E						

Problem 3

(1) Average read or write time = Average seek time + Average rotational delay + Average transfer time + Average controller time. Average rotational delay = $\frac{1}{2} \times \frac{1}{\text{RPM}} \times \frac{60s}{1\,\text{min}}$. Average transfer time = $\frac{\text{Sector size}}{\text{Disk transfer rate}}$. Average controller time = $\frac{\text{Sector size}}{\text{Controller transfer rate}}$.

For disk a, Average seek time = 11 ms, Average rotational delay = $\frac{1}{2} \times \frac{1}{7200} \times 60 = 4.17$ ms, Average transfer time = $\frac{1024}{36 \times 2^{20}} = 0.027$ ms, Average controller time = $\frac{1024}{500 \times \frac{2^{20}}{8}} = 0.016$ ms. Therefore, Average read or write time for disk a = $11 + 4.17 + 0.027 + 0.016 = 15.213$ ms.

For disk b, Average seek time = 9 ms, Average rotational delay = $\frac{1}{2} \times \frac{1}{7200} \times 60 = 4.17$ ms, Average transfer time = $\frac{1024}{32 \times 2^{20}} = 0.031$ ms, Average controller time = $\frac{1024}{520 \times \frac{2^{20}}{8}} = 0.015$ ms. Therefore, Average read or write time for disk a = $9 + 4.17 + 0.031 + 0.015 = 13.216$ ms.

(2) The minimum time is reached when the seek time and rotational delay are 0.

For disk a, Average transfer time = $\frac{2048}{36 \times 2^{20}} = 0.054$ ms, Average controller time = $\frac{2048}{500 \times \frac{2^{20}}{8}} = 0.031$ ms. Therefore, the minimum read or write time for disk a = $0 + 0 + 0.054 + 0.031 = 0.085$ ms.

For disk b, Average transfer time = $\frac{2048}{32 \times 2^{20}} = 0.061$ ms, Average controller time = $\frac{2048}{520 \times \frac{2^{20}}{8}} = 0.030$ ms. Therefore, the minimum read or write time for disk a = $0 + 0 + 0.061 + 0.030 = 0.091$ ms.

(3) For both disks (a and b), the average seek time is the dominant factor, contributing the most to the total time for each operation. The other components like rotational delay, disk transfer time, and controller transfer time are significantly smaller.

If we consider different access patterns, the dominant factor may change.

1. Random Access Patterns:

In random access patterns, the dominant factor is the seek time and rotational latency.

2. Sequential Access Patterns:

In sequential access patterns, the seek time and rotational latency become less significant because the disk head moves continuously in a more predictable manner. In this case, the disk transfer rate and controller transfer rate become more important.

Problem 4

(1) Since its a 1GHz processor, it can run 1000000000 cycles per second. So between each byte arrival, the processor can run $0.02 \times 10^{-3} \times 1000000000 = 20000$ cycles.

We keep polling without stop, so the entire operation will take $20000 \times 999 + 50 + 100 = 19980150$ cycles. And the processor will do $\frac{20000-100}{50} \times 999 + 1 = 397603$ polls.

(2) Between two bytes arrival, the processor can run $20000 - 200 - 100 = 19700$ cycles. So the processor will spend $19700 \times 999 = 19680300$ cycles on another task.

Problem 5

(1) First, Number of pointers per block= $\frac{\text{Block size}}{\text{Pointer size}} = \frac{4096}{\frac{32}{8}} = 1024$.

For a direct block pointer, it points to a data block with 4096 bytes.

For an indirect block pointer, it points to 1024 direct pointers, and each direct pointer points to a data block with 4096 bytes. So the total size is $1024 \times 4096 = 4194304$ bytes.

For a double indirect block pointer, it points to 1024 indirect pointers, and each indirect pointer points to 1024 direct pointers, and each direct pointer points to a data block with 4096 bytes. So the total size is $1024 \times 1024 \times 4096 = 4294967296$ bytes.

We want to store a 4 GB file which is $4 \times 2^{30} = 4294967296$. So we store 4096 bytes in the direct block, 4194304 bytes in the indirect block, and $4294967296 - 4194304 - 12 \times 4096 = 4290723840$ bytes in the double indirect block, which needs $\lceil \frac{4290723840}{4096 \times 1024} \rceil = 1023$ indirect blocks.

We also know that The total inode size is 256 bytes. Therefore, we need $(1 + 1 + 1023) \times 4096 + 4 \times 2^{30} + 256 = 4299165952$ bytes for storage.

(2) The maximum size of a file can be stored is $(12 + 1024 + 1024 \times 1024) \times 4096 = 4299210752$ bytes.

(3)

1. Access inode to get the double-indirect pointer.
2. Access the double-indirect block to get the indirect block pointer.
3. Access the indirect block to get the direct block pointer.
4. Access the last data block to read the byte.

So, 4 disk accesses are required to read the last byte.

For overwriting,

1. Access inode to get the double-indirect pointer.
2. Access the double-indirect block to get the indirect block pointer.
3. Access the indirect block to get the direct block pointer.
4. Access the last data block to write the byte.
5. Update the inode to reflect the change.

So, 5 disk accesses are required to overwrite the last byte.

(4) For this disk with 4 kB block size, the allocated space per 3 kB file is 4 kB, and the allocated space per 8 kB file is 2×4 kB $= 8$ kB. Since there are half 3 kB files and half 8 kB files, we assume there are $\frac{N}{2}$ 3 kB files and $\frac{N}{2}$ 8 kB files. So the space overhead is $\frac{\text{allocated space}}{\text{actual data size}} = \frac{4 \times \frac{N}{2} + 8 \times \frac{N}{2}}{3 \times \frac{N}{2} + 8 \times \frac{N}{2}} = \frac{12}{11}$.

For disk with 2 kB block size, the allocated space per 3 kB file is 2×2 KB $= 4$ kB, and the allocated space per 8 kB file is 4×2 kB $= 8$ kB. Since there are half 3 kB files and half 8 kB files, we assume there are $\frac{N}{2}$ 3 kB files and $\frac{N}{2}$ 8 kB files. So the space overhead is $\frac{\text{allocated space}}{\text{actual data size}} = \frac{4 \times \frac{N}{2} + 8 \times \frac{N}{2}}{3 \times \frac{N}{2} + 8 \times \frac{N}{2}} = \frac{12}{11}$.

(5) There are two blocks that need to change:

1. The block containing the data for the source directory /a/b. This block is modified to remove the directory entry for the file c.
2. The block containing the data for the destination directory /d/e. This block is updated to add the directory entry for the file f.

(6) (i) Only file metadata need to change because the inode needs to store additional information indicating that it is a symbolic link rather than a regular file.

(ii) Only file metadata need to change because the inode needs to manage reference counts that track how many directory entries point to it.