**WEEK 5**

**Question 1: Write a C program to simulate Bankers algorithm for deadlock avoidance.**

**CODE:**

#include <stdio.h>

#include <stdbool.h>

struct process\_info

{

int max[10];

int allocated[10];

int need[10];

};

int no\_of\_process, no\_of\_resources;

// Take the input

void input(struct process\_info process[no\_of\_process], int available[no\_of\_resources])

{

// Fill array of Structure// make in\_use [no\_of\_resources]

for (int i = 0; i < no\_of\_process; i++)

{

printf("Enter process[%d] info\n", i);

printf("Enter Maximum Need: ");

for (int j = 0; j < no\_of\_resources; j++)

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

printf("Enter No. of Allocated Resources for this process: ");

for (int j = 0; j < no\_of\_resources; j++)

{

scanf("%d", &process[i].allocated[j]);

// calculate need/future need

process[i].need[j] = process[i].max[j] - process[i].allocated[j];

}

}

// printf("Enter Available Resources: ");

for (int i = 0; i < no\_of\_resources; i++)

{

scanf("%d", &available[i]);// available[i]-=inuse[i];

}

}

// Print the Info in Tabular Form

void showTheInfo(struct process\_info process[no\_of\_process])

{

printf("\nPID\tMaximum\t\tAllocated\tNeed\n");

for (int i = 0; i < no\_of\_process; i++)

{

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

for (int j = 0; j < no\_of\_resources; j++)

printf("%d ", process[i].max[j]);

printf("\t\t");

for (int j = 0; j < no\_of\_resources; j++)

printf("%d ", process[i].allocated[j]);

printf("\t\t");

for (int j = 0; j < no\_of\_resources; j++)

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

printf("\n");

}

}

// Apply safety algo

bool applySafetyAlgo(struct process\_info process[no\_of\_process], int available[no\_of\_resources], int safeSequence[no\_of\_process])

{

bool finish[no\_of\_process];

int work[no\_of\_resources];

for (int i = 0; i < no\_of\_resources; i++)

{

work[i] = available[i];

}

for (int i = 0; i < no\_of\_process; i++)

finish[i] = false;

bool proceed = true;

int k = 0;

while (proceed)

{

proceed = false;

for (int i = 0; i < no\_of\_process; i++)

{

bool flag = true;

// Find Index i

if (finish[i] == false)

{

for (int j = 0; j < no\_of\_resources; j++)

{

// if Need <= Work

if (process[i].need[j] <= work[j])

{

continue;

}

else

{

flag = false; // implies that the current process need > work

break;

}

}

if (flag == false)

continue; // check for next process

// If we get Index i(or process i), update work

for (int j = 0; j < no\_of\_resources; j++)

work[j] = work[j] + process[i].allocated[j];

finish[i] = true;

safeSequence[k++] = i;

proceed = true; // tells that we got atleast one process in safe state, we can proceed

}

}

}

// check finish array

int i;

for (i = 0; i < no\_of\_process && finish[i] == true; i++)

{

continue;

}

// If all processes are completed, then return true

if (i == no\_of\_process)

return true;

else

return false;

}

// Checks if we State is safe or not

bool isSafeState(struct process\_info process[no\_of\_process], int available[no\_of\_resources], int safeSequence[no\_of\_process])

{

if (applySafetyAlgo(process, available, safeSequence) == true)

return true;

return false;

}

int main()

{

printf("Enter No of Process: ");

scanf("%d", &no\_of\_process);

printf("Enter No of Resource Instances in system: ");

scanf("%d", &no\_of\_resources);

int available[no\_of\_resources];

int safeSequence[no\_of\_process];

// Create Array of Structure to store Processes's Informations

struct process\_info process[no\_of\_process];

printf("\*\*\*\*Enter details of processes\*\*\*\*\n");

// Take the Input

input(process, available);

// Print the Info in Tabular Form

// showTheInfo(process);

if (isSafeState(process, available, safeSequence))

{

printf("\nSystem is in SAFE State\n");

printf("Safe Sequence is: ");

for (int i = 0; i < no\_of\_process; i++)

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

printf("1");

}

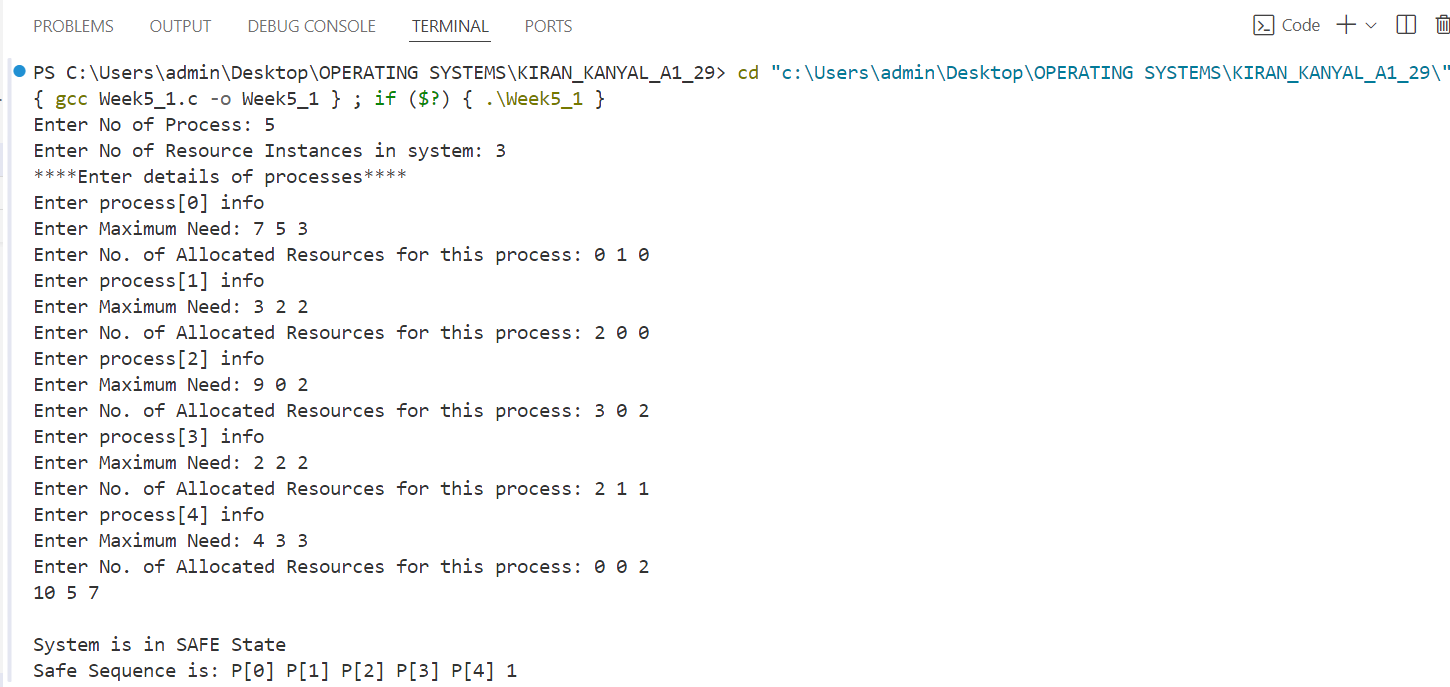
else

printf("0");

return 0;

}

**OUTPUT:**



**Question 2: Write a program to implement deadlock detection algorithm.**

**CODE:**

#include <stdio.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int allocation[MAX\_PROCESSES][MAX\_RESOURCES];

int request[MAX\_PROCESSES][MAX\_RESOURCES];

int available[MAX\_RESOURCES];

int resources[MAX\_RESOURCES];

int work[MAX\_RESOURCES];

int marked[MAX\_PROCESSES];

int main()

{

int num\_processes, num\_resources;

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

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

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

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

// Input total resources

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

{

printf("Enter the total amount of Resource R%d: ", i + 1);

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

}

// Input request matrix

printf("Enter the request matrix:\n");

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

{

for (int j = 0; j < num\_resources; j++)

{

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

}

}

// User Input allocation matrix

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

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

{

for (int j = 0; j < num\_resources; j++)

{

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

}

}

// Initialization of the available resources

for (int j = 0; j < num\_resources; j++)

{

available[j] = resources[j];

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

{

available[j] -= allocation[i][j];

}

}

// Mark processes with zero allocation

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

{

int count = 0;

for (int j = 0; j < num\_resources; j++)

{

if (allocation[i][j] == 0)

{

count++;

}

else

{

break;

}

}

if (count == num\_resources)

{

marked[i] = 1;

}

}

// Initialize work with available

for (int j = 0; j < num\_resources; j++)

{

work[j] = available[j];

}

// Mark processes with requests <= work

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

{

int can\_be\_processed = 1;

if (marked[i] != 1)

{

for (int j = 0; j < num\_resources; j++)

{

if (request[i][j] > work[j])

{

can\_be\_processed = 0;

break;

}

}

if (can\_be\_processed)

{

marked[i] = 1;

for (int j = 0; j < num\_resources; j++)

{

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

}

}

}

}

// Check for unmarked processes (deadlock)

int deadlock = 0;

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

{

if (marked[i] != 1)

{

deadlock = 1;

break;

}

}

if (deadlock)

{

printf("Deadlock detected\n");

}

else

{

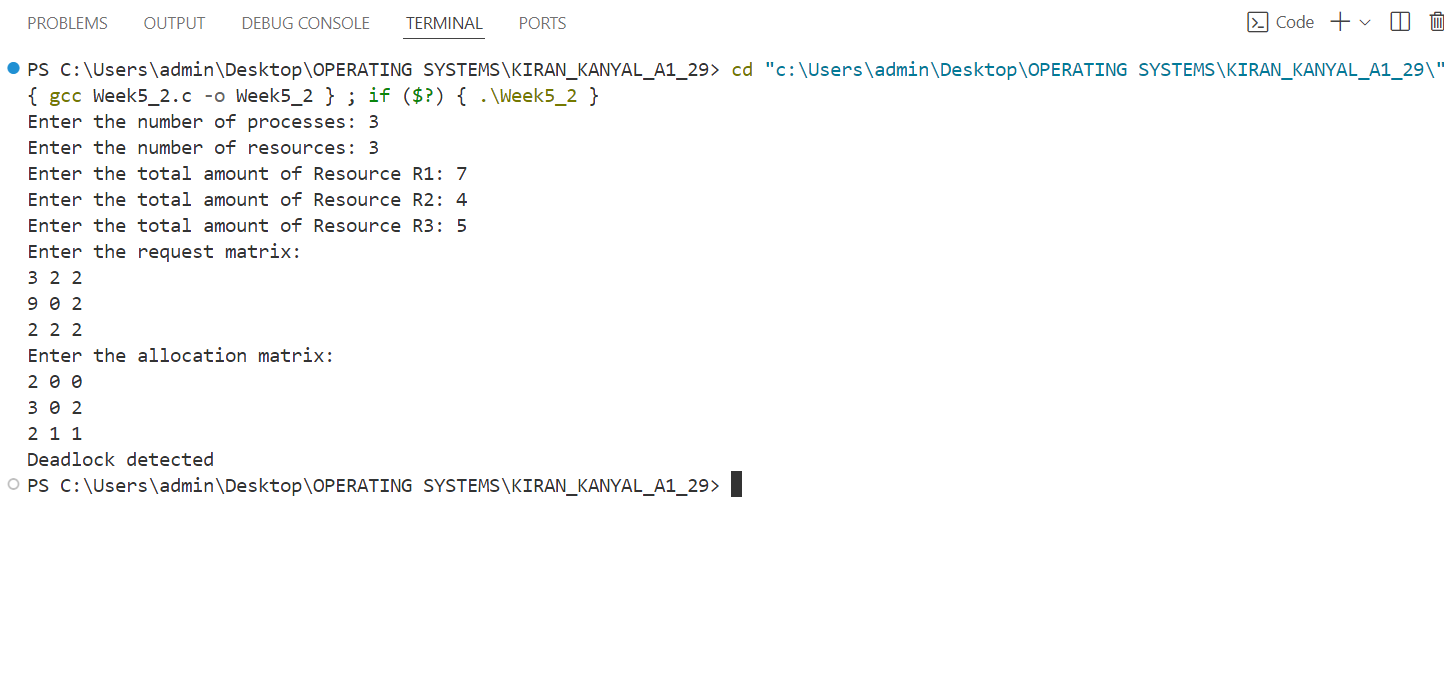
printf("No deadlock possible\n");

}

return 0;

}

**OUTPUT:**



**WEEK 6**

**Question 1: Write a program to communicate the parent and child process with each other so that whenever a child writes something, the parent process can read it. Consider the mode of communication through**

**a) pipe**

**CODE:**

#include <stdio.h>

#include <unistd.h>

#include <string.h>

int main()

{

int pipefds[2];

pid\_t pid;

char writeMsg[] = "Message from Child";

char readMsg[100];

// Create a pipe

if (pipe(pipefds) == -1)

{

perror("Pipe failed");

return 1;

}

// Fork to create a child process

pid = fork();

if (pid < 0)

{

perror("Fork failed");

return 1;

}

if (pid == 0)

{

// Child process

close(pipefds[0]); // Close reading end

write(pipefds[1], writeMsg, strlen(writeMsg) + 1);

close(pipefds[1]); // Close writing end

}

else

{

// Parent process

close(pipefds[1]); // Close writing end

read(pipefds[0], readMsg, sizeof(readMsg));

printf("Parent read: %s\n", readMsg);

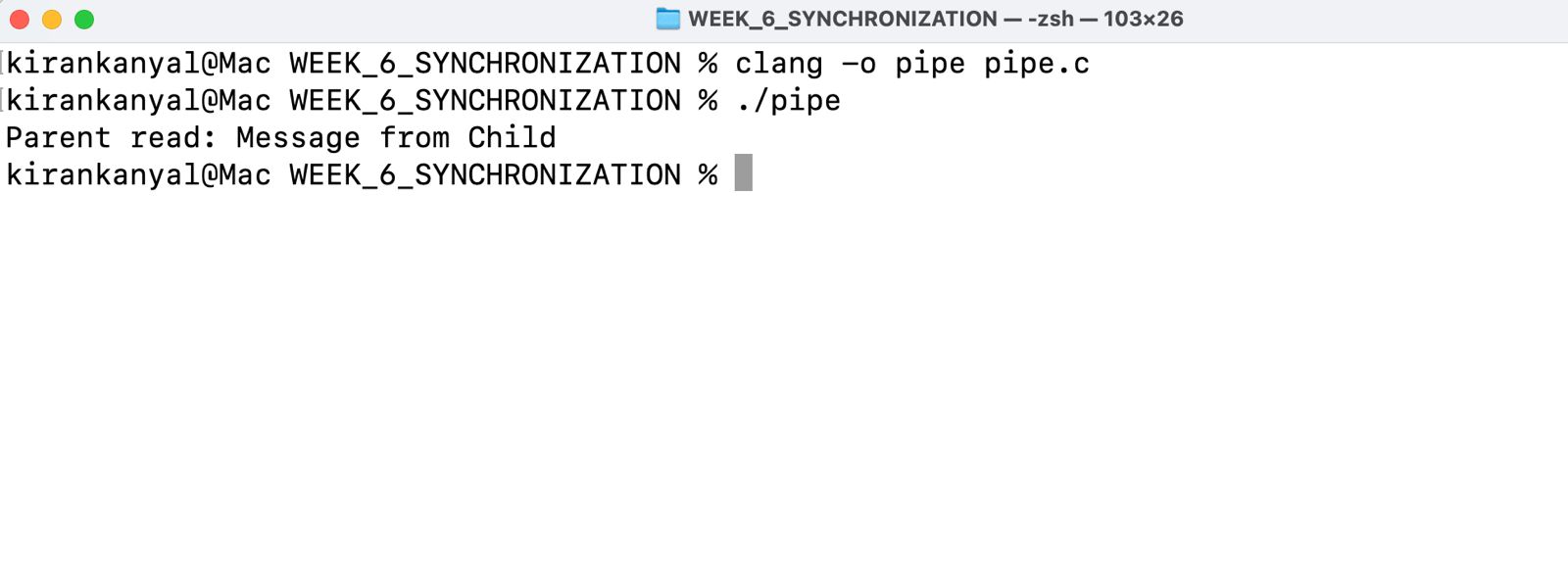
close(pipefds[0]); // Close reading end

}

return 0;

}

**OUTPUT:**



**b) message passing**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/msg.h>

struct msg\_buffer

{

long msg\_type;

char msg\_text[100];

} message;

int main()

{

key\_t key;

int msgid;

key = ftok("progfile", 65);

msgid = msgget(key, 0666 | IPC\_CREAT);

message.msg\_type = 1;

if (fork() == 0)

{

strcpy(message.msg\_text, "Message from Child");

msgsnd(msgid, &message, sizeof(message), 0);

}

else

{

msgrcv(msgid, &message, sizeof(message), 1, 0);

printf("Parent read: %s\n", message.msg\_text);

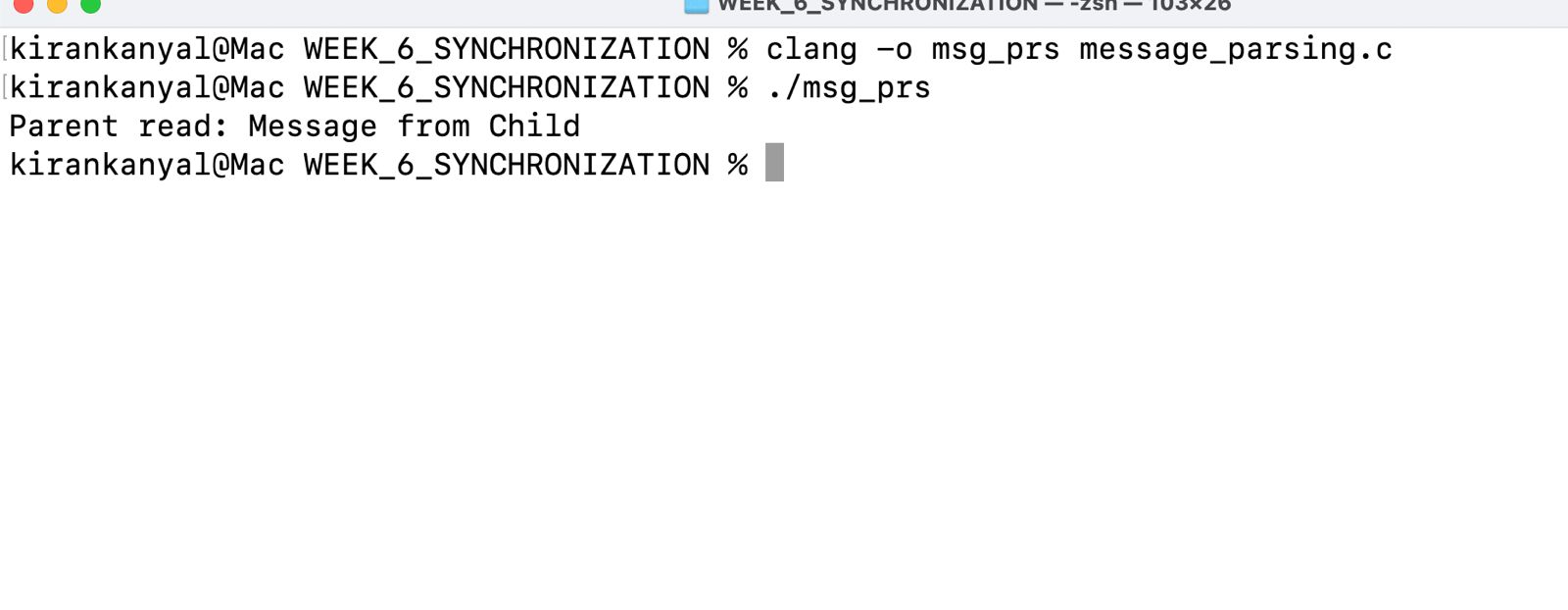
msgctl(msgid, IPC\_RMID, NULL);

}

return 0;

}

**OUTPUT:**



**c) shared memory**

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <sys/shm.h>

#include <sys/ipc.h>

#include <string.h>

#include <unistd.h>

int main() {

// Generate a unique key

key\_t key = ftok("shmfile", 65);

if (key == -1) {

perror("ftok failed");

return 1;

}

// Create a shared memory segment

int shmid = shmget(key, 1024, 0666 | IPC\_CREAT);

if (shmid == -1) {

perror("shmget failed");

return 1;

}

// Fork to create a child process

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

}

if (pid == 0) {

// Child process

// Attach to shared memory

char \*shared\_memory = (char \*)shmat(shmid, NULL, 0);

if (shared\_memory == (char \*)-1) {

perror("shmat failed");

exit(1);

}

// Write a message to the shared memory

char writeMsg[] = "Message from Child";

strcpy(shared\_memory, writeMsg);

// Detach from shared memory

shmdt(shared\_memory);

} else {

// Parent process

// Wait for child process to write

sleep(1);

// Attach to shared memory

char \*shared\_memory = (char \*)shmat(shmid, NULL, 0);

if (shared\_memory == (char \*)-1) {

perror("shmat failed");

exit(1);

}

// Read the message from shared memory

printf("Parent read: %s\n", shared\_memory);

// Detach from shared memory

shmdt(shared\_memory);

// Destroy the shared memory segment

shmctl(shmid, IPC\_RMID, NULL);

}

return 0;

}

**OUTPUT:**

**Question 2: Write a program to implement the concept of the Producer-Consumer problem using semaphores**

**CODE:**

**OUTPUT:**

**Question 3: Write a program to implement the concept of Dining-Philosopher problem.**

**CODE:**

#include <stdio.h>

// Arrays to represent philosophers and chopstick status

int p[5]; // Philosopher states: 0 = Thinking, 1 = Eating

int ch[5]; // Chopstick states: 0 = Free, 1 = In use

// Function to release chopsticks (signal)

void signal(int y) {

int right = (y + 1) % 5;

p[y] = 0; // Philosopher stops eating

ch[y] = 0; // Left chopstick is released

ch[right] = 0; // Right chopstick is released

}

// Function to acquire chopsticks (wait)

void wait(int y) {

int right = (y + 1) % 5;

// Check if both chopsticks are free

if (ch[y] == 0 && ch[right] == 0) {

p[y] = 1; // Philosopher starts eating

ch[y] = 1; // Mark left chopstick as in use

ch[right] = 1; // Mark right chopstick as in use

} else if (p[y] == 1) {

// Philosopher is already eating

int w;

printf("Do you want philosopher %d to stop eating? (1 for Yes, 0 for No): ", y);

scanf("%d", &w);

if (w == 1) {

signal(y); // Release chopsticks

}

} else {

// Chopsticks are busy, philosopher has to wait

printf("Chopsticks %d and %d are busy.\n", y, right);

printf("Philosopher %d has to wait.\n", y);

}

}

int main() {

int u; // Continue flag

// Initialize philosophers and chopsticks

for (int i = 0; i < 5; i++) {

p[i] = 0; // All philosophers start thinking

ch[i] = 0; // All chopsticks are free

}

do {

// Print the current state of philosophers

for (int i = 0; i < 5; i++) {

if (p[i] == 0) {

printf("Philosopher %d: Thinking\n", i);

} else {

printf("Philosopher %d: Eating\n", i);

}

}

int s;

printf("Which philosopher wants to eat? (Enter 0-4): ");

scanf("%d", &s);

if (s >= 0 && s < 5) {

wait(s); // Attempt to make the selected philosopher eat

} else {

printf("Invalid input. Please enter a number between 0 and 4.\n");

}

printf("\nDo you want to continue? (1 for Yes, 0 for No): ");

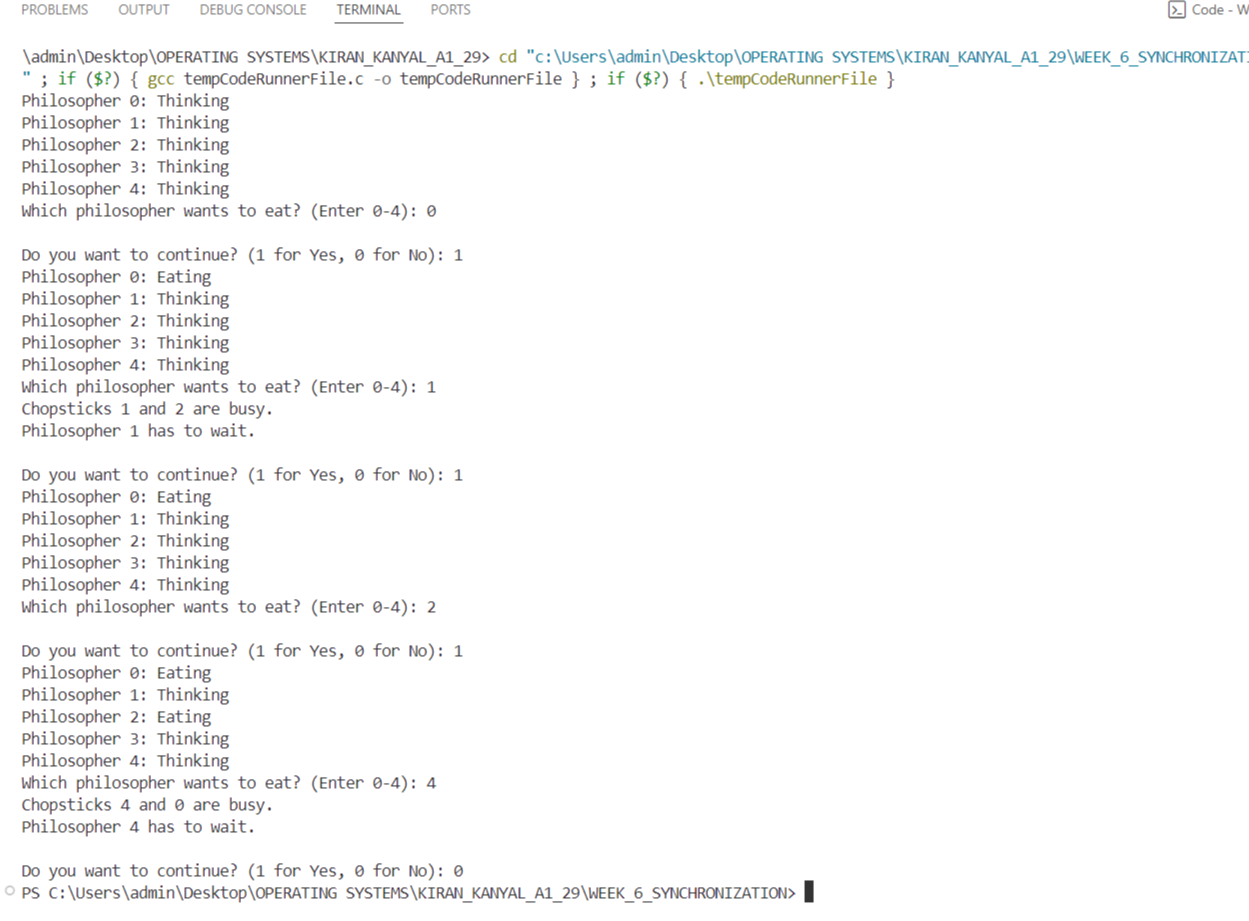
scanf("%d", &u);

} while (u == 1);

return 0;

}

**OUTPUT:**



**WEEK 7**

**Question 1: Write a C program to implement FIFO page replacement algorithm.**

**CODE:**

#include<stdio.h>

#include<stdbool.h>

#include<string.h>

struct PageTable{

int frame\_no;

bool valid;

};

bool isPagePresent(struct PageTable PT[], int page){

if (PT[page].valid == 1){

return true;

}

return false;

}

void updatePageTable (struct PageTable PT[], int page, int fr\_no, int status){

PT[page].valid = status;

if (status == 1) PT[page].frame\_no = fr\_no;

}

void printFrameContents (int frame[], int no\_of\_frames){

for (int i=0; i<no\_of\_frames; i++){

if (frame[i] != -1) printf("%d ",frame[i]);

}

printf("\n");

}

int main(){

int n, no\_of\_frames;

printf("Enter the number of pages (page requests): ");

scanf("%d", &n);

int reference\_string[n];

printf("\nEnter the reference string(different page numbers):\n");

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

scanf("%d",&reference\_string[i]);

}

printf("Enter the no. of frames you want to give to the process:");

scanf("%d", &no\_of\_frames);

int frame[no\_of\_frames];

memset(frame,-1,no\_of\_frames\*sizeof(int));

struct PageTable PT[50];

for (int i=0; i<50; i++){

PT[i].valid = 0;

}

printf("\n\*\*\*The contents inside the Frame array at different time: \*\*\*\n");

int page\_fault = 0, current = 0, flag = false;

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

if (!isPagePresent(PT, reference\_string[i])){

page\_fault++;

if (flag==false && current<no\_of\_frames){

frame[current] = reference\_string[i];

updatePageTable(PT, reference\_string[i],current,1);

current+=1;

if (current==no\_of\_frames){

current = 0;

flag = true;

}

}

else{

updatePageTable(PT,frame[current],-1,0);

frame[current] = reference\_string[i];

updatePageTable(PT, reference\_string[i], current,1);

current = (current+1)%no\_of\_frames;

}

printFrameContents(frame, no\_of\_frames);

}

}

printf("\nTotal No. of Page Faults = %d\n",page\_fault);

printf("\nPage Fault Ratio = %.2f\n", (float) page\_fault/n);

printf("\nPage Hit Ratio = %.2f", (float) (n-page\_fault)/n);

return 0;

}

**OUTPUT:**



**Question 2: Write a C program to implement the LRU page replacement algorithm.**

**CODE:**

#include<stdio.h>

#include<stdbool.h>

#include<string.h>

#include<limits.h>

struct PageTable{

int frame\_no;

int last\_time\_of\_access;

bool valid;

};

bool isPagePresent (struct PageTable PT[], int page){

if (PT[page].valid == 1) return true;

return false;

}

void updatePageTable (struct PageTable PT[], int page, int fr\_no, int status, int access\_time){

PT[page].valid = status;

if (status == 1){

PT[page].frame\_no = fr\_no;

PT[page].last\_time\_of\_access = access\_time;

}

}

void printFrameContents(int frame[], int no\_of\_frames){

for (int i=0; i<no\_of\_frames; i++){

if (frame[i]!=-1) printf("%d ",frame[i]);

}

printf("\n");

}

int searchLRUPage (struct PageTable PT[], int frame[],int no\_of\_frames){

int idx = -1;

for (int i=0; i<no\_of\_frames; i++){

if (idx==-1 || PT[frame[i]].last\_time\_of\_access < PT[frame[idx]].last\_time\_of\_access){

idx = i;

}

}

return idx;

}

int main(){

int n;

printf("Enter the no. of pages: ");

scanf("%d", &n);

int reference\_string[n];

printf("Enter the reference string (different page numbers):\n");

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

scanf("%d", &reference\_string[i]);

}

int no\_of\_frames;

printf("Enter the no. of frames you want to give to the process: ");

scanf("%d", &no\_of\_frames);

int frame[no\_of\_frames];

memset(frame, -1, no\_of\_frames\*sizeof(int));

struct PageTable PT[50];

for (int i=0; i<50; i++){

PT[i].valid = 0;

}

printf("\*\*\*The Contents inside the Frame array at different time.\*\*\*\n");

int pageFault =0, current = 0;

bool flag = false;

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

if (!isPagePresent(PT, reference\_string[i])){

pageFault++;

if (flag==false && current<no\_of\_frames){

frame[current] = reference\_string[i];

updatePageTable(PT,reference\_string[i],current,1,i);

current = current + 1;

if (current == no\_of\_frames){

flag = true;

}

}

else{

int LRU\_page\_index = searchLRUPage (PT,frame,no\_of\_frames);

updatePageTable(PT,frame[LRU\_page\_index],-1,0,i);

frame[LRU\_page\_index] = reference\_string[i];

updatePageTable(PT, reference\_string[i],LRU\_page\_index,1,i);

}

}

printFrameContents(frame, no\_of\_frames);

PT[reference\_string[i]].last\_time\_of\_access = i;

}

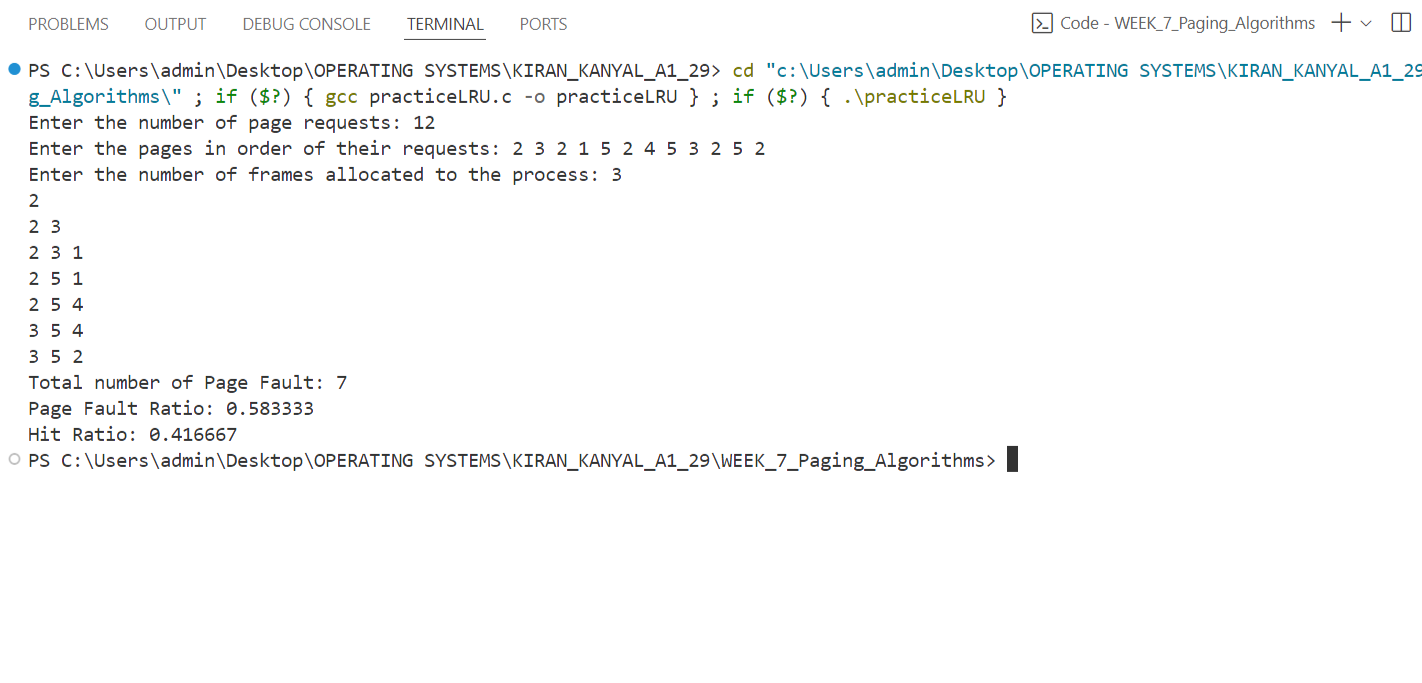
printf("Total number of Page Fault: %d\n",pageFault);

printf("Page Fault Ratio: %f\n", (float)(pageFault)/n);

printf("Hit Ratio: %f\n", (float) (n-pageFault)/n);

}

**OUTPUT:**



**WEEK 8**

**Question 1: Write a C program to implement Best Fit Memory Allocation Techniques.**

**CODE:**

#include<stdio.h>

#include<string.h>

#include<limits.h>

int main(){

int no\_of\_blocks;

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

scanf("%d",&no\_of\_blocks);

int blocks[no\_of\_blocks];

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

for (int i=0; i<no\_of\_blocks; i++){

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

}

int no\_of\_processes;

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

scanf("%d", &no\_of\_processes);

int given\_block[no\_of\_processes],process[no\_of\_processes];

memset(given\_block,-1,sizeof(int)\*no\_of\_processes);

printf("Enter the memory requirement of each process: \n");

for (int i=0; i<no\_of\_processes; i++){

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

}

for (int i=0; i<no\_of\_processes; i++){

int minFragmentation = INT\_MAX;

int blockNo=-1;

for (int j=0; j<no\_of\_blocks; j++){

if (blocks[j]>=process[i]){

if (minFragmentation > (blocks[j] - process[i])){

minFragmentation = blocks[j]-process[i];

blockNo = j;

}

}

}

if (blockNo!=-1){

given\_block[i] = blockNo+1;

blocks[blockNo] = 0;

}

}

for (int i=0; i<no\_of\_processes; i++){

if (given\_block[i] != -1){

printf("Memory block assigned to Process %d: %d\n",process[i],given\_block[i]);

}

else{

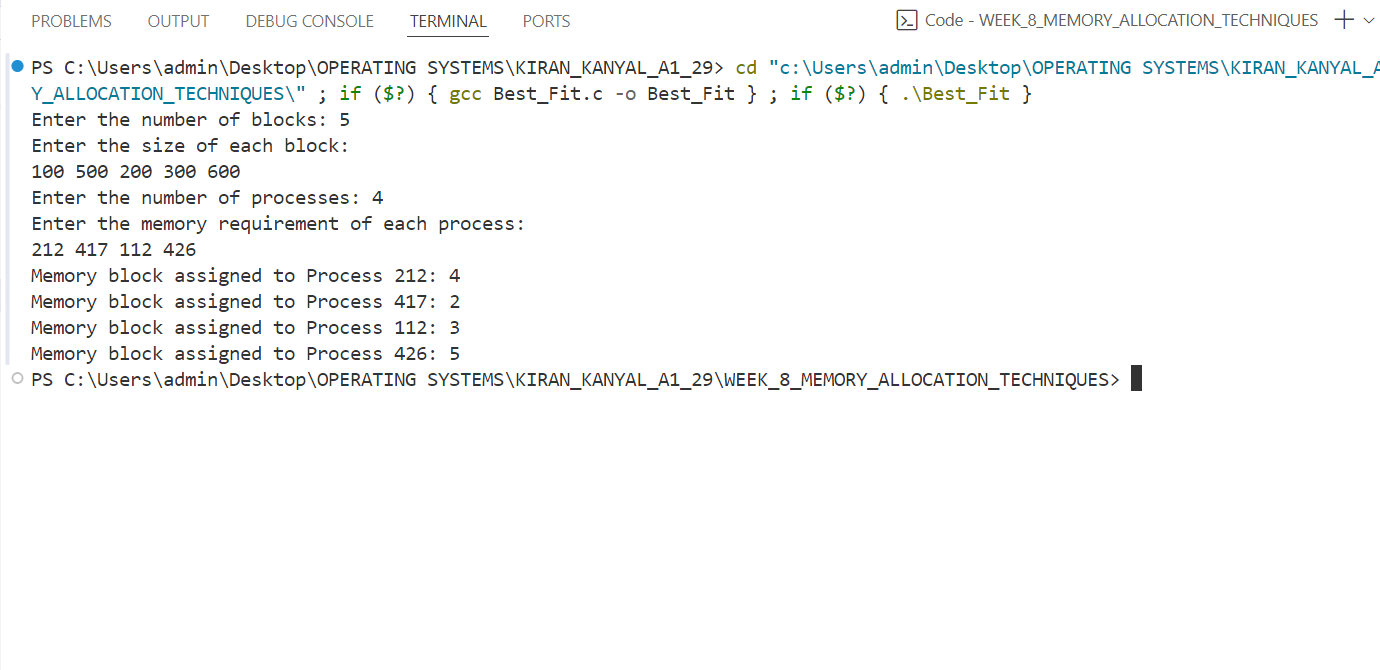
printf("No memory block could be assigned.\n");

}

}

}

**OUTPUT:**



**Question 2: Write a C program to implement First Fit Memory Allocation Technique.**

**CODE:**

#include<stdio.h>

#include<string.h>

int main(){

int no\_of\_blocks;

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

scanf("%d",&no\_of\_blocks);

int blocks[no\_of\_blocks];

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

for (int i=0; i<no\_of\_blocks; i++){

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

}

int no\_of\_processes;

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

scanf("%d", &no\_of\_processes);

int given\_block[no\_of\_processes],process[no\_of\_processes];

memset(given\_block,-1,sizeof(int)\*no\_of\_processes);

printf("Enter the memory requirement of each process: \n");

for (int i=0; i<no\_of\_processes; i++){

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

}

for (int i=0; i<no\_of\_processes; i++){

for (int j=0; j<no\_of\_blocks; j++){

if (blocks[j]>=process[i]){

given\_block[i] = j+1;

blocks[j] = 0;

break;

}

}

}

for (int i=0; i<no\_of\_processes; i++){

if (given\_block[i] != -1){

printf("Memory block assigned to Process %d: %d\n",process[i],given\_block[i]);

}

else{

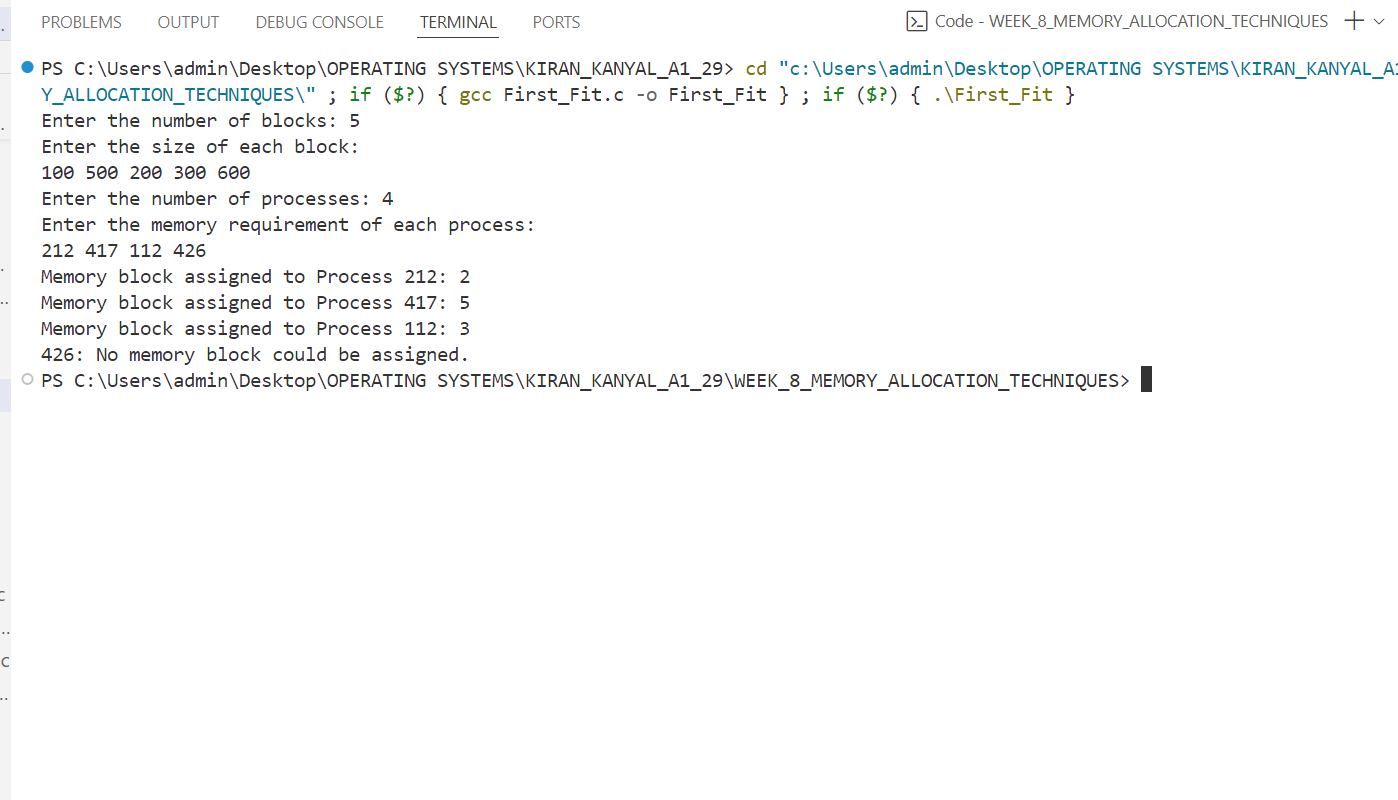
printf("%d: No memory block could be assigned.\n",process[i]);

}

}

}

**OUTPUT:**



**Question 3: Write a program to implement Worst Fit Memory Allocation Technique.**

**CODE:**

#include<stdio.h>

#include<string.h>

#include<limits.h>

int main(){

int no\_of\_blocks;

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

scanf("%d",&no\_of\_blocks);

int blocks[no\_of\_blocks];

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

for (int i=0; i<no\_of\_blocks; i++){

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

}

int no\_of\_processes;

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

scanf("%d", &no\_of\_processes);

int given\_block[no\_of\_processes],process[no\_of\_processes];

memset(given\_block,-1,sizeof(int)\*no\_of\_processes);

printf("Enter the memory requirement of each process: \n");

for (int i=0; i<no\_of\_processes; i++){

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

}

for (int i=0; i<no\_of\_processes; i++){

int maxFragmentation = INT\_MIN;

int blockNo=-1;

for (int j=0; j<no\_of\_blocks; j++){

if (blocks[j]>=process[i]){

if (maxFragmentation < (blocks[j] - process[i])){

maxFragmentation = blocks[j]-process[i];

blockNo = j;

}

}

}

if (blockNo!=-1){

given\_block[i] = blockNo+1;

blocks[blockNo] =0;

}

}

for (int i=0; i<no\_of\_processes; i++){

if (given\_block[i] != -1){

printf("Memory block assigned to Process %d: %d\n",process[i],given\_block[i]);

}

else{

printf("%d: No memory block could be assigned.\n",process[i]);

}

}

}

**OUTPUT:**

