**State University of New York at New Paltz**

**Student Name: Antonio Finelli**

**OS Lab section: 02 (Friday, 2:00-4:50)**

**Semester: Spring 2022**

**REPORT for LAB # 10**

**“Operating Systems” Spring 2022**

**(Professor Hanh Pham)**

**Table of Contents:**

3: Algorithms

4, 5,6,7: Implementation

8, 9, 10: Experiments

11 : Conclusions, References

# Algorithms

FF Memory Allocation Method :

|  |  |
| --- | --- |
| Step # | Action |
| 1 | Check if memory slot is has enough room for incoming process. If false, move to step 2 |
| 2 | Assign process to memory |
| 3 | If no slot satisfies steps 1, the process will not be assigned to memory |

BF Memory Allocation Method :

|  |  |
| --- | --- |
| Step # | Action |
| 1 | I Loop On Process List |
| 2 | J Loop On Memory List |
| 3 | Initialize currentBest (The current best fit calculated by memory size – process size) to 99999 Initialize currBestIndext(Index currently the best fit) to -1 |
| 4 | Check if the traversed memory block has enough room to fit incoming process |
| 5 | Check if current memory block is a better fit than the previous best fit (memory size – process size) by comparing it to the previous current best. Also check if this size is larger than zero. This prevents a value under zero from being selected since it is technically smallest. |
| 6 | If steps 4 and 5 are good, set the current best to the new current best and set currBestIndex to the j of the loop (current memory block being traversed). |
| 7 | Occurs after each iteration of the I loop. If the current best index is -1, add it to the list of rejected processes. If current best index is not -1, add the process to the memory block indicated by current best index. |

WF Memory Allocation Method :

|  |  |
| --- | --- |
| Step # | Action |
| 1 | I Loop On Process List |
| 2 | J Loop On Memory List |
| 3 | Initialize currentBest (The current worst fit calculated by memory size – process size) to 0 Initialize currBestIndext(Index currently the best fit) to -1 |
| 4 | Check if the traversed memory block has enough room to fit incoming process |
| 5 | Check if current memory block is a worse than the previous worse fit (memory size – process size) by comparing it to the previous current worst. Also check if this size is larger than zero. This prevents a value under zero from being selected since it is technically smallest. |
| 6 | If steps 4 and 5 are good, set the current best to the new current best and set currBestIndex to the j of the loop (current memory block being traversed). |
| 7 | Occurs after each iteration of the I loop. If the current best index is -1, add it to the list of rejected processes. If current best index is not -1, add the process to the memory block indicated by current best index. |

# Implementation

Project Structure :

Overall, the project uses two major classes. A memory block, and a process. The memory block allows for start and end locations, and the ability to hold processes. A process contains a size and ID, and is added to the memory blocks.

Classes:

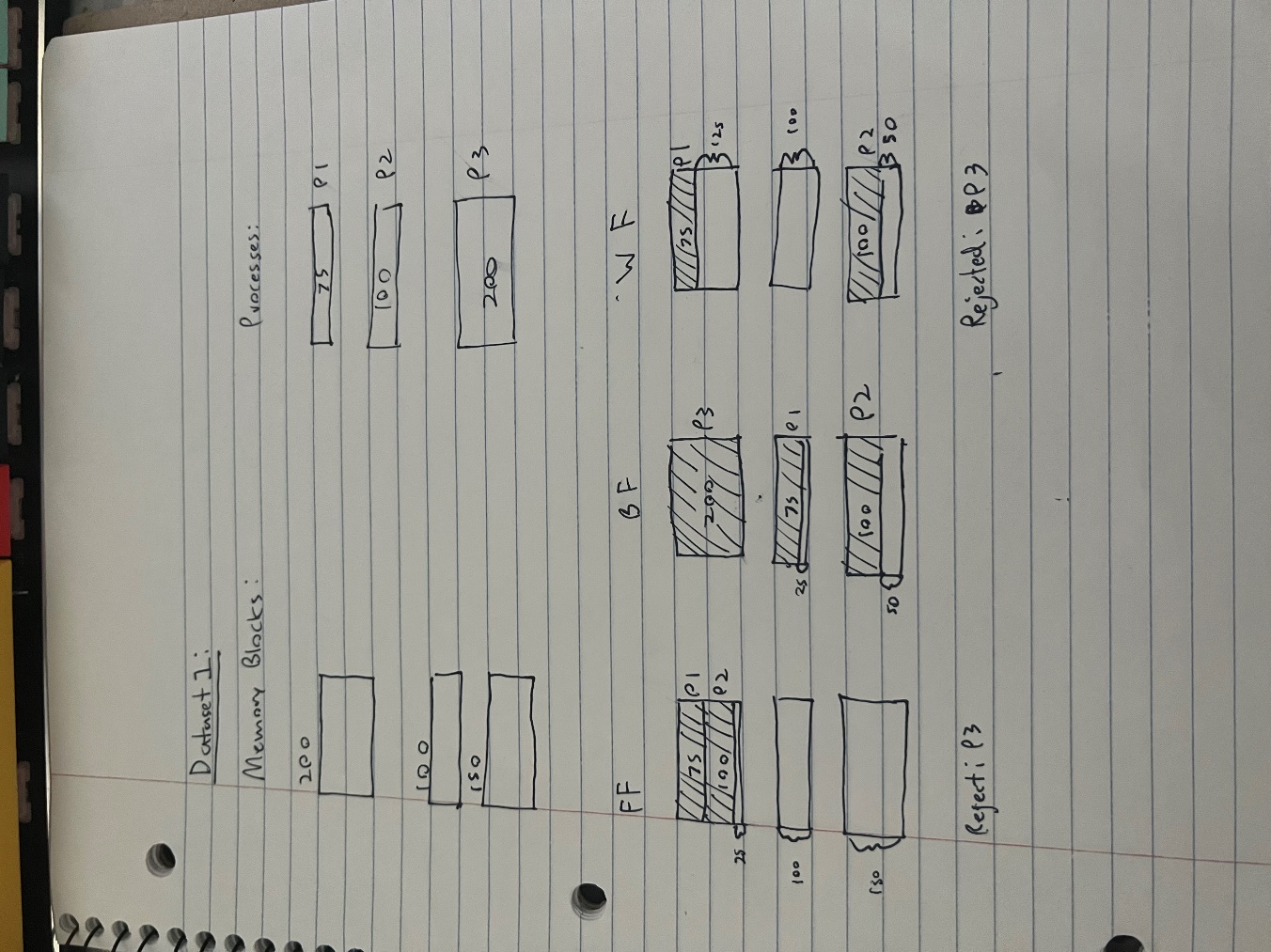
* Class
  + Variable
  + Method
* memBlock: Block containing memory. A slot for processes
  + memory int[2] : Stores memory address in format [startAddress,endAddress]
  + LinkedList<process> pList : linked list to hold all processes contained in the memory block
  + memBlock(int, int) // value constructor
  + int getSize() // Returns the overall size of the memory block (start – end)
  + void setStartAddress(int) // setter for start address
  + void setEndAddress(int) // setter for end address
  + int getStartAddress() // getter for start address
  + int setEndAddress() // getter for end address
  + LinkedList<process> getProcessList() // getter for process list
  + Void addProcess(process) // adds process to process list
  + String toString // general to string
  + Int getTotalOccupiedSpace() // Returns the total size of every process in pList
* Process : process and related information
  + Int id // the id of the process
  + Int size // the size of the process
  + Process(int, int) // value constructor
  + Void setID(int) // id setter
  + int getID() // id getter
  + void setSize (int) // size setter
  + int getSize(int) // size getter
  + String toString // general to string

Functions :

* Main
  + The main function of the program. Calls getMinput and getProcesses and displays results to screen. Also calls ff,bf,and wf functions.
* getMinput
  + reads in info from file
  + First row is the length of the memory array
  + Memory array is filled with the rest of the data split by space
* getProcesses
  + same as getMinput, but for processes.
* Ff
  + The main ff function.
  + Uses IJ loop to the following effect :
    - I loop iterates through processes
    - J loop iterates through memory slots
    - Each process will check if a slot meets the follow criteria:
      * Slot empty
      * Slot big enough
    - If both are correct, the process will be added to the process list of that memory slot
    - If no slot is found, the process is added to the rejectedProcesses linked list.
  + After all of the above is done, the result is neatly written to the file with the end location being the memory start address added to the previous process size
* BF
  + The main bf function.
  + Uses IJ loop to the following effect :
    - I loop iterates through processes
    - J loop iterates through memory slots
    - Each process will check if a slot meets the follow criteria:
      * Slot has enough room to fit incoming process
      * Slot best fit
        + Determined by checking if memory – process size < current best size
    - If both are correct, the process will be added to the process list of memory slot at best index
    - If no slot is found (indicated when best index is -1), the process is added to the rejectedProcesses linked list.
  + After all of the above is done, the result is neatly written to the file with the end location being the memory start address added to the previous process size
* WF
  + The main WF function.
  + Uses IJ loop to the following effect :
    - I loop iterates through processes
    - J loop iterates through memory slots
    - Each process will check if a slot meets the follow criteria:
      * Slot has enough room to fit incoming process
      * Slot best fit
        + Determined by checking if memory – process size > current worst size
    - If both are correct, the process will be added to the process list of memory slot at best index
    - If no slot is found (indicated when best index is -1), the process is added to the rejectedProcesses linked list.
  + After all of the above is done, the result is neatly written to the file with the end location being the memory start address added to the previous process size

# Experiments :

***Dataset 1 :***



**Inputs :**

Minput1 :

3

100 300

600 700

1500 1650

P Input 1:

3

1 75

2 100

3 200

**Expected Outputs:**

FF:

100 175 1

175 275 2

-3

BF :

100 300 3

600 675 1

1500 1600 2

-0

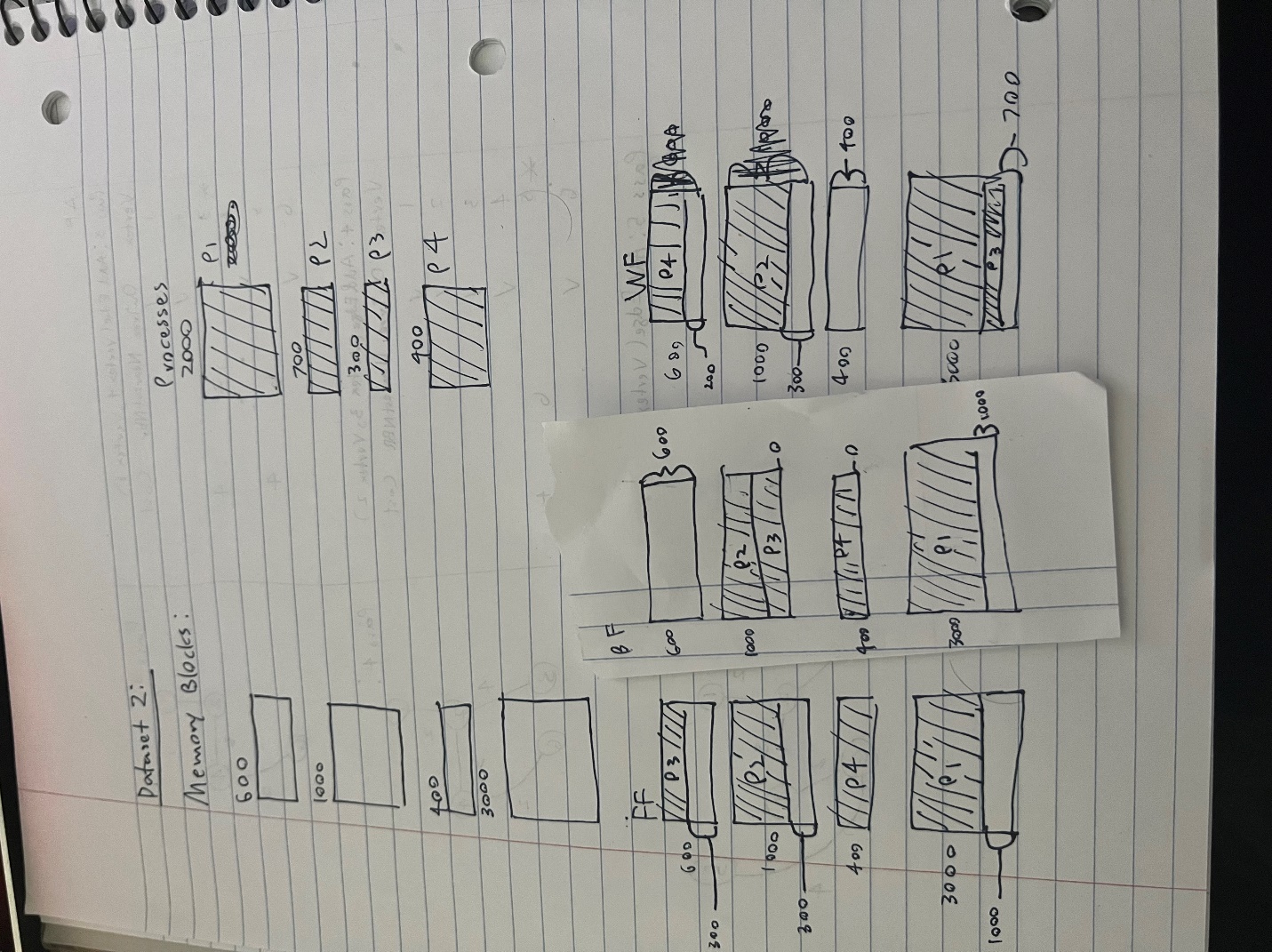
WF :

100 175 1

1500 1600 2

-3

***Data Set 2:***



**Inputs :**

Pinput 2:

4

1 2000

2 700

3 300

4 200

Minput 2:

4

100 700

800 1800

1900 2300

2400 5400

**Expected Outputs :**

FF :

100 400 3

800 1500 2

1900 2300 4

2400 4400 1

-0

BF :

800 1500 2

1500 1800 3

1900 2300 4

2400 4400 1

-0

WF:

100 500 4

800 1500 2

2400 4400 1

4400 4700 3

-0

# Conclusions :

From this lab, I can conclude a few key points. Firstly, among the three datasets that I tested; the odds of a process not being allocated are lowest with the Best First algorithm. In my testing, I found that the First Fit and Worst Fit were roughly equal in terms of chance of rejecting a process for lack of space.

I also noticed that Best Fit seemed to have the least fragmentation when compared to First and Worst Fit. I believe that this is because the best fit will always find the smallest gap instead of simply placing it into the first gap available.

For clarification of my results, for the WF algorithm I created, the processes are being stored correctly, but the output of the memory locations to the file is incorrect. It is important to note that despite this the process IDs prove that they are being stored correctly.

# References :

Information for file read in from <https://www.w3schools.com/java/java_files_read.asp>

File writing code from https://www.baeldung.com/java-write-to-file