**U19EC046 | OS | LAB 6**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**//FIRSTFIT.JAVA**

**package OS.lab6;**

**import java.util.LinkedList;**

**import java.util.Scanner;**

**public class firstFit {**

**private static LinkedList<Block> memory;**

**private static class Block {**

**public int startAddress;**

**public int endAddress;**

**public int size;**

**public String PID;**

**public Boolean free;**

**Block(int s, int e, String PID) {**

**startAddress = s;**

**endAddress = e;**

**this.PID = PID;**

**size = e - s + 1;**

**free = "free".equals(PID.toLowerCase()) ? true : false;**

**}**

**}**

**static void allocateBlock(String PID, int size, int index) {**

**Block removed = memory.remove(index);**

**LinkedList<Block> \_new = new LinkedList<>();**

**\_new.add(new Block(removed.startAddress, removed.startAddress + size - 1, PID));**

**if (removed.size != size)**

**\_new.add(new Block(removed.startAddress + size, removed.endAddress, "FREE"));**

**memory.addAll(index, \_new);**

**}**

**static void allocateProcessFirstFit(String PID, int processSize) {**

**int i = 0;**

**for (Block current : memory) {**

**if (current.free && current.size >= processSize) {**

**allocateBlock(PID, processSize, i);**

**return;**

**}**

**i++;**

**}**

**System.out.println(">>> No free space available of size " + processSize + " for " + PID + " <<<");**

**}**

**static void mergeHoles() {**

**for (int i = 0; i < memory.size() - 1; i++) {**

**Block current = memory.get(i);**

**Block next = memory.get(i + 1);**

**if (current.free && next.free) {**

**memory.set(i, new Block(current.startAddress, next.endAddress, "FREE"));**

**memory.remove(next);**

**i--;**

**}**

**}**

**}**

**static void deallocateProcess(String PID) {**

**int i = 0;**

**for (Block current : memory) {**

**if (PID.equals(current.PID)) {**

**current.PID = "FREE";**

**current.free = true;**

**mergeHoles();**

**break;**

**}**

**i++;**

**}**

**}**

**static void printMemory() {**

**String leftAlignFormat = "| %-7s | %-5d | %-5d | %-5d |%n";**

**System.out.println("---------- Memory Table ----------");**

**System.out.format("+---------+-------+-------+-------+%n");**

**System.out.format("| PID     | Start | End   | Size  |%n");**

**System.out.format("+---------+-------+-------+-------+%n");**

**for (Block curr : memory) {**

**System.out.format(leftAlignFormat, curr.PID, curr.startAddress, curr.endAddress, curr.size);**

**}**

**System.out.format("+---------+-------+-------+-------+%n");**

**}**

**static void initializeMemory() {**

**memory = new LinkedList<>();**

**memory.add(new Block(0, 7, "P1"));**

**memory.add(new Block(8, 12, "FREE"));**

**memory.add(new Block(13, 24, "P2"));**

**memory.add(new Block(25, 64, "FREE"));**

**memory.add(new Block(65, 80, "P3"));**

**memory.add(new Block(81, 100, "P4"));**

**memory.add(new Block(101, 127, "FREE"));**

**}**

**public static void main(String[] args) {**

**Scanner s = new Scanner(System.in);**

**String PID;**

**int processSize;**

**initializeMemory();**

**printMemory();**

**while (true) {**

**System.out.println(**

**"Enter 1 to allocate a process\nEnter 2 to deallocate a process\nEnter 0 to exit\nYour choice : ");**

**int option = Integer.parseInt(s.nextLine());**

**switch (option) {**

**case 1:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**System.out.println("Enter process size");**

**processSize = Integer.parseInt(s.nextLine());**

**allocateProcessFirstFit(PID, processSize);**

**printMemory();**

**break;**

**case 2:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**deallocateProcess(PID);**

**printMemory();**

**break;**

**default:**

**return;**

**}**

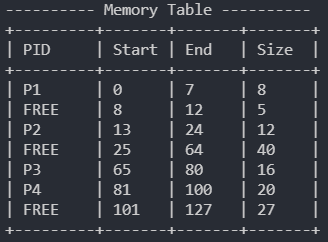
**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**OUTPUT**



|  |  |
| --- | --- |
|  |  |
|  |  |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**//BESTFIT.JAVA**

**package OS.lab6;**

**import java.util.LinkedList;**

**import java.util.Scanner;**

**public class bestFit {**

**private static LinkedList<Block> memory;**

**private static class Block {**

**public int startAddress;**

**public int endAddress;**

**public int size;**

**public String PID;**

**public Boolean free;**

**Block(int s, int e, String PID) {**

**startAddress = s;**

**endAddress = e;**

**this.PID = PID;**

**size = e - s + 1;**

**free = "free".equals(PID.toLowerCase()) ? true : false;**

**}**

**}**

**static void allocateBlock(String PID, int size, int index) {**

**Block removed = memory.remove(index);**

**LinkedList<Block> \_new = new LinkedList<>();**

**\_new.add(new Block(removed.startAddress, removed.startAddress + size - 1, PID));**

**if (removed.size != size)**

**\_new.add(new Block(removed.startAddress + size, removed.endAddress, "FREE"));**

**memory.addAll(index, \_new);**

**}**

**static void allocateProcessBestFit(String PID, int processSize) {**

**int i = 0;**

**int minIndex = -1;**

**int minSize = Integer.MAX\_VALUE;**

**for (Block current : memory) {**

**if (current.free && current.size >= processSize && current.size < minSize) {**

**minSize = current.size;**

**minIndex = i;**

**}**

**i++;**

**}**

**if(minIndex == -1)**

**System.out.println(">>> No free space available of size " + processSize + " for " + PID + " <<<");**

**else**

**allocateBlock(PID, processSize, minIndex);**

**}**

**static void mergeHoles() {**

**for (int i = 0; i < memory.size() - 1; i++) {**

**Block current = memory.get(i);**

**Block next = memory.get(i + 1);**

**if (current.free && next.free) {**

**memory.set(i, new Block(current.startAddress, next.endAddress, "FREE"));**

**memory.remove(next);**

**i--;**

**}**

**}**

**}**

**static void deallocateProcess(String PID) {**

**int i = 0;**

**for (Block current : memory) {**

**if (PID.equals(current.PID)) {**

**current.PID = "FREE";**

**current.free = true;**

**mergeHoles();**

**break;**

**}**

**i++;**

**}**

**}**

**static void printMemory() {**

**String leftAlignFormat = "| %-7s | %-5d | %-5d | %-5d |%n";**

**System.out.println("---------- Memory Table ----------");**

**System.out.format("+---------+-------+-------+-------+%n");**

**System.out.format("| PID     | Start | End   | Size  |%n");**

**System.out.format("+---------+-------+-------+-------+%n");**

**for (Block curr : memory) {**

**System.out.format(leftAlignFormat, curr.PID, curr.startAddress, curr.endAddress, curr.size);**

**}**

**System.out.format("+---------+-------+-------+-------+%n");**

**}**

**static void initializeMemory() {**

**memory = new LinkedList<>();**

**memory.add(new Block(0, 7, "P1"));**

**memory.add(new Block(8, 12, "FREE"));**

**memory.add(new Block(13, 24, "P2"));**

**memory.add(new Block(25, 64, "FREE"));**

**memory.add(new Block(65, 80, "P3"));**

**memory.add(new Block(81, 100, "P4"));**

**memory.add(new Block(101, 127, "FREE"));**

**}**

**public static void main(String[] args) {**

**Scanner s = new Scanner(System.in);**

**String PID;**

**int processSize;**

**initializeMemory();**

**printMemory();**

**while (true) {**

**System.out.println(**

**"Enter 1 to allocate a process\nEnter 2 to deallocate a process\nEnter 0 to exit\nYour choice : ");**

**int option = Integer.parseInt(s.nextLine());**

**switch (option) {**

**case 1:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**System.out.println("Enter process size");**

**processSize = Integer.parseInt(s.nextLine());**

**allocateProcessBestFit(PID, processSize);**

**printMemory();**

**break;**

**case 2:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**deallocateProcess(PID);**

**printMemory();**

**break;**

**default:**

**return;**

**}**

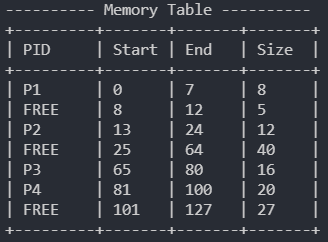
**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**OUTPUT**



|  |  |
| --- | --- |
|  |  |
|  |  |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**//WORSTFIT.JAVA**

**package OS.lab6;**

**import java.util.LinkedList;**

**import java.util.Scanner;**

**public class worstFit {**

**private static LinkedList<Block> memory;**

**private static class Block {**

**public int startAddress;**

**public int endAddress;**

**public int size;**

**public String PID;**

**public Boolean free;**

**Block(int s, int e, String PID) {**

**startAddress = s;**

**endAddress = e;**

**this.PID = PID;**

**size = e - s + 1;**

**free = "free".equals(PID.toLowerCase()) ? true : false;**

**}**

**}**

**static void allocateBlock(String PID, int size, int index) {**

**Block removed = memory.remove(index);**

**LinkedList<Block> \_new = new LinkedList<>();**

**\_new.add(new Block(removed.startAddress, removed.startAddress + size - 1, PID));**

**if (removed.size != size)**

**\_new.add(new Block(removed.startAddress + size, removed.endAddress, "FREE"));**

**memory.addAll(index, \_new);**

**}**

**static void allocateProcessWorstFit(String PID, int processSize) {**

**int i = 0;**

**int maxIndex = -1;**

**int maxSize = Integer.MIN\_VALUE;**

**for (Block current : memory) {**

**if (current.free && current.size >= processSize && current.size > maxSize) {**

**maxSize = current.size;**

**maxIndex = i;**

**}**

**i++;**

**}**

**if(maxIndex == -1)**

**System.out.println(">>> No free space available of size " + processSize + " for " + PID + " <<<");**

**else**

**allocateBlock(PID, processSize, maxIndex);**

**}**

**static void mergeHoles() {**

**for (int i = 0; i < memory.size() - 1; i++) {**

**Block current = memory.get(i);**

**Block next = memory.get(i + 1);**

**if (current.free && next.free) {**

**memory.set(i, new Block(current.startAddress, next.endAddress, "FREE"));**

**memory.remove(next);**

**i--;**

**}**

**}**

**}**

**static void deallocateProcess(String PID) {**

**int i = 0;**

**for (Block current : memory) {**

**if (PID.equals(current.PID)) {**

**current.PID = "FREE";**

**current.free = true;**

**mergeHoles();**

**break;**

**}**

**i++;**

**}**

**}**

**static void printMemory() {**

**String leftAlignFormat = "| %-7s | %-5d | %-5d | %-5d |%n";**

**System.out.println("---------- Memory Table ----------");**

**System.out.format("+---------+-------+-------+-------+%n");**

**System.out.format("| PID     | Start | End   | Size  |%n");**

**System.out.format("+---------+-------+-------+-------+%n");**

**for (Block curr : memory) {**

**System.out.format(leftAlignFormat, curr.PID, curr.startAddress, curr.endAddress, curr.size);**

**}**

**System.out.format("+---------+-------+-------+-------+%n");**

**}**

**static void initializeMemory() {**

**memory = new LinkedList<>();**

**memory.add(new Block(0, 7, "P1"));**

**memory.add(new Block(8, 12, "FREE"));**

**memory.add(new Block(13, 24, "P2"));**

**memory.add(new Block(25, 64, "FREE"));**

**memory.add(new Block(65, 80, "P3"));**

**memory.add(new Block(81, 100, "P4"));**

**memory.add(new Block(101, 127, "FREE"));**

**}**

**public static void main(String[] args) {**

**Scanner s = new Scanner(System.in);**

**String PID;**

**int processSize;**

**initializeMemory();**

**printMemory();**

**while (true) {**

**System.out.println(**

**"Enter 1 to allocate a process\nEnter 2 to deallocate a process\nEnter 0 to exit\nYour choice : ");**

**int option = Integer.parseInt(s.nextLine());**

**switch (option) {**

**case 1:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**System.out.println("Enter process size");**

**processSize = Integer.parseInt(s.nextLine());**

**allocateProcessWorstFit(PID, processSize);**

**printMemory();**

**break;**

**case 2:**

**System.out.println("Enter process ID");**

**PID = s.nextLine();**

**deallocateProcess(PID);**

**printMemory();**

**break;**

**default:**

**return;**

**}**

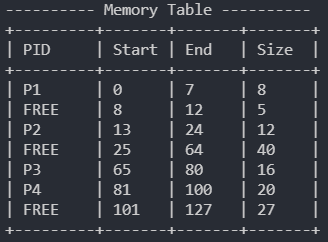
**}**

**}**

**}**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**OUTPUT**



|  |  |
| --- | --- |
|  |  |
|  |  |