5(b). Memory Allocation Strategies

Aim:

To implement various memory allocation strategies like the first fit, best fit, and worst fit strategy.

Theory:

In Partition Allocation, when there are more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.
Below are the various partition allocation schemes :

1. First Fit
In the first fit strategy, the first partition, sufficient to hold a process, from the top of Main Memory, is allocated to the process.
Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.
It may have problems of not allowing processes to take space even if it was possible to allocate.
Consider the below example, process number 4 (of size 426) does not get memory.

```
Input: blockSize[]   = {100, 500, 200, 300, 600};
       processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
   1               212           2
   2               417           5
   3               112           2
   4               426        Not Allocated
```

```
Implementation:
1- Input memory blocks with size and processes with size.
2- Initialize all memory blocks as free.
3- Start by picking each process and check if it can
   be assigned to current block.
4- If size-of-process <= size-of-block if yes then
   assign and check for next process.
5- If not then keep checking the further blocks.
```

2. Best Fit

Best fit allocates the process to a partition which is the smallest sufficient partition among the free available partitions.
Although, best fit minimizes the wastage space, it consumes a lot of processor time for searching the block which is close to required size. Also, Best-fit may perform poorer than other algorithms in some cases. For example, see below exercise.

Example: Consider the requests from processes in given order 300K, 25K, 125K and 50K. Let there be two blocks of memory available of size 150K followed by a block size 350K.

Best Fit:

300K is allocated from block of size 350K. 50 is left in the block.
25K is allocated from the remaining 50K block. 25K is left in the block.
125K is allocated from 150 K block. 25K is left in this block also.
50K can't be allocated even if there is 25K + 25K space available.

First Fit:

300K request is allocated from 350K block, 50K is left out.
25K is be allocated from 150K block, 125K is left out.
Then 125K and 50K are allocated to remaining left out partitions.
So, first fit can handle requests.

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
 1             212             4
 2             417             2
 3             112             3
 4             426             5
```

```
Implementation:
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the minimum block size that can
be assigned to current process i.e., find min(bockSize[1],
blockSize[2],.....blockSize[n]) > processSize[current], if found then
assign it to the current process.
4- If not then leave that process and keep checking the further processes.
```

3. Worst Fit

Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.

Although worst fit strategy is easy to implement, a lot of storage space may be wasted and it consumes additional time that is required for sorting the available partitions and searching for the free ones.

```
Input : blockSize[]   = {100, 500, 200, 300, 600};
        processSize[] = {212, 417, 112, 426};
Output:
Process No.    Process Size    Block no.
   1           212             5
   2           417             2
   3           112             5
   4           426             Not Allocated
```

```
Implementation:
1- Input memory blocks and processes with sizes.
2- Initialize all memory blocks as free.
3- Start by picking each process and find the maximum block size that can
be assigned to current process i.e., find max(bockSize[1],
blockSize[2],.....blockSize[n]) > processSize[current], if found then
assign it to the current process.
4- If not then leave that process and keep checking the further processes.
```

Program:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Comparator;

/**
 * JAVA class to demonstrate the dynamic memory
 * allocation techniques.
 *
 * @author jimil
 */
public class DynamicMemoryAllocationImpl {
    static ArrayList<Process> processes = new ArrayList<Process>();
    static ArrayList<MemoryBlock> blocks = new ArrayList<MemoryBlock>();

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        /**
         * Take process details as input from
         * the user.
         */
        System.out.print("Enter the number of processes: ");
        int processCount = Integer.parseInt(br.readLine());
        System.out.println();
        System.out.println("Enter the process details: ");
        for(int i = 0; i < processCount; i++) {
            Process p = new Process();
            p.setId(i);
            System.out.print("Enter the process' memory request size: ");
            p.setMemoryRequestSize(Integer.parseInt(br.readLine()));
            processes.add(p);
            System.out.println();
        }
        /**
         * Take details of the memory blocks as input
         * from the user.
         */
        System.out.print("Enter the number of memory blocks: ");
        int blocksCount = Integer.parseInt(br.readLine());
```

```
        System.out.println();
        System.out.println("Enter the memory block details: ");
        for(int i = 0; i < blocksCount; i++) {
            MemoryBlock mb = new MemoryBlock();
            mb.setId(i);
            System.out.print("Enter the size of the memory block: ");
            mb.setSize(Integer.parseInt(br.readLine()));
            mb.setAllocatedProcessId(-10);
            blocks.add(mb);
            System.out.println();
        }
        /**
         * Take choice of algorithm as input from the user.
         */
        System.out.println("1.            First Fit");
        System.out.println("2.            Best Fit");
        System.out.println("3.            Worst Fit");
        System.out.println("Which algorithm do you want to implement?");
        int choice = Integer.parseInt(br.readLine());
        switch(choice) {
            case 1: doFirstFit();
                    printDetails();
                    release(choice);
                break;
            case 2: doBestFit();
                    printDetails();
                    release(choice);
                break;
            case 3: doWorstFit();
                    printDetails();
                    release(choice);
                break;
            default: System.out.println("Please Enter a valid choice!");
                break;
        }
    }

    /**
     * Default implementation to perform dynamic memory
     * allocation using first fit algorithm.
     * The algorithm iterates over the list of processes
     * and does the following steps:
```

```
     *   1. For each process, iterate over the memory blocks
     *      from the beginning
     *   2. The first block that is found to be free and of size
     *      greater than the process' requested size, do the following:
     *      2.1 Update the block's allocation status to true
     *      2.2 Update the process' PCB to indicate the block it got
allocated
     *      2.3 Update the memory block's descriptor to indicate the
process that is
     *            currently residing in it.
     *      2.4 New size of memory block =
     *                  size of memory block - process' requested memory
size
     */
    public static void doFirstFit() {
        for(Process p: processes) {
            if(p.isAllocated() == false) {
                System.out.println("Searching for memory of size "
                        + p.getMemoryRequestSize() + " for process P" +
p.getId());
                boolean flag = false;
                for(MemoryBlock mb: blocks) {
                    if(mb.getSize() >= p.getMemoryRequestSize()) {
                        mb.setAllocatedProcessId(p.getId());
                        p.setMemoryBlockIdAllocated(mb.getId());
                        p.setAllocated(true);
                        mb.setSize(mb.getSize() -
p.getMemoryRequestSize());
                        System.out.println("Process P" + p.getId() + "
allocated to memory block B"
                                + mb.getId());
                        System.out.println("New Partition created: B" +
mb.getId()
                                + " of size: " + mb.getSize());
                        flag = true;
                        break;
                    }
                }
                if(flag == false) {
                    System.out.println("Process P" + p.getId() + " has to
wait since no memory block is available for allocation!");
                }
```

```
            }
        }
    }

    /**
     * Releases the memory held by a particular process
     *
     * @param pid
     */
    public static void releaseProcess(int pid) {
        if(processes.get(pid).isAllocated()) {
            Process p = processes.get(pid);
            if(p != null) {
                int blockId = p.getMemoryBlockIdAllocated();
                int index = -10;
                for(MemoryBlock mb: blocks) {
                    if(mb.getId() == blockId) {
                          index = blocks.indexOf(mb);
                          break;
                    }
                }
                if(index >= 0) {
                    int newSize = blocks.get(index).getSize() +
p.getMemoryRequestSize();
                    blocks.get(index).setSize(newSize);
                    blocks.get(index).setAllocated(false);
                }
            }
        }
    }

    /**
     * Whether to release a process and if yes,
     * release the process and run algorithm again.
     *
     * @throws Exception
     */
    public static void release(int algo) throws Exception {
        System.out.println("Do you want to release any process?");
        System.out.println("Enter 1. for Yes and 2. for No");
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

```
        int ch = Integer.parseInt(br.readLine());
        if(ch == 1) {
            System.out.print("Enter ID of process to release: ");
            int pid = Integer.parseInt(br.readLine());
            releaseProcess(pid);
            switch(algo) {
                case 1: doFirstFit();
                    break;
                case 2: doBestFit();
                    break;
                case 3: doWorstFit();
                    break;
            }
        }
    }

    /**
     * Default implementation to perform dynamic memory
     * allocation using worst fit algorithm.
     * The algorithm iterates over the list of processes
     * and does the following steps:
     *   1. Arrange the memory blocks in decreasing order of
     *      their size.
     *   2. For each process, do the following:
     *      2.1 Allocate the next largest block to this process
     *      2.2 Update process' PCB to indicate the memory block
     *          it is allocated.
     *      2.3 Update memory block's descriptor to indicate the
     *          process currently residing in it.
     *      2.4 Update size of the memory block =
     *              size of memory block - process' requested memory size
     */
    public static void doWorstFit() {
        blocks.sort(new Comparator<MemoryBlock>() {
            @Override
            public int compare(MemoryBlock o1, MemoryBlock o2) {
                return o2.getSize() - o1.getSize();
            }
        });
        for(int i = 0; i <= processes.size()-1; i++) {
            Process p = processes.get(i);
            if(!p.isAllocated()) {
```

```
                System.out.println("Searching for memory of size "
                        + p.getMemoryRequestSize() + " for process P" +
p.getId());
                boolean flag = false;
                for(int j = 0; j < blocks.size(); j++) {
                    MemoryBlock mb = blocks.get(j);
                    if(!mb.isAllocated() && mb.getSize() >=
p.getMemoryRequestSize()) {
                        mb.setAllocatedProcessId(p.getId());
                        p.setMemoryBlockIdAllocated(mb.getId());
                        p.setAllocated(true);
                        mb.setAllocated(true);
                        mb.setSize(mb.getSize() -
p.getMemoryRequestSize());
                        System.out.println("Process P" + p.getId() + "
allocated to memory block B"
                                + mb.getId());
                        System.out.println("New Partition created: B" +
mb.getId()
                                + " of size: " + mb.getSize());
                        flag = true;
                        break;
                    }
                }
                if(!flag) {
                    System.out.println("Process P" + p.getId() + " has to
wait since no memory block is available for allocation!");
                }
            }
        }
    }

    /**
     * Default implementation to perform dynamic memory
     * allocation using best fit algorithm.
     * The algorithm iterates over the list of processes
     * and does the following steps:
     *   1. Arrange the memory blocks in increasing order of
     *      their size.
     *   2. For each process, do the following:
     *      2.1 Allocate the next optimum block to this process
     *      2.2 Update process' PCB to indicate the memory block
```

```
     *          it is allocated.
     *      2.3 Update memory block's descriptor to indicate the
     *          process currently residing in it.
     *      2.4 Update size of the memory block =
     *              size of memory block - process' requested memory size
     */
    public static void doBestFit() {
        blocks.sort(new Comparator<MemoryBlock>() {
            @Override
            public int compare(MemoryBlock o1, MemoryBlock o2) {
                return o1.getSize() - o2.getSize();
            }
        });
        for(int i = 0; i <= processes.size()-1; i++) {
            Process p = processes.get(i);
            if(!p.isAllocated()) {
                System.out.println("Searching for memory of size "
                        + p.getMemoryRequestSize() + " for process P" +
p.getId());
                boolean flag = false;
                for(int j = 0; j < blocks.size(); j++) {
                    MemoryBlock mb = blocks.get(j);
                    if(!mb.isAllocated() && mb.getSize() >=
p.getMemoryRequestSize()) {
                        mb.setAllocatedProcessId(p.getId());
                        p.setMemoryBlockIdAllocated(mb.getId());
                        p.setAllocated(true);
                        mb.setAllocated(true);
                        mb.setSize(mb.getSize() -
p.getMemoryRequestSize());
                        System.out.println("Process P" + p.getId() + "
allocated to memory block B"
                                + mb.getId());
                        System.out.println("New Partition created: B" +
mb.getId()
                                + " of size: " + mb.getSize());
                        flag = true;
                        break;
                    }
                }
                if(!flag) {
                    System.out.println("Process P" + p.getId() + " has to
```

```
wait since no memory block is available for allocation!");
                }
            }
        }
    }

    /**
     * Prints memory status after algorithm is run.
     */
    public static void printDetails() {
        System.out.println("Dynamic Memory Allocation Details: ");
        System.out.println("-----------------------------------");
        System.out.println("Block ID\tSize\tProcess Allocated");
        for(MemoryBlock mb: blocks) {
            if(mb.getAllocatedProcessId() >= 0) {
                System.out.print(mb.getId() + "\t\t" + mb.getSize() +
"\t\t" + mb.getAllocatedProcessId());
                System.out.println();
            }
        }
    }
}

/**
 * A POJO class that denotes a process in
 * memory
 *
 * @author jimil
 */
class Process {
    int id;
    int memoryRequestSize;
    boolean isAllocated;
    int memoryBlockIdAllocated;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getMemoryRequestSize() {
        return memoryRequestSize;
```

```
    }
    public void setMemoryRequestSize(int memoryRequestSize) {
        this.memoryRequestSize = memoryRequestSize;
    }
    public boolean isAllocated() {
        return isAllocated;
    }
    public void setAllocated(boolean isAllocated) {
        this.isAllocated = isAllocated;
    }
    public int getMemoryBlockIdAllocated() {
        return memoryBlockIdAllocated;
    }
    public void setMemoryBlockIdAllocated(int memoryBlockIdAllocated) {
        this.memoryBlockIdAllocated = memoryBlockIdAllocated;
    }
}

/**
 * A POJO class that denotes a block in
 * memory.
 *
 * @author jimil
 */
class MemoryBlock {
    int id;
    int size;
    int allocatedProcessId;
    boolean isAllocated;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public int getSize() {
        return size;
    }
    public void setSize(int size) {
        this.size = size;
    }
    public int getAllocatedProcessId() {
```

```
        return allocatedProcessId;
    }
    public void setAllocatedProcessId(int allocatedProcessId) {
        this.allocatedProcessId = allocatedProcessId;
    }
    public boolean isAllocated() {
        return isAllocated;
    }
    public void setAllocated(boolean allocated) {
        isAllocated = allocated;
    }
}
```

Output:

First-Fit Strategy:

```
Enter the number of processes: 4
Enter the process details:
Enter the process' memory request size: 212
Enter the process' memory request size: 417
Enter the process' memory request size: 112
Enter the process' memory request size: 426

Enter the number of memory blocks: 5
Enter the memory block details:
Enter the size of the memory block: 100
Enter the size of the memory block: 500
Enter the size of the memory block: 200
Enter the size of the memory block: 300
Enter the size of the memory block: 600

1.          First Fit
2.          Best Fit
3.          Worst Fit
Which algorithm do you want to implement? 1

Searching for memory of size 212 for process P0
Process P0 allocated to memory block B1
New Partition created: B1 of size: 288
Searching for memory of size 417 for process P1
Process P1 allocated to memory block B4
```

```
New Partition created: B4 of size: 183
Searching for memory of size 112 for process P2
Process P2 allocated to memory block B2
New Partition created: B2 of size: 88
Searching for memory of size 426 for process P3
Process P3 has to wait since no memory block is available for allocation!

Dynamic Memory Allocation Details:
-------------------------------------------------
Block ID    Size  Process Allocated
1           500         0
2           200         2
4           600         1

Do you want to release any process?
Enter 1. for Yes and 2. for No: 1
Enter ID of process to release: 0
Searching for memory of size 426 for process P3
Process P3 allocated to memory block B1
New Partition created: B1 of size: 712

Process finished with exit code 0
```

Best-Fit Strategy:

```
Enter the number of processes: 4
Enter the process details:
Enter the process' memory request size: 212
Enter the process' memory request size: 417
Enter the process' memory request size: 112
Enter the process' memory request size: 426

Enter the number of memory blocks: 5
Enter the memory block details:
Enter the size of the memory block: 100
Enter the size of the memory block: 500
Enter the size of the memory block: 200
Enter the size of the memory block: 300
Enter the size of the memory block: 600

1.          First Fit
2.          Best Fit
```

```
3.          Worst Fit
Which algorithm do you want to implement? 2

Searching for memory of size 212 for process P0
Process P0 allocated to memory block B3
New Partition created: B3 of size: 88
Searching for memory of size 417 for process P1
Process P1 allocated to memory block B1
New Partition created: B1 of size: 83
Searching for memory of size 112 for process P2
Process P2 allocated to memory block B2
New Partition created: B2 of size: 88
Searching for memory of size 426 for process P3
Process P3 allocated to memory block B4
New Partition created: B4 of size: 174

Dynamic Memory Allocation Details:
-----------------------------------
Block ID    Size  Process Allocated
2           88          2
3           88          0
1           83          1
4           174         3

Do you want to release any process?
Enter 1. for Yes and 2. for No 2

Process finished with exit code 0
```

Worst-Fit Strategy:

```
Enter the number of processes: 4
Enter the process details:
Enter the process' memory request size: 212
Enter the process' memory request size: 417
Enter the process' memory request size: 112
Enter the process' memory request size: 426

Enter the number of memory blocks: 5
Enter the memory block details:
Enter the size of the memory block: 100
Enter the size of the memory block: 500
```

```
Enter the size of the memory block: 200
Enter the size of the memory block: 300
Enter the size of the memory block: 600

1.          First Fit
2.          Best Fit
3.          Worst Fit
Which algorithm do you want to implement? 3

Searching for memory of size 212 for process P0
Process P0 allocated to memory block B4
New Partition created: B4 of size: 388
Searching for memory of size 417 for process P1
Process P1 allocated to memory block B1
New Partition created: B1 of size: 83
Searching for memory of size 112 for process P2
Process P2 allocated to memory block B3
New Partition created: B3 of size: 188
Searching for memory of size 426 for process P3
Process P3 has to wait since no memory block is available for allocation!

Dynamic Memory Allocation Details:
-----------------------------------
Block ID    Size  Process Allocated
4           388         0
1           83          1
3           188         2

Do you want to release any process?
Enter 1. for Yes and 2. for No: 1
Enter ID of process to release: 0
Searching for memory of size 426 for process P3
Process P3 allocated to memory block B4
New Partition created: B4 of size: 174

Process finished with exit code 0
```

Conclusion:

Thus, through this experiment, I understood and implemented the various memory allocation strategies i.e. the first fit, the best fit, and the worst fit strategies.