**COMP-304 ASSIGNMENT-3 SPRING-2021**

**NAME-SURNAME: BARIŞ KAPLAN**

**ID-NUMBER: 69054**

**E-MAIL ADDRESS: BKAPLAN18@KU.EDU.TR**

**1-)**

**The memory management method of contigious allocation:**

**-> Because of the fact that there is no non-contigious parts in the memory for the contigious allocation, the memory management method of contigious allocation does not enable code sharing among processes (no code sharing among processes). For the case where the memory partitions have variable sizes, because of the existence of some unused gaps between the spaces allocated for the different processes in the memory, the external fragmentation exists. In the version of variable size partitions, even though the total amount of free space is greater than or equal to the total amount of space which coming process requires, there might be some cases where each of the holes does not satisfy the free space need of the upcoming process. This situation creates the unused gaps for the case where the memory partitions have variable sizes. However; for the case where the memory partitions have constant sizes, because of the fact that there are no unused gaps between the spaces allocated for different processes in the memory, the external fragmentation does not exist.**

**The memory management method of paging:**

**-> Since the size of each page in the memory is fixed, when all of the pages in the memory are utilized, and when there are no unused spaces between the pages in the memory management method of paging, there is no external fragmentation for the memory management method of paging. By the reference which is given to the identical page frame by the page tables which are located in the memory, the memory management method of paging allows code sharing among processes.**

**The memory management method of segmentation:**

**->Due to the fact that the segments are variable-sized in the memory management method of segmentation and some segments might not be utilized in the memory, the external fragmentation exists for the memory management method of segmentation. Even though we have enough space in total to allocate a coming process in the memory, we cannot find enough space to allocate this process in the memory. When the segments give reference to the several processes in the memory, the memory management method of segmentation enables code sharing among processes. The memory management method of segmentation can share code by the reference given to a particular segment by the other segments.**

**2-)**

**a-)**

**The virtual address size in bits= 32 bits**

**1 kilobyte= 2^10 bytes**

**The page size: 8 kilobyte (8 KB)**

**8 kilobyte= (2^10)\*(2^3)= 2^13 bytes**

**Since the page size is equal to 2^13 bytes, the page offset size in bits is equal to 13 bits.**

**The virtual address size in bits= The page offset size in bits+The virtual page number size in bits**

**32= 13+The virtual page number size in bits**

**32-13= The virtual page number size in bits**

**The virtual page number size in bits= 19 bits**

**The total number of entries in the page table= The total number of virtual pages= 2^19**

**The total space which the page table requires if a single-level page table is used= (The total number of virtual pages) \* (bytes per table entry)**

**Bytes per table entry= 4 bytes= 2^2 bytes**

**The total space which the page table requires if a single-level page table is used= (2^19)\*(2^2)= 2^21 bytes**

**1 megabyte= 2^20 bytes**

**So, 2^21 bytes is equal to 2^1 megabytes.**

**2^1 megabytes= 2 MB(The total space which the page table requires if a single-level page table is used)**

**b-)**

**The physical address size in bits: 28 bits**

**The physical address size in bits= The page offset size in bits + The physical page number size in bits**

**The physical address size in bits= 13+The physical page number size in bits**

**28= 13+The physical page number size in bits**

**28-13= The physical page number size in bits**

**The physical page number size in bits= 15 bits**

**The total number of pages the physical memory have = 2^15**

**The physical memory size= The total number of pages the physical memory have \* The page size**

**The physical memory size= (2^15)\*(8 KB)**

**The physical memory size= (2^15)\*(2^3)**

**The physical memory size= 2^18 KB**

**1 MB= 2^10 KB**

**2^18 KB= 2^8 MB= 256 MB**

**So, the physical memory size in MB is 256 MB.**

**c-)**

**The physical address size in bits: 28 bits**

**The physical address size in bits= The page offset size in bits+ The physical page number size in bits**

**The physical address size in bits= 13+The physical page number size in bits**

**28= 13+The physical page number size in bits**

**28-13= The physical page number size in bits**

**The physical page number size in bits= 15**

**The total number of pages the physical memory have = 2^15**

**3-)**

**For the FIFO Page Replacement Algorithm**

**The page reference string: 1, 2, 3, 4, 2, 1, 5, 2, 1, 2, 3**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Miss** | **Miss** | **Miss** | **Miss** | **Hit** | **Miss** | **Miss** | **Miss** | | **Hit** | **Hit** | **Miss** |
| **1** | **1** | **1** | **4** | **4** | **4** | **4** | | **2** | **2** | **2** | **2** |
|  | **2** | **2** | **2** | **2** | **1** | **1** | | **1** | **1** | **1** | **3** |
|  |  | **3** | **3** | **3** | **3** | **5** | | **5** | **5** | **5** | **5** |

**Firstly; the page with the reference number 1 comes to an empty frame. This results in 1 page fault.**

**Secondly, the page with the reference number 2 comes to the another empty frame. This results in 1 page fault.**

**Thirdly, the page with the reference number 3 comes to the last empty frame. This results in 1 page fault.**

**Then, the page with the reference number 4 comes. 4 is not currently in the memory. In the FIFO page replacement algorithm, if we need to replace a page, we select the oldest page to be replaced. Now, the oldest page’s reference number is 1. So, 4 replaces 1. This causes 1 page fault.**

**After that, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, we do not need to replace any pages here, which means there is no page fault here. (0 page fault)**

**Subsequently, the page with the reference number 1 comes. 1 is not currently in the memory. Currently, the oldest page’s reference number is 2. So, 1 replaces 2. This results in 1 page fault.**

**Next, the page with the reference number 5 comes. 5 is not currently in the memory. Now, the reference number of the oldest page is 3. Hence, 5 replaces 3. This results in 1 page fault.**

**Then, the page with the reference number 2 comes. 2 is not currently in the memorry. Now, the reference number of the oldest page is 4. Therefore, 2 replaces 4. This results in 1 page fault.**

**Subsequently, the page with the reference number 1 comes. Since the page with the reference number 1 is currently in the memory, no page replacement is needed. So, there is no page fault here. (0 page fault)**

**After that, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault)**

**Lastly, the page with the reference number 3 comes. 3 is not currently in the memory. Now, the reference number of the oldest page is 1. So, 3 replaces 1. This results in 1 page fault.**

**The page fault number in total for the FIFO page replacement algorithm: 8 page faults**

**For the Optimal Page Replacement Algorithm**

**The page reference string: 1, 2, 3, 4, 2, 1, 5, 2, 1, 2, 3**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Miss** | **Miss** | **Miss** | **Miss** | **Hit** | **Hit** | **Miss** | **Hit** | **Hit** | **Hit** | **Miss** |
| **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **3** |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** |
|  |  | **3** | **4** | **4** | **4** | **5** | **5** | **5** | **5** | **5** |

**Firstly; the page with the reference number 1 comes to an empty frame. This results in 1 page fault.**

**Secondly, the page with the reference number 2 comes to an another empty frame. This results in 1 page fault.**

**Thirdly, the page with the reference number 3 comes to the last empty frame. This results in 1 page fault.**

**Then, the page with the reference number 4 comes. 4 is not currently in the memory. In the optimal page replacement algorithm, we need to replace the page which will not be referenced for the longest time period in the future. Now, the page which will not be referenced for the longest period of time in the future is 3. So, 4 replaces 3. At the end, 1 page fault occurs here.**

**After that, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault)**

**Subsequently, the page with the reference number 1 comes. Since the page with the reference number 1 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault).**

**Next, the page with the reference number 5 comes. 5 is not currently in the memory. Since 4 is not used in the future, we need to replace 4 with 5. Therefore, 1 page fault occurs here.**

**Then, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault)**

**After that, the page with the reference number comes. Since the page with the reference number 1 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault)**

**Then, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Thus, there is no page fault here. (0 page fault)**

**Lastly, the page with the reference number 3 comes. 3 is not currently in the memory. Currently, there is an equality in terms of the future usage of the existing pages in the memory. So, we need to use FIFO algorithm here. Since the oldest page is 1 now, we need to replace 1 with 3. At the end, 1 page fault occurs here.**

**The page fault number in total for the Optimal page replacement algorithm: 6 page faults**

**For the LRU (Least Recently Used) Page Replacement Algorithm**

**The page reference string: 1, 2, 3, 4, 2, 1, 5, 2, 1, 2, 3**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Miss** | **Miss** | **Miss** | **Miss** | **Hit** | **Miss** | **Miss** | **Hit** | **Hit** | **Hit** | **Miss** |
| **1** | **1** | **1** | **4** | **4** | **4** | **5** | **5** | **5** | **5** | **3** |
|  | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** | **2** |
|  |  | **3** | **3** | **3** | **1** | **1** | **1** | **1** | **1** | **1** |

**Firstly; the page with the reference number 1 comes to an empty frame. This results in 1 page fault.**

**Secondly, the page with the reference number 2 comes to an another empty frame. This results in 1 page fault.**

**Thirdly, the page with the reference number 3 comes to the last empty frame. This results in 1 page fault.**

**Then, the page with the reference number 4 comes. In the LRU algorithm, the page which has not been used for the longest time period needs to be replaced. Now, the page which has not been used for the longest time is 1. So, 4 replaces 1. At the end, 1 page fault happens here.**

**After that, the page with the reference number 2 comes. Since 2 is currently in the memory, no page replacement is needed here. Therefore, there is no page fault here. (0 page fault)**

**Subsequently, the page with the reference number 1 comes. Now, the page with the reference number 3 is the page which has not been used for the longest time period. So, 1 replaces 3. At the end, 1 page fault occurs here.**

**Next, the page with the reference number 5 comes. Now, the page which has not been used for the longest period of time is 4. Therefore, 5 replaces 4. At the end, 1 page fault occurs here.**

**Then, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Thus, there is no page fault here. (0 page fault)**

**After that, the page with the reference number 1 comes. Since the page with the reference number 1 is currently in the memory, no page replacement is needed here. Hence, there is no page fault here. (0 page fault)**

**Subsequently, the page with the reference number 2 comes. Since the page with the reference number 2 is currently in the memory, no page replacement is needed here. Thus, there is no page fault here. (0 page fault)**

**Lastly, the page with the reference number 3 comes. the page which has not been used for the longest period of time is 5. Therefore, 3 replaces 5. At the end, 1 page fault occurs here.**

**The page fault number in total for the LRU page replacement algorithm: 7 page faults**

**4-)**

**The working set window : 7**

**a-)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Time** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **P1** | **2** | **5** | **7** | **4** | **4** | **1** | **2** | **3** | **7** | **3** | **3** |
| **P2** | **1** | **1** | **1** | **3** | **3** | **4** | **4** | **3** | **5** | **6** | **7** |
| **P3** | **4** | **9** | **3** | **2** | **9** | **8** | **7** | **7** | **7** | **1** | **2** |

**Since the working set window is equal to 7, we should consider the time interval starting at time 3 (3 is inclusive) and ending at time 9(ending time is given in the part a) for each process.**

**For P1**

**The working set at time 9(inclusive): {1,2,3,4,7}**

**The working set size at time 9(inclusive): 5 (The number of the elements in the working set**

**at time 9(inclusive) )**

**---------------------------------------------------------------------------------------------------------------------------**

**For P2**

**The working set at time 9(inclusive): {3,4,5,6}**

**The working set size at time 9(inclusive): 4 (The number of the elements in the working set**

**at time 9(inclusive) )**

**---------------------------------------------------------------------------------------------------------------------------**

**For P3**

**The working set at time 9(inclusive): {1,2,7,8,9}**

**The working set size at time 9(inclusive): 5(The number of the elements in the working set**

**at time 9(inclusive) )**

**b-)**

**The total number of frames in the physical memory: 10**

**If the total demand of frames is higher than the total number of frames in the physical memory, thrashing will occur.**

**The total demand of frames at time 9(inclusive): The working set size of P1 at time 9(inclusive)+ the working set size of P2 at time 9(inclusive) + the working set size of P3 at time 9(inclusive)**

**The working set size for P1 at time 9(inclusive): 5 ( The total number of pages which are**

**referenced in the most recent working set window at time 9(inclusive))**

**The working set size for P2 at time 9(inclusive): 4 (The number of the elements in the**

**referenced in the most recent working set window at time 9(inclusive))**

**The working set size for P3 at time 9(inclusive): 5(The number of the elements in the**

**referenced in the most recent working set window at time 9(inclusive))**

**The total demand of frames at time 9(inclusive): 5+4+5= 14 frames**

**Since the total demand of frames at time 9(inclusive) is greater than the total number of frames in the physical memory(14>10), thrashing will occur.**

**c-)**

**If the total demand of frames is higher than the total number of frames in the physical memory, trashing will occur.**

**Since the working set window is equal to 7, we should consider the time interval starting at time 1(1 is inclusive) and ending at time 7(ending time is given in the part c).**

**The working set for P1 at time 7(inclusive): {1,2,3,4,5,7}**

**The working set size for P1 at time 7(inclusive): 6 ( The total number of pages which are**

**referenced in the most recent working set window at time 7(inclusive))**

**-----------------------------------------------------------------------------------------------------------------------**

**The working set for P2 at time 7(inclusive): {1,3,4}**

**The working set size for P2 at time 7(inclusive): 3 ( The total number of pages which are**

**referenced in the most recent working set window at time 7(inclusive))**

**The working set for P3 at time 7(inclusive): {2,3,7,8,9}**

**The working set size for P3 at time 7(inclusive): 5 (The total number of pages which are referenced in the most recent working set window at time 7(inclusive))**

**The total demand of frames at time 7(inclusive): The working set size of P1 at time 7(inclusive)+ the working set size of P2 at time 7(inclusive) + the working set size of P3 at time 7(inclusive)**

**The total demand of frames at time 7(inclusive): 6+3+5= 14 frames**

**If the total number of frames in the physical memory is greater than or equal to the total demand of frames at time 7(inclusive), the system will not suffer from thrashing. Therefore, there should be at least 14 frames in the physical memory so that trashing does not occur in the system.**

**5-)**

**What are the inode values of file1.txt and file2.txt? Are they the identical or different? Do the two files have the same—or different— contents?**

**The inode value of the file1.txt is 271130.**

**The inode value of the file2.txt 271130.**

**After I create a hard link between file1.txt and file2.txt, the inode value of the file1.txt and the inode value of the file2.txt are same.**

**After I create a hard link between file1.txt and file2.txt by using a command called ‘ln file1.txt file2.txt’ , the file1.txt and the file2.txt have same contents.**

**Then, edit file2.txt and change its contents. After you do this change, examine the contents of file1.txt. Are the contents of file1.txt and file2.txt the same or different?**

**After I alter the content of the file2.txt, the content of the file1.txt and the content of the file2.txt are same.**

**Subsequently , enter ‘rm file1.txt’ to remove the file called file1.txt**

**Does file2.txt still exist as well?**

**Yes, even though I have removed the file1.txt, the file2.txt still exists.**

**What system call is used for removing file2.txt?**

**The unlink system call is used for removing file2.txt.**

**Are the inode values the identical, or is each different? Subsequently, edit the contents of file4.txt. Have the contents of file3.txt been changed? Finally, delete file3.txt. After you have done this deletion, explain what occurs when you attempt to edit file4.txt.**

**After I create a soft link between file3.txt and file4.txt by typing ‘ln -s file3.txt file4.txt’ to the terminal and after I get the inode value of the file3.txt and the inode value of the file4.txt by typing ‘ls -li file\*.txt’ to the terminal, I observe that the inode value of the file3.txt and the inode value of the file4.txt are different. When I change the content of the file4.txt, the content of the file3.txt has also been changed. At the end, when I delete the content of the file3.txt, I observed that I could not edit the file4.txt. Moreover, I observed that the file4.txt became a read-only file.**

**NOTE: You can see the screenshots for the problem-5 in my submission.**