Exp 6

#include <stdio.h>

#include <stdlib.h>

#define NUM\_PROCESSES 3

#define NUM\_RESOURCES 3

int main() {

// Initialize the resource matrices

int max[NUM\_PROCESSES][NUM\_RESOURCES] = {{3, 6, 8}, {4, 3, 3}, {3, 4, 4}};

int allocation[NUM\_PROCESSES][NUM\_RESOURCES] = {{3, 3, 3}, {2, 0, 3}, {1, 2, 4}};

int available[NUM\_RESOURCES] = {1, 2, 0};

int finish[NUM\_PROCESSES] = {0};

// Compute the need matrix

int need[NUM\_PROCESSES][NUM\_RESOURCES];

for (int i = 0; i < NUM\_PROCESSES; i++) {

for (int j = 0; j < NUM\_RESOURCES; j++) {

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

}

}

// Initialize the work vector

int work[NUM\_RESOURCES];

for (int i = 0; i < NUM\_RESOURCES; i++) {

work[i] = available[i];

}

// Iterate until all processes have finished or a deadlock is detected

int count = 0;

while (count < NUM\_PROCESSES) {

int found = 0;

for (int i = 0; i < NUM\_PROCESSES; i++) {

if (!finish[i]) {

int j;

for (j = 0; j < NUM\_RESOURCES; j++) {

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

break;

}

}

if (j == NUM\_RESOURCES) {

for (int k = 0; k < NUM\_RESOURCES; k++) {

work[k] += allocation[i][k];

}

finish[i] = 1;

found = 1;

count++;

}

}

}

if (!found) {

printf("Deadlock detected!\n");

// Identify the deadlocked processes

printf("Deadlocked processes: ");

for (int i = 0; i < NUM\_PROCESSES; i++) {

if (!finish[i]) {

printf("%d ", i);

}

}

printf("\n");

exit(0);

}

}

printf("No deadlock detected.\n");

return 0;

}

Exp 7

#include <stdio.h>

#define MAX\_FRAMES 3

int main() {

int page\_frames[MAX\_FRAMES] = {0};

int page\_faults = 0;

int page\_reference[] = {1, 2, 3, 2, 1, 5, 2, 1, 6, 2, 5, 6, 3, 1, 3, 6, 1, 2, 4, 3};

int num\_pages = sizeof(page\_reference) / sizeof(page\_reference[0]);

for (int i = 0; i < MAX\_FRAMES; i++) {

page\_frames[i] = -1;

}

for (int i = 0; i < num\_pages; i++) {

int page = page\_reference[i];

int found = 0;

for (int j = 0; j < MAX\_FRAMES; j++) {

if (page\_frames[j] == page) {

found = 1;

break;

}

}

if (!found) {

int lru\_frame = 0;

int lru\_time = page\_frames[0];

for (int j = 1; j < MAX\_FRAMES; j++) {

if (page\_frames[j] == -1) {

lru\_frame = j;

break;

}

if (page\_frames[j] < lru\_time) {

lru\_frame = j;

lru\_time = page\_frames[j];

}

}

page\_frames[lru\_frame] = page;

page\_faults++;

}

}

printf("Number of page faults: %d\n", page\_faults);

return 0;

}

Exp8

#include <stdio.h>

#include <stdlib.h>

int main() {

int num\_tracks = 5;

int track\_positions[] = {55, 58, 60, 70, 18};

int initial\_position;

printf("Enter the initial position of the disk head: ");

scanf("%d", &initial\_position);

int total\_head\_movement = 0;

int current\_position = initial\_position;

printf("Disk head movement:\n");

for (int i = 0; i < num\_tracks; i++) {

int track = track\_positions[i];

int head\_movement = abs(track - current\_position);

total\_head\_movement += head\_movement;

printf("%d -> %d\t Head movement: %d\n", current\_position, track, head\_movement);

current\_position = track;

}

float average\_head\_movement = (float) total\_head\_movement / num\_tracks;

printf("Average head movement: %.2f\n", average\_head\_movement);

return 0;

}

Exp 9

#include <stdio.h>

int main()

{

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

int burst\_time[3] = {2, 4, 8};

int arrival\_time[3] = {0, 0, 0};

int completion\_time[3];

int waiting\_time[3];

int turnaround\_time[3];

float avg\_waiting\_time = 0;

float avg\_turnaround\_time = 0;

int i, j;

completion\_time[0] = burst\_time[0];

for (i = 1; i < 3; i++) {

completion\_time[i] = completion\_time[i-1] + burst\_time[i];

}

for (i = 0; i < 3; i++) {

waiting\_time[i] = completion\_time[i] - burst\_time[i] - arrival\_time[i];

turnaround\_time[i] = completion\_time[i] - arrival\_time[i];

avg\_waiting\_time += waiting\_time[i];

avg\_turnaround\_time += turnaround\_time[i];

}

printf("Process\tBurst time\tArrival time\tWaiting time\tTurnaround time\n");

for (i = 0; i < 3; i++) {

printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i], burst\_time[i], arrival\_time[i], waiting\_time[i], turnaround\_time[i]);

}

avg\_waiting\_time /= 3;

avg\_turnaround\_time /= 3;

printf("Average waiting time: %.2f\n", avg\_waiting\_time);

printf("Average turnaround time: %.2f\n", avg\_turnaround\_time);

return 0;

}

Exp 10

#include <stdio.h>

#define NUM\_PROCESSES 5

#define NUM\_RESOURCES 3

void calculateNeedMatrix(int allocation[][NUM\_RESOURCES], int max[][NUM\_RESOURCES], int need[][NUM\_RESOURCES], int available[]) {

for (int i = 0; i < NUM\_PROCESSES; i++) {

for (int j = 0; j < NUM\_RESOURCES; j++) {

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

}

}

}

int main() {

int allocation[NUM\_PROCESSES][NUM\_RESOURCES] = {

{1, 1, 2},

{2, 1, 2},

{3, 0, 1},

{0, 2, 0},

{1, 1, 2}

};

int max[NUM\_PROCESSES][NUM\_RESOURCES] = {

{5, 4, 4},

{4, 3, 3},

{9, 1, 3},

{8, 6, 4},

{2, 2, 3}

};

int available[NUM\_RESOURCES] = {3, 2, 1};

int need[NUM\_PROCESSES][NUM\_RESOURCES];

calculateNeedMatrix(allocation, max, need, available);

printf("Need Matrix:\n");

for (int i = 0; i < NUM\_PROCESSES; i++) {

for (int j = 0; j < NUM\_RESOURCES; j++) {

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

}

printf("\n");

}

return 0;

}