1. Deadlock detection:

using namespace std;

int arrmax[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n, r;

void input()

{

int i, j;

cout << "Enter the no of Processes\t";

cin >> n;

cout << "Enter the no of resource instances\t";

cin >> r;

cout << "Enter the Max Matrix\n";

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

{

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

{

cin >> arrmax[i][j];

}

}

cout << "Enter the Allocation Matrix\n";

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

{

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

{

cin >> alloc[i][j];

}

}

cout << "Enter the available Resources\n";

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

{

cin >> avail[j];

}

}

void show()

{

int i, j;

cout << "Process\t Allocation\t Max\t Available\t";

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

{

cout << "\nP" << i + 1 << "\t ";

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

{

cout << alloc[i][j] << " ";

}

cout << "\t\t";

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

{

cout << arrmax[i][j] << " ";

}

cout << "\t ";

if (i == 0)

{

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

cout << avail[j] << " ";

}

}

}

void cal()

{

int finish[100], temp, need[100][100], flag = 1, k, c1 = 0;

int dead[100];

int safe[100];

int i, j;

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

{

finish[i] = 0;

}

//find need matrix

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

{

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

{

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

}

}

while (flag)

{

flag = 0;

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

{

int c = 0;

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

{

if ((finish[i] == 0) && (need[i][j] <= avail[j]))

{

c++;

if (c == r)

{

for (k = 0; k < r; k++)

{

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

finish[i] = 1;

flag = 1;

}

//cout<<"\nP%d",i;

if (finish[i] == 1)

{

i = n;

}

}

}

}

}

}

j = 0;

flag = 0;

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

{

if (finish[i] == 0)

{

dead[j] = i;

j++;

flag = 1;

}

}

if (flag == 1)

{

cout << "\n\nSystem is in Deadlock and the Deadlock process are\n";

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

{

cout << "P" << dead[i] << "\t";

}

}

else

{

cout << "\nNo Deadlock Occur";

}

}

int main()

{

int i, j;

cout << "\*\*\*\*\*\*\*\*\*\* Deadlock Detection Algorithm \*\*\*\*\*\*\*\*\*\*\*\*\n";

input();

show();

cal();

return 0;

}

Output:

Enter the no of Processes 4

Enter the no of resource instances 3

Enter the Max Matrix

1

3

4

5

7

9

11

14

6

8

15

5

Enter the Allocation Matrix

1

3

4

6

7

9

0

11

12

14

16

51

Enter the available Resources

1

2

0

Process Allocation Max Available

P1 1 3 4 1 3 4 1 2 0

P2 6 7 9 5 7 9

P3 0 11 12 11 14 6

P4 14 16 51 8 15 5

No Deadlock Occur.

2. Thread

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

int g = 0;

void \*myThreadFun(void \*vargp)

{

int \*myid = (int \*)vargp;

static int s = 0;

++s; ++g;

printf("Thread ID: %d, Static: %d, Global: %d\n", \*myid, ++s, ++g);

}

int main()

{

int i;

pthread\_t tid;

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

pthread\_create(&tid, NULL, myThreadFun, (void \*)&tid);

pthread\_exit(NULL);

return 0;

}

Output:

Thread ID: 3, Static: 2, Global: 2

Thread ID: 3, Static: 4, Global: 4

Thread ID: 3, Static: 6, Global: 6

3.FIFO

#include <stdio.h>

#define NUM\_FRAMES 3

#define NUM\_PAGES 10

int frames[NUM\_FRAMES];

int pages[NUM\_PAGES];

int page\_faults = 0;

void initializeFrames() {

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

frames[i] = -1; // Initialize frames to -1 (indicating an empty frame)

}

}

int findLRUIndex() {

int min = frames[0], index = 0;

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

if (frames[i] < min) {

min = frames[i];

index = i;

}

}

return index;

}

int isPageInFrames(int page) {

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

if (frames[i] == page) {

return 1;

}

}

return 0;

}

void printFrames() {

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

if (frames[i] == -1) {

printf("- ");

} else {

printf("%d ", frames[i]);

}

}

printf("\n");

}

void simulatePaging() {

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

printf("Accessing Page %d: ", pages[i]);

if (!isPageInFrames(pages[i])) {

page\_faults++;

int emptyIndex = -1;

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

if (frames[j] == -1) {

emptyIndex = j;

break;

}

}

if (emptyIndex != -1) {

frames[emptyIndex] = pages[i];

} else {

int replaceIndex = findLRUIndex();

frames[replaceIndex] = pages[i];

}

}

printFrames();

}

}

int main()

{

printf("Enter the sequence of page references (0-9):\n");

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

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

}

initializeFrames();

simulatePaging();

printf("Total Page Faults: %d\n", page\_faults);

return 0;

}

Output:

Enter the sequence of page references (0-9):

3

4

5

7

8

2

4

1

6

9

Accessing Page 3: 3 - -

Accessing Page 4: 3 4 -

Accessing Page 5: 3 4 5

Accessing Page 7: 7 4 5

Accessing Page 8: 7 8 5

Accessing Page 2: 7 8 2

Accessing Page 4: 7 8 4

Accessing Page 1: 7 8 1

Accessing Page 6: 7 8 6

Accessing Page 9: 7 8 9

Total Page Faults: 10

4.First fit

#include <stdio.h>

#define MEMORY\_SIZE 100

#define MAX\_BLOCKS 10

int memory[MEMORY\_SIZE];

int blocks[MAX\_BLOCKS];

int num\_blocks;

void initializeMemory() {

for (int i = 0; i < MEMORY\_SIZE; i++) {

memory[i] = -1; // Initialize memory to -1 (indicating free space)

}

}

void firstFit(int process\_id, int size) {

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] == -1) { // Check for free space

int j, count = 0;

for (j = i; j < MEMORY\_SIZE && memory[j] == -1; j++) {

count++;

if (count == size) {

break;

}

}

if (count == size) {

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

memory[k] = process\_id;

}

printf("Process %d allocated at memory location %d\n", process\_id, i);

return;

}

}

}

printf("Insufficient memory to allocate Process %d\n", process\_id);

}

int main() {

initializeMemory();

printf("Enter number of memory blocks: ");

scanf("%d", &num\_blocks);

printf("Enter size of each block:\n");

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

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

}

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

int process\_id, size;

printf("Enter process ID and size for Block %d: ", i + 1);

scanf("%d %d", &process\_id, &size);

firstFit(process\_id, size);

}

return 0;

}

Output:

Enter number of memory blocks: 2

Enter size of each block:

1

4

Enter process ID and size for Block 1: 7

3

Process 7 allocated at memory location 0

Enter process ID and size for Block 2: 2

5

Process 2 allocated at memory location 2

6. Worst fit

#include <stdio.h>

#define MEMORY\_SIZE 100

#define MAX\_BLOCKS 10

int memory[MEMORY\_SIZE];

int blocks[MAX\_BLOCKS];

int num\_blocks;

void initializeMemory() {

for (int i = 0; i < MEMORY\_SIZE; i++) {

memory[i] = -1; // Initialize memory to -1 (indicating free space)

}

}

void worstFit(int process\_id, int size) {

int max\_size = 0, max\_index = -1;

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] == -1) { // Check for free space

int j, count = 0;

for (j = i; j < MEMORY\_SIZE && memory[j] == -1; j++) {

count++;

}

if (count >= size && count > max\_size) {

max\_size = count;

max\_index = i;

}

}

}

if (max\_index != -1) {

for (int k = max\_index; k < max\_index + size; k++) {

memory[k] = process\_id;

}

printf("Process %d allocated at memory location %d\n", process\_id, max\_index);

} else {

printf("Insufficient memory to allocate Process %d\n", process\_id);

}

}

int main() {

initializeMemory();

printf("Enter number of memory blocks: ");

scanf("%d", &num\_blocks);

printf("Enter size of each block:\n");

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

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

}

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

int process\_id, size;

printf("Enter process ID and size for Block %d: ", i + 1);

scanf("%d %d", &process\_id, &size);

worstFit(process\_id, size);

}

return 0;

}

Output:

Enter number of memory blocks: 4

Enter size of each block:

2

4

5

7

Enter process ID and size for Block 1: 2

3

Process 2 allocated at memory location 0

Enter process ID and size for Block 2: 4

6

Process 4 allocated at memory location 3

Enter process ID and size for Block 3: 5

8

Process 5 allocated at memory location 9

Enter process ID and size for Block 4: 7

9

Process 7 allocated at memory location 17

Best fit:

#include <stdio.h>

#define MEMORY\_SIZE 100

#define MAX\_BLOCKS 10

int memory[MEMORY\_SIZE];

int blocks[MAX\_BLOCKS];

int num\_blocks;

void initializeMemory() {

for (int i = 0; i < MEMORY\_SIZE; i++) {

memory[i] = -1; // Initialize memory to -1 (indicating free space)

}

}

void bestFit(int process\_id, int size) {

int min\_size = MEMORY\_SIZE, min\_index = -1;

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] == -1) { // Check for free space

int j, count = 0;

for (j = i; j < MEMORY\_SIZE && memory[j] == -1; j++) {

count++;

}

if (count >= size && count < min\_size) {

min\_size = count;

min\_index = i;

}

}

}

if (min\_index != -1) {

for (int k = min\_index; k < min\_index + size; k++) {

memory[k] = process\_id;

}

printf("Process %d allocated at memory location %d\n", process\_id, min\_index);

} else {

printf("Insufficient memory to allocate Process %d\n", process\_id);

}

}

int main() {

initializeMemory();

printf("Enter number of memory blocks: ");

scanf("%d", &num\_blocks);

printf("Enter size of each block:\n");

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

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

}

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

int process\_id, size;

printf("Enter process ID and size for Block %d: ", i + 1);

scanf("%d %d", &process\_id, &size);

bestFit(process\_id, size);

}

return 0;

}

Output:

Enter number of memory blocks: 2

Enter size of each block:

4

7

Enter process ID and size for Block 1: 4

5

Process 4 allocated at memory location 95

Enter process ID and size for Block 2: 7

8

Process 7 allocated at memory location 87

LRU

#include <stdio.h>

#include <stdbool.h>

#define NUM\_FRAMES 3

#define NUM\_PAGES 10

int frames[NUM\_FRAMES];

int pages[NUM\_PAGES];

int page\_faults = 0;

int access\_order[NUM\_PAGES];

int next\_replace = 0;

void initializeFrames() {

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

frames[i] = -1; // Initialize frames to -1 (indicating an empty frame)

}

}

bool isPageInFrames(int page) {

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

if (frames[i] == page) {

return true;

}

}

return false;

}

void updateAccessOrder(int page) {

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

if (access\_order[i] == page) {

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

access\_order[j] = access\_order[j - 1];

}

access\_order[0] = page;

break;

}

}

}

void replacePage(int page) {

frames[next\_replace] = page;

updateAccessOrder(page);

next\_replace = (next\_replace + 1) % NUM\_FRAMES;

}

void simulateLRU() {

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

printf("Accessing Page %d: ", pages[i]);

if (!isPageInFrames(pages[i])) {

page\_faults++;

replacePage(pages[i]);

} else {

updateAccessOrder(pages[i]);

}

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

if (frames[j] == -1) {

printf("- ");

} else {

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

}

}

printf("\n");

}

}

int main() {

initializeFrames();

printf("Enter the sequence of page references (0-9):\n");

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

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

access\_order[i] = -1;

}

simulateLRU();

printf("Total Page Faults: %d\n", page\_faults);

return 0;

}

Output:

Enter the sequence of page references (0-9):

2

4

6

8

9

0

1

3

5

7

Accessing Page 2: 2 - -

Accessing Page 4: 2 4 -

Accessing Page 6: 2 4 6

Accessing Page 8: 8 4 6

Accessing Page 9: 8 9 6

Accessing Page 0: 8 9 0

Accessing Page 1: 1 9 0

Accessing Page 3: 1 3 0

Accessing Page 5: 1 3 5

Accessing Page 7: 7 3 5

Total Page Faults: 10

Optimal page replacement

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define NUM\_FRAMES 3

#define NUM\_PAGES 10

int frames[NUM\_FRAMES];

int pages[NUM\_PAGES];

int page\_faults = 0;

int next\_replace = 0;

void initializeFrames() {

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

frames[i] = -1; // Initialize frames to -1 (indicating an empty frame)

}

}

bool isPageInFrames(int page) {

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

if (frames[i] == page) {

return true;

}

}

return false;

}

int findNextReference(int start) {

for (int i = start; i < NUM\_PAGES; i++) {

if (!isPageInFrames(pages[i])) {

return i;

}

}

return INT\_MAX;

}

void replacePage(int page) {

frames[next\_replace] = page;

next\_replace = (next\_replace + 1) % NUM\_FRAMES;

}

void simulateOptimal() {

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

printf("Accessing Page %d: ", pages[i]);

if (!isPageInFrames(pages[i])) {

page\_faults++;

int max\_future\_reference = -1;

int page\_to\_replace = -1;

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

int next\_reference = findNextReference(i + 1);

if (next\_reference > max\_future\_reference) {

max\_future\_reference = next\_reference;

page\_to\_replace = j;

}

}

replacePage(pages[i]);

}

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

if (frames[j] == -1) {

printf("- ");

} else {

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

}

}

printf("\n");

}

}

int main() {

initializeFrames();

printf("Enter the sequence of page references (0-9):\n");

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

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

}

simulateOptimal();

printf("Total Page Faults: %d\n", page\_faults);

return 0;

}

Output

Enter the sequence of page references (0-9):

2

3

5

6

8

9

0

1

10

5

Accessing Page 2: 2 - -

Accessing Page 3: 2 3 -

Accessing Page 5: 2 3 5

Accessing Page 6: 6 3 5

Accessing Page 8: 6 8 5

Accessing Page 9: 6 8 9

Accessing Page 0: 0 8 9

Accessing Page 1: 0 1 9

Accessing Page 10: 0 1 10

Accessing Page 5: 5 1 10

Total Page Faults: 10

Sequential file organization:

#include <stdio.h>

#include <string.h>

struct Student {

int roll\_number;

char name[50];

};

int main() {

struct Student student;

FILE \*file;

file = fopen("students.dat", "wb");

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

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

printf("Enter details for Student %d:\n", i + 1);

printf("Roll Number: ");

scanf("%d", &student.roll\_number);

printf("Name: ");

scanf("%s", student.name);

fwrite(&student, sizeof(struct Student), 1, file);

}

fclose(file);

file = fopen("students.dat", "rb");

if (file == NULL) {

printf("Error opening file.\n");

return 1;

}

printf("\nStudent details:\n");

while (fread(&student, sizeof(struct Student), 1, file)) {

printf("Roll Number: %d, Name: %s\n", student.roll\_number, student.name);

}

fclose(file);

return 0;

}

Output

Enter details for Student 1:

Roll Number: 191921048

Name: Saran

Enter details for Student 2:

Roll Number: 191921031

Name: Ragu

Enter details for Student 3:

Roll Number: 191921021

Name: santosh raj

Student details:

Roll Number: 191921048, Name: Saran

Roll Number: 191921031, Name: Ragu

Roll Number: 191921021, Name: Santosh

Indexed file

#include <stdio.h>

#include <string.h>

struct Student {

int roll\_number;

char name[50];

};

struct Index {

int roll\_number;

long offset;

};

int main() {

struct Student student;

struct Index index;

FILE \*data\_file, \*index\_file;

data\_file = fopen("students.dat", "wb");

if (data\_file == NULL) {

printf("Error opening data file.\n");

return 1;

}

index\_file = fopen("index.dat", "wb");

if (index\_file == NULL) {

printf("Error opening index file.\n");

return 1;

}

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

printf("Enter details for Student %d:\n", i + 1);

printf("Roll Number: ");

scanf("%d", &student.roll\_number);

printf("Name: ");

scanf("%s", student.name);

fwrite(&student, sizeof(struct Student), 1, data\_file);

index.roll\_number = student.roll\_number;

index.offset = ftell(data\_file) - sizeof(struct Student);

fwrite(&index, sizeof(struct Index), 1, index\_file);

}

fclose(data\_file);

fclose(index\_file);

int roll;

printf("\nEnter Roll Number to retrieve details: ");

scanf("%d", &roll);

index\_file = fopen("index.dat", "rb");

if (index\_file == NULL) {

printf("Error opening index file.\n");

return 1;

}

int found = 0;

while (fread(&index, sizeof(struct Index), 1, index\_file)) {

if (index.roll\_number == roll) {

found = 1;

break;

}

}

if (found) {

data\_file = fopen("students.dat", "rb");

if (data\_file == NULL) {

printf("Error opening data file.\n");

return 1;

}

fseek(data\_file, index.offset, SEEK\_SET);

fread(&student, sizeof(struct Student), 1, data\_file);

printf("\nStudent details:\n");

printf("Roll Number: %d, Name: %s\n", student.roll\_number, student.name);

fclose(data\_file);

} else {

printf("Student not found.\n");

}

fclose(index\_file);

return 0;

}

Output

Enter details for Student 1:

Roll Number: 191921021

Name: santosh raj

Enter details for Student 2:

Roll Number: Name: Enter details for Student 3:

Roll Number: 191921048

Name: saran

Enter Roll Number to retrieve details: 191921048

Student details:

Roll Number: 191921048, Name: saran