SJF

package com.company;

class Process

{

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

public Process(int pid, int bt, int art)

{

this.pid = pid;

this.bt = bt;

this.art = art;

}

}

public class fcfs {

// Method to find the waiting time for all

// processes

static void findWaitingTime(Process proc[], int n,

int wt[]) {

int rt[] = new int[n];

// Copy the burst time into rt[]

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = Integer.MAX\_VALUE;

int shortest = 0, finish\_time;

boolean check = false;

// Process until all processes gets

// completed

while (complete != n) {

// Find process with minimum

// remaining time among the

// processes that arrives till the

// current time`

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

t++;

continue;

}

// Reduce remaining time by one

rt[shortest]--;

// Update minimum

minm = rt[shortest];

if (minm == 0)

minm = Integer.MAX\_VALUE;

// If a process gets completely

// executed

if (rt[shortest] == 0) {

// Increment complete

complete++;

check = false;

// Find finish time of current

// process

finish\_time = t + 1;

// Calculate waiting time

wt[shortest] = finish\_time -

proc[shortest].bt -

proc[shortest].art;

if (wt[shortest] < 0)

wt[shortest] = 0;

}

// Increment time

t++;

}

}

// Method to calculate turn around time

static void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[]) {

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = proc[i].bt + wt[i];

}

// Method to calculate average time

static void findavgTime(Process proc[], int n) {

int wt[] = new int[n], tat[] = new int[n];

int total\_wt = 0, total\_tat = 0;

// Function to find waiting time of all

// processes

findWaitingTime(proc, n, wt);

// Function to find turn around time for

// all processes

findTurnAroundTime(proc, n, wt, tat);

// Display processes along with all

// details

System.out.println("Processes " +

" Burst time " +

" Waiting time " +

" Turn around time");

// Calculate total waiting time and

// total turnaround time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.println(" " + proc[i].pid + "\t\t"

+ proc[i].bt + "\t\t " + wt[i]

+ "\t\t" + tat[i]);

}

System.out.println("Average waiting time = " +

(float) total\_wt / (float) n);

System.out.println("Average turn around time = " +

(float) total\_tat / (float) n);

}

// Driver Method

public static void main(String[] args) {

Process proc[] = {new Process(1, 6, 1),

new Process(2, 8, 1),

new Process(3, 7, 2),

new Process(4, 3, 3)};

findavgTime(proc, proc.length);

}

}

Fcfs

package com.company;

import java.text.ParseException;

public class fcfs2 {

// Function to find the waiting time for all

// processes

static void findWaitingTime(int processes[], int n,

int bt[], int wt[]) {

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

for (int i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1];

}

}

// Function to calculate turn around time

static void findTurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[]) {

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = bt[i] + wt[i];

}

}

//Function to calculate average time

static void findavgTime(int processes[], int n, int bt[]) {

int wt[] = new int[n], tat[] = new int[n];

int total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

System.out.printf("Processes Burst time Waiting"

+" time Turn around time\n");

// Calculate total waiting time and total turn

// around time

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

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.printf(" %d ", (i + 1));

System.out.printf(" %d ", bt[i]);

System.out.printf(" %d", wt[i]);

System.out.printf(" %d\n", tat[i]);

}

float s = (float)total\_wt /(float) n;

int t = total\_tat / n;

System.out.printf("Average waiting time = %f", s);

System.out.printf("\n");

System.out.printf("Average turn around time = %d ", t);

}

// Driver code

public static void main(String[] args) throws ParseException {

//process id's

int processes[] = {1, 2, 3};

int n = processes.length;

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

}

}

Best fit

package com.company;

public class bestfit

{

// Method to allocate memory to blocks as per Best fit

// algorithm

static void bestFit(int blockSize[], int m, int processSize[],

int n)

{

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++)

allocation[i] = -1;

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

// Find the best fit block for current process

int bestIdx = -1;

for (int j=0; j<m; j++)

{

if (blockSize[j] >= processSize[i])

{

if (bestIdx == -1)

bestIdx = j;

else if (blockSize[bestIdx] > blockSize[j])

bestIdx = j;

}

}

// If we could find a block for current process

if (bestIdx != -1)

{

// allocate block j to p[i] process

allocation[i] = bestIdx;

// Reduce available memory in this block.

blockSize[bestIdx] -= processSize[i];

}

}

System.out.println("\nProcess No.\tProcess Size\tBlock no.");

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

{

System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");

if (allocation[i] != -1)

System.out.print(allocation[i] + 1);

else

System.out.print("Not Allocated");

System.out.println();

}

}

// Driver Method

public static void main(String[] args)

{

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

bestFit(blockSize, m, processSize, n);

}

}

Worstfit:

package com.company;

public class worstfit

{

// Method to allocate memory to blocks as per worst fit

// algorithm

static void worstFit(int blockSize[], int m, int processSize[],

int n)

{

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++)

allocation[i] = -1;

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

// Find the best fit block for current process

int wstIdx = -1;

for (int j=0; j<m; j++)

{

if (blockSize[j] >= processSize[i])

{

if (wstIdx == -1)

wstIdx = j;

else if (blockSize[wstIdx] < blockSize[j])

wstIdx = j;

}

}

// If we could find a block for current process

if (wstIdx != -1)

{

// allocate block j to p[i] process

allocation[i] = wstIdx;

// Reduce available memory in this block.

blockSize[wstIdx] -= processSize[i];

}

}

System.out.println("\nProcess No.\tProcess Size\tBlock no.");

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

{

System.out.print(" " + (i+1) + "\t\t" + processSize[i] + "\t\t");

if (allocation[i] != -1)

System.out.print(allocation[i] + 1);

else

System.out.print("Not Allocated");

System.out.println();

}

}

// Driver Method

public static void main(String[] args)

{

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

worstFit(blockSize, m, processSize, n);

}

}

Nextfit

package com.company;

import java.util.Arrays;

public class nextfit {

// Function to allocate memory to blocks as per Next fit

// algorithm

static void NextFit(int blockSize[], int m, int processSize[], int n) {

// Stores block id of the block allocated to a

// process

int allocation[] = new int[n], j = 0;

// Initially no block is assigned to any process

Arrays.fill(allocation, -1);

// pick each process and find suitable blocks

// according to its size ad assign to it

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

// Do not start from beginning

int count =0;

while (j < m) {

count++; //makes sure that for every process we traverse through entire array maximum once only.This avoids the problem of going into infinite loop if memory is not available

if (blockSize[j] >= processSize[i]) {

// allocate block j to p[i] process

allocation[i] = j;

// Reduce available memory in this block.

blockSize[j] -= processSize[i];

break;

}

// mod m will help in traversing the blocks from

// starting block after we reach the end.

j = (j + 1) % m;

}

}

System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");

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

System.out.print( i + 1 + "\t\t" + processSize[i]

+ "\t\t");

if (allocation[i] != -1) {

System.out.print(allocation[i] + 1);

} else {

System.out.print("Not Allocated");

}

System.out.println("");

}

}

// Driver program

static public void main(String[] args) {

int blockSize[] = {5, 10, 20};

int processSize[] = {10, 20, 5};

int m = blockSize.length;

int n = processSize.length;

NextFit(blockSize, m, processSize, n);

}

}

First fit

package com.company;

class firstfit

{

// Method to allocate memory to

// blocks as per First fit algorithm

static void firstFit(int blockSize[], int m,

int processSize[], int n)

{

// Stores block id of the

// block allocated to a process

int allocation[] = new int[n];

// Initially no block is assigned to any process

for (int i = 0; i < allocation.length; i++)

allocation[i] = -1;

// pick each process and find suitable blocks

// according to its size ad assign to it

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

{

for (int j = 0; j < m; j++)

{

if (blockSize[j] >= processSize[i])

{

// allocate block j to p[i] process

allocation[i] = j;

// Reduce available memory in this block.

blockSize[j] -= processSize[i];

break;

}

}

}

System.out.println("\nProcess No.\tProcess\_Size");

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

{

System.out.print(" " + (i+1) + "\t\t" +

processSize[i] + "\t\t");

if (allocation[i] != -1)

System.out.print(allocation[i] + 1);

else

System.out.print("Not Allocated");

System.out.println();

}

}

// Driver Code

public static void main(String[] args)

{

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = blockSize.length;

int n = processSize.length;

firstFit(blockSize, m, processSize, n);

}

}