**Individual Assignment 2 (25%)**

CSE 4600 (Section 01) – Operating Systems – Spring 2022

Submitted to

Department of Computer Science and Engineering

California State University, San Bernardino, California

by

Nathan Bush (007463099)

Due Date: May 15, 2022 11:59 pm

*No copy and paste from other colleagues with the same answers and description (particularly in part 1) is allowed. It is required that you carry out the exercises by yourself (with the possibility for collaborating with other colleagues) and provide descriptions (with screenshots wherever necessary) based on your own experience. Copied material from other colleagues will be considered as cheating and dealt with seriously through University academic integrity policies.*

Email: nathan.bush3099@coyote.csusb.edu

Code: <https://github.com/natebush707/CSE_4600_A2.git>

\*\*\* all source .cpp files and screenshots are also included in a .zip file with this report.

1. **(5%) Consider the swapping system below with the memory blocks in the following order. Assume that the following segment requests arrive in the order given**
2. **13K**
3. **10K**
4. **5K**
5. **9K**



Which blocks will be taken for the above requests for the algorithms we covered in the class. Fit the above requests for (i) first fit, (ii) best fit, and (iii) worst fit. Provide your answers step by step for each of these algorithms. For example, for First fit, create a section in your report where you will have the block sequence with the block highlighted per request. Once the block is selected for a request, also indicate remaining capacity of the blocks on the right (e.g., if the request (a) is 4K and the first fit finds the first block (i.e., 12K), then highlight the block with 12K for option (a) and write your answer 8K as the remaining block capacity.

1. **First Fit**
   1. **13K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SELECT 16K BLOCK | | |  |  |  |  |
|  |  | 16K - 13K = 3K | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **12K** | **4K** | **8K** | **3K** | **9K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 3K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |

* 1. **10K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SELECT 12K BLOCK | |  |  |  |  |  |  |  |
| 12K - 10K = 2K | |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **8K** | **3K** | **9K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
| 2K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |  |  |

* 1. **5K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | SELECT 8K BLOCK | | |  |  |  |  |  |
|  | 8K - 5K = 3K | | |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **3K** | **3K** | **9K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  | 3K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |  |

* 1. **9K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SELECT 9K BLOCK | | | | |  |  |
|  |  | 9K - 9K = 0K | | | | |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **3K** | **3K** | **0K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | 0K REMAINING BLOCK CAPACITY | | |  |  |  |  |

1. **Best Fit**
   1. **13K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SELECT 16K BLOCK | | |  |  |  |  |
|  |  | 16K - 13K = 3K | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **12K** | **4K** | **8K** | **3K** | **9K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | 3K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |

* 1. **10K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SELECT 12K BLOCK | |  |  |  |  |  |  |  |
| 12K - 10K = 2K | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **8K** | **3K** | **9K** | **5K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
| 2K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |  |  |

* 1. **5K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | SELECT 5K BLOCK | | |  |  |
|  |  |  |  | 5K - 5K = 0 | | |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **8K** | **3K** | **9K** | **0K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | 0K REMAINING BLOCK CAPACITY | | |  |  |  |

* 1. **9K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | SELECT 9K BLOCK | | |  |  |  |
|  | |  | 9K - 9K - 0K | | |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **2K** | **4K** | **8K** | **3K** | **0K** | **0K** | **12K** | **17K** | **19K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  | 0K REMAINING BLOCK CAPACITY | | |  |  |  |  |

1. **Worst Fit**
   1. **13K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | SELECT 19K BLOCK | |
|  |  |  |  |  |  |  | 19K - 13K = 6K | |
|  |  |  |  |  |  |  |  |  |
| **12K** | **4K** | **8K** | **16K** | **9K** | **5K** | **12K** | **17K** | **6K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 6K REMAINING BLOCK CAPACITY | | | |  |

* 1. **10K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | SELECT 17K BLOCK | | |
|  |  |  |  |  |  | 17K - 10K = 7K | | |
|  |  |  |  |  |  |  |  |  |
| **12K** | **4K** | **8K** | **16K** | **9K** | **5K** | **12K** | **7K** | **6K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  | 7K REMAINING BLOCK CAPACITY | | | |  |

* 1. **5K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | SELECT 16K BLOCK | | |  |  |  |  |
|  |  | 16K - 5K = 11K | | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **12K** | **4K** | **8K** | **11K** | **9K** | **5K** | **12K** | **7K** | **6K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
|  | 11K REMAINING BLOCK CAPACITY | | | | |  |  |  |  |

* 1. **9K segment requested**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SELECT 12K BLOCK | |  |  |  |  |  |  |  |
| 12K - 9K = 3K | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| **3K** | **4K** | **8K** | **11K** | **9K** | **5K** | **12K** | **7K** | **6K** |
|  |
|  |  |  |  |  |  |  |  |  |  |
| 3K REMAINING BLOCK CAPACITY | | |  |  |  |  |  |  |  |

1. **(5%) Banker’s algorithm. Consider the following system with the resources as given in the tables below**



Answer the following questions

1. Calculate and display the Need matrix

Max – Allocation = Need

|  |  |  |  |
| --- | --- | --- | --- |
| Need | | | |
|  | R1 | R2 | R3 |
| P1 | 6 | 1 | 2 |
| P2 | 2 | 3 | 4 |
| P3 | 5 | 0 | 1 |
| P4 | 0 | 2 | 2 |
| P5 | 0 | 0 | 2 |

1. Is the system in safe state? If yes, then show at least one safe sequence of the processes. In addition to the sequence, show how the Available (working array) changes as each process terminates.
   1. Yes, the system is in a safe state. One possible safe sequence would be P5 > P3 > P4 > P1 > P2.
   2. The Available working array is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Available | | | |
|  |  | R1 | R2 | R3 |
| Safe Sequence | Init | 1 | 1 | 6 |
| P5 | 5 | 1 | 8 |
| P3 | 9 | 3 | 13 |
| P4 | 11 | 6 | 16 |
| P1 | 12 | 10 | 21 |
| P2 | 15 | 10 | 22 |

1. Suppose P2 now has the request (3,1,3). Should this request be granted? Why or why not?
   1. Yes, the request should be granted. The current available resources after executing the prior safe sequence will be (15, 10, 22). That’s more than enough resources than are required to run P2’s new request of (3, 1, 3). If the question was intended to ask whether there is a safe sequence given the original scenario if P2’s request is (3,1,3) the answer is still yes, since (3,1,3) is a lighter weight request than (5,3,5) as P2 was initially.
2. **(a) 5% - Consider the following reference string**



Use the algorithms below with the indicated number of frames to calculate the number of faults for each algorithm and number of frames considered. Please note that all the frames are initially empty which will cause a page fault as demonstrated in the class

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Number of frames considered | | | |
|  | 3 | 4 | 5 | 7 |
| FIFO | 18 | 18 | 15 | 8 |
| LRU | 18 | 17 | 16 | 8 |
| Optimal replacement | 14 | 12 | 10 | 8 |

Work for each algorithm is shown below:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FIFO - 3 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 |
| F | F | F |  | F | F |  | F | F | F | F | F | F | F | F | F | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 18 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FIFO - 4 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 |
|  |  |  |  | 3 | 2 |  | 4 | 1 | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 |
| F | F | F |  | F | F |  | F | F | F | F | F | F | F | F | F | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 18 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FIFO - 5 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 |  | 2 |  |  | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 | 8 | 2 | 4 | 3 | 1 | 6 |  | 8 |  |  | 2 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 | 7 | 8 | 2 | 4 | 3 | 1 |  | 6 |  |  | 8 |
|  |  |  |  | 3 | 2 |  | 4 | 1 | 6 | 7 | 8 | 2 | 4 | 3 |  | 1 |  |  | 6 |
|  |  |  |  |  | 3 |  | 2 | 4 | 1 | 6 | 7 | 8 | 2 | 4 |  | 3 |  |  | 1 |
| F | F | F |  | F | F |  | F | F | F | F | F | F | F | F |  | F |  |  | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 15 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| FIFO - 7 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 |  |  |  |  |  |  |  |  |  |  | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 |  |  |  |  |  |  |  |  |  |  | 8 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 |  |  |  |  |  |  |  |  |  |  | 7 |
|  |  |  |  | 3 | 2 |  | 4 | 1 |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  |  | 3 |  | 2 | 4 |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  | 3 | 2 |  |  |  |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  |  |  | 2 |
| F | F | F |  | F | F |  | F | F |  |  |  |  |  |  |  |  |  |  | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 8 |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LRU - 3 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 4 | 4 |  | 4 | 4 | 2 | 2 | 2 | 1 | 1 | 1 | 4 | 4 | 4 | 1 | 1 |
|  | 3 | 2 |  | 2 | 6 |  | 6 | 8 | 8 | 8 | 3 | 3 | 3 | 8 | 8 | 8 | 3 | 3 | 3 |
|  |  | 3 |  | 1 | 1 |  | 7 | 7 | 7 | 4 | 4 | 4 | 6 | 6 | 6 | 2 | 2 | 2 | 5 |
| F | F | F |  | F | F |  | F | F | F | F | F | F | F | F | F | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 18 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| LRU - 4 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 1 |  | 1 | 8 | 8 |  | 8 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 |
|  | 3 | 2 |  | 4 | 4 |  | 4 | 4 | 4 |  | 4 | 4 | 4 | 8 | 8 | 8 | 8 | 1 | 1 |
|  |  | 3 |  | 2 | 2 |  | 7 | 7 | 7 |  | 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 5 |
|  |  |  |  | 3 | 6 |  | 6 | 6 | 2 |  | 2 | 2 | 6 | 6 | 6 | 6 | 3 | 3 | 3 |
| F | F | F |  | F | F |  | F | F | F |  | F | F | F | F | F | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 17 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| LRU - 5 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 6 | 6 | 6 |  | 3 | 3 | 3 | 3 |  | 2 | 2 | 2 | 2 |
|  | 3 | 2 |  | 4 | 1 |  | 1 | 1 | 2 |  | 2 | 2 | 2 | 8 |  | 8 | 8 | 8 | 5 |
|  |  | 3 |  | 2 | 4 |  | 4 | 4 | 4 |  | 4 | 4 | 4 | 4 |  | 4 | 4 | 4 | 4 |
|  |  |  |  | 3 | 2 |  | 2 | 8 | 8 |  | 8 | 8 | 6 | 6 |  | 6 | 6 | 1 | 1 |
|  |  |  |  |  | 3 |  | 7 | 7 | 7 |  | 7 | 1 | 1 | 1 |  | 1 | 3 | 3 | 3 |
| F | F | F |  | F | F |  | F | F | F |  | F | F | F | F |  | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 16 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| LRU - 7 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 |  |  |  |  |  |  |  |  |  |  | 8 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 |  |  |  |  |  |  |  |  |  |  | 5 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  | 3 | 2 |  | 4 | 1 |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  | 3 |  | 2 | 4 |  |  |  |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  | 3 | 2 |  |  |  |  |  |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  |  |  | 3 |
| F | F | F |  | F | F |  | F | F |  |  |  |  |  |  |  |  |  |  | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 8 |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Optimal Replacement - 3 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 4 | 4 |  | 4 | 4 |  |  | 4 | 4 | 4 |  |  | 2 | 3 | 1 | 5 |
|  | 3 | 2 |  | 2 | 2 |  | 2 | 2 |  |  | 3 | 1 | 6 |  |  | 4 | 2 | 3 | 1 |
|  |  | 3 |  | 1 | 6 |  | 7 | 8 |  |  | 8 | 8 | 8 |  |  | 6 | 4 | 2 | 3 |
| F | F | F |  | F | F |  | F | F |  |  | F | F | F |  |  | F | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 14 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Optimal Replacement - 4 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 |  |  |  | 8 | 8 |  |  |  | 3 | 1 | 5 |
|  | 3 | 2 |  | 4 | 4 |  | 4 | 4 |  |  |  | 4 | 4 |  |  |  | 4 | 3 | 1 |
|  |  | 3 |  | 2 | 2 |  | 2 | 2 |  |  |  | 2 | 2 |  |  |  | 2 | 4 | 3 |
|  |  |  |  | 3 | 3 |  | 3 | 3 |  |  |  | 1 | 6 |  |  |  | 6 | 2 | 4 |
| F | F | F |  | F | F |  | F | F |  |  |  | F | F |  |  |  | F | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 12 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Optimal Replacement - 5 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 |  |  |  |  | 8 |  |  |  |  | 1 | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 1 | 1 |  |  |  |  | 6 |  |  |  |  | 8 | 1 |
|  |  | 3 |  | 2 | 4 |  | 4 | 4 |  |  |  |  | 4 |  |  |  |  | 6 | 8 |
|  |  |  |  | 3 | 2 |  | 2 | 2 |  |  |  |  | 2 |  |  |  |  | 4 | 6 |
|  |  |  |  |  | 3 |  | 3 | 3 |  |  |  |  | 3 |  |  |  |  | 2 | 4 |
| F | F | F |  | F | F |  | F | F |  |  |  |  | F |  |  |  |  | F | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 10 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Optimal Replacement - 7 Frames | | | | | | | | | | | | | | | | | | | |
| 3 | 2 | 4 | 2 | 1 | 6 | 4 | 7 | 8 | 2 | 4 | 3 | 1 | 6 | 8 | 4 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 |  | 1 | 6 |  | 7 | 8 |  |  |  |  |  |  |  |  |  |  | 5 |
|  | 3 | 2 |  | 4 | 1 |  | 6 | 7 |  |  |  |  |  |  |  |  |  |  | 8 |
|  |  | 3 |  | 2 | 4 |  | 1 | 6 |  |  |  |  |  |  |  |  |  |  | 7 |
|  |  |  |  | 3 | 2 |  | 4 | 1 |  |  |  |  |  |  |  |  |  |  | 6 |
|  |  |  |  |  | 3 |  | 2 | 4 |  |  |  |  |  |  |  |  |  |  | 1 |
|  |  |  |  |  |  |  | 3 | 2 |  |  |  |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  | 3 |  |  |  |  |  |  |  |  |  |  | 2 |
| F | F | F |  | F | F |  | F | F |  |  |  |  |  |  |  |  |  |  | F |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | Total Faults: 8 |  |  |

**3. (b) 3% - Following is the source code for FIFO replacement algorithm that uses a queue to save the referenced page numbers, replacing the oldest page when the frames are used up. That is, the first one that goes into the queue will be the first one to be removed. The following is an implementation of the straight FIFO algorithm, where the *R* bit is not used.**

// fifo.cpp: First in first out page replacement

// Compile: g++ -o fifo fifo.cpp

// Execute: ./fifo

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<deque>

using namespace std;

class Cframe {

public:

int frameNo; // frame number

int pageNo; // page number

int r; // reference bit

Cframe ( int n, int p ) // constructor

{

frameNo = n;

pageNo = p; // no page loaded at beginning

r = 0;

}

};

deque <Cframe> Q;

int nFaults = 0;

int page, frame;

int displayMsg()

{

cout <<"\npage " << page <<" is allocated to frame " << frame;

return 1;

}

void fault ()

{

nFaults++;

cout<<"\nTotal page faults = " << nFaults;

}

int search ( const deque<Cframe> &q, int p )

{

int n = q.size();

for ( int i = 0; i < n; i++ ){

if ( q[i].pageNo == p )

return q[i].frameNo;

}

return -1;

}

int main()

{

int maxFrames;

cout << "\nEnter max. number of frames allowed in main memory: ";

cin >> maxFrames;

int n;

cout << "\nEnter sequence of page requests (-99 to terminate). ";

while ( true ) {

cout << "\nNew page : ";

cin >> page;

if ( page == -99 )

break;

if (( frame = search ( Q, page )) != -1 ) {

cout << "\npage "<< page << " already in frame " << frame;

} else {

n = Q.size();

if ( n < maxFrames ) {

Cframe aFrame ( n, page ) ;

Q.push\_back ( aFrame );

frame = aFrame.frameNo;

displayMsg ();

} else {

Cframe aFrame = Q.front();

Q.pop\_front();

aFrame.pageNo = page;

Q.push\_back ( aFrame );

frame = aFrame.frameNo;

displayMsg ();

}

fault();

}

} // while

cout << "\nTotal number of faults: " << nFaults << endl;

return 0;

}

The following is a sample input and output of this program:

-------------------------------------------------------------------

Enter max. number of frames allowed in main memory: 3

Enter sequence of page requests (-99 to terminate).

New page: 2

page 2 is allocated to frame 0

Total page faults = 1

New page: 3

page 3 is allocated to frame 1

Total page faults = 2

New page: 2

page 2 already in frame 0

New page: -99

Total number of faults: 2

-------------------------------------------------------------------

1. Compile the above program and run it with the reference string for FIFO that was used in the class. Copy your output as given in the format above.

Using the reference string from page replacement algorithm slides, with 3 frames:

0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

Text

Description automatically generated

1. Try the belady’s anomaly example that was discussed in the class. Compare the page faults resulted from (i) and show that belady’s anomaly occurred.

As shown above, 3 frames results in 9 faults

As shown below, 4 frames results in an increase to 10 faults. An increase in the number of frames resulted in an increase in the number of faults. Therefore, Belady’s anomaly has occurred.

Graphical user interface, text

Description automatically generated

1. **(4%) Socket programming multiple client single server implementation**

We covered an example of socket programming in week 8 where we implemented and tested a chat application between a single client and a single server. The code examples for both the client and server are available via Blackboard (Week 8 of Programming discussions section).

Requirement: Your task is to extend TCP server file and allow multiple client connections (and chats). At present, the ports are hardcoded in the programs. For ease, you can make port input as part of the command line arguments for the clients so that you can avoid hard coding port numbers for each newly created clients. At the minimum, two clients should be able to connect with the server (you can test more than 2 for robustness). Please upload your TCP server and client .cpp (or .c) files and write your answers with screenshots in your report. Your report can indicate a brief description of what parts were extended in the code to allow multiple clients and what is your output (with the help of screenshots). For multiple client example, please refer to the bulb activity that was uploaded in Week 8 folder.

Following are the copies of the code covered in the class:

**Server:**

// g++ -o tcpServer tcpServer.cpp

// ./tcpServer Server

#include <stdio.h>

#include <netdb.h>

#include <netinet/in.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#define MAX 80

#define PORT 54550

#define SA struct sockaddr

// Function designed for chat between client and server.

void openChat(int sockfd, char \*name)

{

    char buff[MAX], clientName[MAX];

    int n;

    // infinite loop for chat

    for (;;) {

        bzero(buff, MAX);

        // read the message from client and copy it in buffer

        read(sockfd, buff, sizeof(buff));

        // print buffer which contains the client contents

        printf("From client: %s \n %s: ", buff, name);

        bzero(buff, MAX);

        n = 0;

        // copy server message in the buffer

        while ((buff[n++] = getchar()) != '\n')

            ;

        // and send that buffer to client

        write(sockfd, buff, sizeof(buff));

    printf("Message sent. Waiting for client\n");

        // if msg contains "Exit" then server exit and chat ended.

        if (strncmp("exit", buff, 4) == 0) {

            printf("Server Exit...\n");

            break;

        }

    }

}

// Driver function

int main(int arc, char \*argv[])

{

    int sockfd, connfd, len;

    struct sockaddr\_in servaddr, cli;

    socklen\_t addr\_size;

    char \*name = argv[1];

    // socket create and verification

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sockfd == -1) {

        printf("socket creation failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully created..\n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    servaddr.sin\_port = htons(PORT);

    // Binding newly created socket to given IP and verification

    if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {

        printf("socket bind failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification

    if ((listen(sockfd, 5)) != 0) {

        printf("Listen failed...\n");

        exit(0);

    }

    else

        printf("Server listening..\n");

    len = sizeof(cli);

    // Accept the data packet from client and verification

    connfd = accept(sockfd, (SA\*)&cli, &addr\_size);

    if (connfd < 0) {

        printf("server acccept failed...\n");

        exit(0);

    }

    else

        printf("server acccepted the client...\n");

    // Open chat

    openChat(connfd, name);

    // After chatting close the socket

    close(sockfd);

}

**Client:**

// g++ -o tcpClient tcpClient.cpp

// ./tcpClient Client

#include <netdb.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define MAX 80

#define PORT 54550

#define SA struct sockaddr

void openChat(int sockfd, char \*name)

{

    char buff[MAX], serverName[MAX];

    int n;

    for (;;) {

        bzero(buff, sizeof(buff));

        printf("%s : ", name);

        n = 0;

        while ((buff[n++] = getchar()) != '\n')

            ;

        write(sockfd, buff, sizeof(buff));

        printf("Message sent. Waiting for server\n");

        bzero(buff, sizeof(buff));

        read(sockfd, buff, sizeof(buff));

        printf("From Server : %s", buff);

        if ((strncmp(buff, "exit", 4)) == 0) {

            printf("Client Exit...\n");

            break;

        }

    }

}

int main(int arc, char \*argv[])

{

    int sockfd, connfd;

    struct sockaddr\_in servaddr, cli;

    char \*name = argv[1];

    // socket create and varification

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sockfd == -1) {

        printf("socket creation failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully created..\n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    servaddr.sin\_port = htons(PORT);

    // Client to server socket connection

    if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr)) != 0) {

        printf("Could not connect with server...\n");

        exit(0);

    }

    else

        printf("connected to the server..\n");

    // open chat

    openChat(sockfd, name);

    // close socket

    close(sockfd);

}

**Modified Source Code:**

* 1. **Server**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

* 1. **Client**

Text

Description automatically generated

Text

Description automatically generated

**Screenshots of output:**

* 1. **Text

     Description automatically generatedServer Starting up with 3 ports running**
  2. **All clients connected**

**Graphical user interface, text

Description automatically generated**

* 1. **All clients chatting**

**A screenshot of a computer

Description automatically generated with medium confidence**

* 1. **All clients exiting**

Text

Description automatically generated

**Report:**

For this modification I decided to multithread the business logic in the server file. I modified the command line arguments to accept port numbers after the server name e.g. ./part4\_server my\_server 54550 54551 54552. The script will allow any number of ports to be added in this manner. Once the program starts, it will create a new thread for every port passed in as a command line argument. The port is passed to each thread when it is implemented, and the original logic that was provided in a main function is moved to the thread logic. Each thread will then listen on its corresponding port. Once the connection for each thread is exited, the threads join before the main program exits. The client file required very little modification. I just added command line specification of the port number rather than hardcoding it. The screenshots show an example of starting a server listening on 3 ports, connecting to all three of them with different clients, and the communication between all 3 clients with the server. Finally, the connections are exited one at a time.

1. **(3%) Implement a synchronized call center (*please also see note at the end for bonus*)**

One of the typical software engineering questions by industrial sectors is to design or implement a solution of their choice using a programming language of your choice. We will go over a famous example which is normally discussed online very often. Therefore, you will find several places for help when you do this exercise. You are welcome to read from external sources including any help for coding solutions. However, please clearly cite any useful resources you will use for this exercise. DO NOT use the same code if you do find it online. Please submit your solution in the form of a program (Java program NOT recommended) for this exercise along with a paragraph of description on how you implemented your solution. Imagine you have a call center with employees as agents to receive calls. An incoming telephone call must be allocated to an employee who is free. The customers making the phone calls to a call center are referred to as threads. Think of the scenario as a producer/consumer where producers are the employees working in the call center and your critical section is the phone call simulated with thread function. Implement a thread-based solution where you have at least 5 customer threads. The names of the customers should be Alice, Bob, John, Mark, Alex. Please note that when a phone call is made (i.e., the critical section is being executed) the critical section should be secured (either with semaphore or mutex lock).

Your sample output should be something like below

Welcome to the call center. Please wait for the setup to start

Initial operator availability = 3

Alice is waiting to speak to the operator...

Bob is waiting to speak to the operator...

John is waiting to speak to the operator...

Mark is waiting to speak to the operator...

Alex is waiting to speak to the operator...

All operators are currently busy assisting other callers - MUSIC ...

Alice is getting the connection to the operator ...

Alex is getting the connection to the operator ...

All operators are currently busy assisting other callers - MUSIC ...

John is getting the connection to the operator ...

All operators are currently busy assisting other callers - MUSIC ...

Alex is now ending the conversation with the operator...

Available operators=1

Mark is getting the connection to the operator ...

John is now ending the conversation with the operator...

Available operators=1

For specific help, please refer to the producer/consumer and thread/process synchronization problems covered in various sessions in our course. Please clearly cite the references with links to any supporting material that you will use outside the class content, including any coding sources. Please visit the following links for more help and getting the ideas of implementation

1. [Java thread version](https://lauraliparulo.altervista.org/java-threads-synchronization-with-semaphore/)
2. [Javascript non-thread example](https://github.com/careercup/CtCI-6th-Edition-JavaScript/blob/ec307944f9c824376ea36bf9d730b0b709b9a923/chapter07/7.02%20-%20Call%20Center/callCenter.js)

**NOTE**: Successful implementation with a working solution will provide a 3% bonus grade of your remaining total grade (i.e., your final exam will be graded for the remaining grade – 3% and you will be given 3% extra for this exercise, otherwise your final exam will graded as normal)

\*\*\* report starts on the next page \*\*\*

Source:

Text

Description automatically generated

Screenshot:

A screenshot of a computer

Description automatically generated with medium confidence

Report:

I chose to write my program in Python to get some more experience using semaphores and threads in python. The only documentation I used for reference can be found at:

<https://docs.python.org/3/library/threading.html>

I used a bounded semaphore and set it to 3 to simulate the total number of operators. The critical section of each thread automatically decrements the bounded semaphore value when it enters the logic and increments the value again when it finishes executing. Before attempting to acquire the semaphore, a check is made against the current bounded semaphore value to decide whether to play hold music. I generated a random number between 3-10 for sleeping to simulate call length. The script was written in Python 3.9.4 and can be executed by running:

python a2\_ec\_call\_center.py