**Fcfs algorithm ‘**

#include <stdio.h>

void findWaitingTime(int n, int bt[], int at[], int wt[]) {

wt[0] = 0;

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

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

if (wt[i] < 0) {

wt[i] = 0;

}

}

}

void findTurnAroundTime(int n, int bt[], int at[], int wt[], int tat[]) {

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

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

}

}

void findAverageTimes(int n, int bt[], int at[]) {

int wt[n], tat[n];

findWaitingTime(n, bt, at, wt);

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

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

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

total\_wt += wt[i];

total\_tat += tat[i];

}

printf("\nProcess\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

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

}

printf("\nAverage Waiting Time: %.2f", (float)total\_wt / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int bt[n], at[n];

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

printf("\nEnter Burst Time and Arrival Time for Process %d: ", i + 1);

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

}

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

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

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

int temp = bt[i];

bt[i] = bt[j];

bt[j] = temp;

temp = at[i];

at[i] = at[j];

at[j] = temp;

}

}

}

findAverageTimes(n, bt, at);

return 0;

}

**SJF algorithm (preemitive)**

#include <stdio.h>

#define MAX\_PROCESSES 10

void calculateWaitingTime(int n, int bt[], int at[], int wt[]) {

int remaining\_bt[n];

int complete = 0, time = 0, min\_time, shortest, finish\_time;

int check[n];

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

remaining\_bt[i] = bt[i];

check[i] = 0;

}

while (complete < n) {

min\_time = 9999;

shortest = -1;

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

if ((at[i] <= time) && (check[i] == 0) && (remaining\_bt[i] < min\_time) && (remaining\_bt[i] > 0)) {

min\_time = remaining\_bt[i];

shortest = i;

}

}

if (shortest == -1) {

time++;

continue;

}

remaining\_bt[shortest]--;

if (remaining\_bt[shortest] == 0) {

complete++;

finish\_time = time + 1;

wt[shortest] = finish\_time - at[shortest] - bt[shortest];

if (wt[shortest] < 0) {

wt[shortest] = 0;

}

}

time++;

}

}

void calculateAverageWaitingTime(int n, int bt[], int at[]) {

int wt[n];

int total\_wt = 0;

calculateWaitingTime(n, bt, at, wt);

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

total\_wt += wt[i];

}

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

printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int bt[n], at[n];

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

printf("\nEnter Burst Time and Arrival Time for Process %d: ", i + 1);

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

}

calculateAverageWaitingTime(n, bt, at);

return 0;

}

**SJF (Non preemitive )**

#include <stdio.h>

#define MAX\_PROCESSES 10

void calculateWaitingTime(int n, int bt[], int at[], int wt[]) {

int remaining\_bt[n], completed[n];

int time = 0, complete = 0, min\_time, shortest = -1;

int finish\_time;

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

remaining\_bt[i] = bt[i];

completed[i] = 0;

}

while (complete < n) {

min\_time = 9999;

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

if (at[i] <= time && completed[i] == 0 && remaining\_bt[i] < min\_time) {

min\_time = remaining\_bt[i];

shortest = i;

}

}

if (shortest == -1) {

time++;

continue;

} time += remaining\_bt[shortest];

finish\_time = time;

wt[shortest] = finish\_time - at[shortest] - bt[shortest];

if (wt[shortest] < 0) {

wt[shortest] = 0;

} completed[shortest] = 1;

complete++;

}

}

void calculateAverageWaitingTime(int n, int bt[], int at[]) {

int wt[n], total\_wt = 0;

calculateWaitingTime(n, bt, at, wt);

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

total\_wt += wt[i];

}

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

printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int bt[n], at[n];

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

printf("\nEnter Burst Time and Arrival Time for Process %d: ", i + 1);

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

}

calculateAverageWaitingTime(n, bt, at);

return 0;}

**Round Robin Scheduling**

#include <stdio.h>

#define MAX\_PROCESSES 10

void calculateWaitingTime(int n, int bt[], int at[], int wt[], int tq) {

int remaining\_bt[n], completed[n];

int time = 0, complete = 0;

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

remaining\_bt[i] = bt[i];

completed[i] = 0;

}

while (complete < n) {

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

if (remaining\_bt[i] > 0 && at[i] <= time) {

if (remaining\_bt[i] > tq) {

time += tq;

remaining\_bt[i] -= tq;

} else {

time += remaining\_bt[i];

wt[i] = time - at[i] - bt[i];

if (wt[i] < 0) {

wt[i] = 0;

}

remaining\_bt[i] = 0;

completed[i] = 1;

complete++;

}}}}}

void calculateAverageWaitingTime(int n, int bt[], int at[], int tq) {

int wt[n], total\_wt = 0;

calculateWaitingTime(n, bt, at, wt, tq);

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

total\_wt += wt[i];

}

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

printf("\nAverage Waiting Time: %.2f\n", avg\_wt);

}

int main() {

int n, tq;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter Time Quantum: ");

scanf("%d", &tq);

int bt[n], at[n];

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

printf("\nEnter Burst Time and Arrival Time for Process %d: ", i + 1);

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

}

calculateAverageWaitingTime(n, bt, at, tq);

return 0;

}

**Bankers algorithm**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

void calculateNeedMatrix(int n, int m, int max[][MAX\_RESOURCES], int alloc[][MAX\_RESOURCES], int need[][MAX\_RESOURCES]) {

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

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

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

}}}

void displayMatrix(int n, int m, int matrix[][MAX\_RESOURCES]) {

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

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

printf("%d ", matrix[i][j]);

}

printf("\n");

}}

int main() {

int n, m;

printf("Enter the number of processes: ");

scanf("%d", &n);

printf("Enter the number of resources: ");

scanf("%d", &m);

int available[MAX\_RESOURCES], max[MAX\_PROCESSES][MAX\_RESOURCES], alloc[MAX\_PROCESSES][MAX\_RESOURCES], need[MAX\_PROCESSES][MAX\_RESOURCES];

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

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

printf("Resource %d: ", i + 1);

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

}

printf("\nEnter Maximum Resource Requirement for each process:\n");

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

printf("\nProcess %d:\n", i + 1);

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

printf("Resource %d: ", j + 1);

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

}

}

printf("\nEnter Allocation Matrix:\n");

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

printf("\nProcess %d:\n", i + 1);

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

printf("Resource %d: ", j + 1);

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

}

}

calculateNeedMatrix(n, m, max, alloc, need);

printf("\nNeed Matrix (Max - Allocation):\n");

displayMatrix(n, m, need);

return 0;}

**Matrix**

#include <stdio.h>

int main() {

int n, m, i, j, k;

n = 5;

m = 3;

int alloc[5][3] = { { 0, 1, 0 },

{ 2, 0, 0 },

{ 3, 0, 2 },

{ 2, 1, 1 },

{ 0, 0, 2 } };

int max[5][3] = { { 7, 5, 3 },

{ 3, 2, 2 },

{ 9, 0, 2 },

{ 2, 2, 2 },

{ 4, 3, 3 } };

int avail[3] = { 3, 3, 2 };

int f[n], ans[n], ind = 0;

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

f[k] = 0;

}

int need[n][m];

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

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

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

}

int y = 0;

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

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

if (f[i] == 0) {

int flag = 0;

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

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

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

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

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

f[i] = 1;

}}}}

int flag = 1;

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

if(f[i] == 0) {

flag = 0;

printf("The following system is not safe");

Break;}}

if(flag == 1) {

printf("Following is the SAFE Sequence\n");

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

printf(" P%d ->", ans[i]);

printf(" P%d", ans[n - 1]);

}

return (0);}