**fcfs CPU scheduling**

#include <stdio.h>

int main() {

int n, bt[10], wt = 0, tat = 0, i;

float avg\_wt = 0, avg\_tat = 0;

printf("Processes: ");

scanf("%d", &n);

printf("Burst times:\n");

for (i = 0; i < n; i++) scanf("%d", &bt[i]);

printf("\nP\tBT\tWT\tTAT\n");

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

int tat = wt + bt[i];

printf("%d\t%d\t%d\t%d\n", i + 1, bt[i], wt, tat);

avg\_wt += wt;

avg\_tat += tat;

wt = tat;

}

printf("\nAvg WT=%.2f\nAvg TAT=%.2f\n", avg\_wt / n, avg\_tat / n);

return 0;

}

**fcfs disk**

#include <stdio.h>

#include <stdlib.h> // For abs()

int main() {

int n, i, head, seek = 0;

int queue[100];

printf("Enter number of disk requests: ");

scanf("%d", &n);

printf("Enter the disk request sequence: ");

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

scanf("%d", &queue[i]);

printf("Enter initial head position: ");

scanf("%d", &head);

printf("\nSequence of head movement:\n");

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

printf("Head moves from %d to %d with seek %d\n", head, queue[i], abs(queue[i] - head));

seek += abs(queue[i] - head);

head = queue[i];

}

printf("\nTotal Seek Time = %d\n", seek);

printf("Average Seek Time = %.2f\n", (float)seek / n);

return 0;

}

**SCAN**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, i, head, req[20], seek = 0;

scanf("%d", &n);

for (i = 0; i < n; i++) scanf("%d", &req[i]);

scanf("%d", &head);

req[n++] = 0; // add start and end of disk

req[n++] = 199;

req[n++] = head;

// sort requests

for (i = 0; i < n - 1; i++)

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

if (req[i] > req[j]) {

int t = req[i];

req[i] = req[j];

req[j] = t;

}

// find head position

for (i = 0; i < n; i++) if (req[i] == head) break;

printf("Sequence: ");

// move towards higher tracks

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

printf("%d ", req[j]);

seek += abs(head - req[j]);

head = req[j];

}

// reverse and move towards lower tracks

for (int j = i - 1; j >= 0; j--) {

printf("%d ", req[j]);

seek += abs(head - req[j]);

head = req[j];

}

printf("\nTotal Seek Time: %d\n", seek);

}

**Worst**

#include <stdio.h>

void worstFit(int blocks[], int m, int process[], int n) {

int alloc[n];

for (int i = 0; i < n; i++) alloc[i] = -1;

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

int worst = -1;

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

if (blocks[j] >= process[i] && (worst == -1 || blocks[j] > blocks[worst]))

worst = j;

if (worst != -1) {

alloc[i] = worst;

blocks[worst] -= process[i];

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

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

printf("%d\t\t%d\t\t%s\n", i + 1, process[i], (alloc[i] != -1) ? ("Block " + alloc[i]) : "Not Allocated");

}

int main() {

int blocks[] = {100, 500, 200, 300, 600}, process[] = {212, 417, 112, 426};

int m = sizeof(blocks) / sizeof(blocks[0]), n = sizeof(process) / sizeof(process[0]);

worstFit(blocks, m, process, n);

}

**Best**

#include <stdio.h>

int main() {

int blockSize[5] = {100, 500, 200, 300, 600};

int processSize[4] = {212, 417, 112, 426};

int alloc[4], i, j;

for (i = 0; i < 4; i++) alloc[i] = -1;

for (i = 0; i < 4; i++) {

int bestIdx = -1;

for (j = 0; j < 5; j++) {

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

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx])

bestIdx = j;

}

}

if (bestIdx != -1) {

alloc[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

for (i = 0; i < 4; i++) {

if (alloc[i] != -1)

printf("%d\t\t%d\t\t%d\n", i + 1, processSize[i], alloc[i] + 1);

else

printf("%d\t\t%d\t\tNot Allocated\n", i + 1, processSize[i]);

}

return 0;

}

**First**

#include <stdio.h>

int main() {

int blockSize[5] = {100, 500, 200, 300, 600};

int processSize[4] = {212, 417, 112, 426};

int allocation[4] = {-1, -1, -1, -1};

int i, j;

for (i = 0; i < 4; i++) {

for (j = 0; j < 5; j++) {

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

allocation[i] = j;

blockSize[j] -= processSize[i];

break;

}

}

}

printf("Process No.\tProcess Size\tBlock No.\n");

for (i = 0; i < 4; i++) {

if (allocation[i] != -1)

printf("%d\t\t%d\t\t%d\n", i + 1, processSize[i], allocation[i] + 1);

else

printf("%d\t\t%d\t\tNot Allocated\n", i + 1, processSize[i]);

}

return 0;

}

**Round Robin**

#include <stdio.h>

#define MAX\_PROCESS 10

void roundRobin(int arrivalTime[], int burstTime[], int n, int quantum) {

int remainingTime[MAX\_PROCESS];

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

remainingTime[i] = burstTime[i];

}

int currentTime = 0;

int flag = 0;

while (1) {

flag = 0;

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

if (remainingTime[i] > 0) {

flag = 1;

if (remainingTime[i] > quantum) {

currentTime += quantum;

remainingTime[i] -= quantum;

} else {

currentTime += remainingTime[i];

remainingTime[i] = 0;

printf("Process %d completed at %d\n", i, currentTime);

}

}

}

if (flag == 0) {

break;

}

}

}

int main() {

int arrivalTime[] = {0, 1, 2, 3};

int burstTime[] = {5, 4, 2, 1};

int n = sizeof(arrivalTime) / sizeof(arrivalTime[0]);

int quantum = 2;

roundRobin(arrivalTime, burstTime, n, quantum);

return 0;

}

**Banker’s Alg**

#include <stdio.h>

int main() {

int n, m;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter number of resources: ");

scanf("%d", &m);

int alloc[n][m], max[n][m], avail[m];

int need[n][m];

printf("Enter allocation matrix:\n");

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

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

scanf("%d", &alloc[i][j]);

printf("Enter max matrix:\n");

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

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

scanf("%d", &max[i][j]);

printf("Enter available resources:\n");

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

scanf("%d", &avail[i]);

// Calculate need matrix

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

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

need[i][j] = max[i][j] - alloc[i][j];

int finish[n], safeSeq[n], count = 0;

for(int i=0; i<n; i++) finish[i] = 0;

while (count < n) {

int found = 0;

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

if (!finish[i]) {

int j;

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

if (need[i][j] > avail[j])

break;

if (j == m) {

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

avail[k] += alloc[i][k];

safeSeq[count++] = i;

finish[i] = 1;

found = 1;

}

}

}

if (!found) {

printf("System is not in safe state.\n");

return 0;

}

}

printf("System is in a safe state.\nSafe sequence: ");

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

printf("P%d ", safeSeq[i]);

printf("\n");

return 0;

}