1)

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

pid\_t pid = fork();

if (pid == 0)

{

printf("Child PID: %d\n", getpid());

printf("Parent PID: %d\n", getppid());

}

else if (pid > 0)

{

printf("Parent PID: %d\n", getpid());

}

else

{

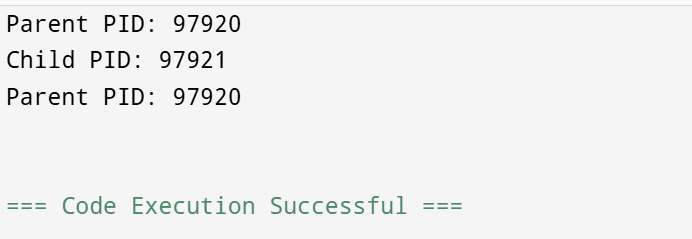
perror("Fork failed");

}

return 0;

}

**OUT PUT:**

****

**2)**

#include <stdio.h>

typedef struct

{

int id, burst\_time, waiting\_time, turnaround\_time;

}Process;

void calculateTimes(Process p[], int n)

{

p[0].waiting\_time = 0;

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

{

p[i].waiting\_time = p[i - 1].waiting\_time + p[i - 1].burst\_time;

}

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

{

p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time;

}

}

void printProcesses(Process p[], int n)

{

printf("ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

{

printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst\_time, p[i].waiting\_time, p[i].turnaround\_time);

}

}

int main()

{

Process p[] = {{1, 5}, {2, 3}, {3, 8}};

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

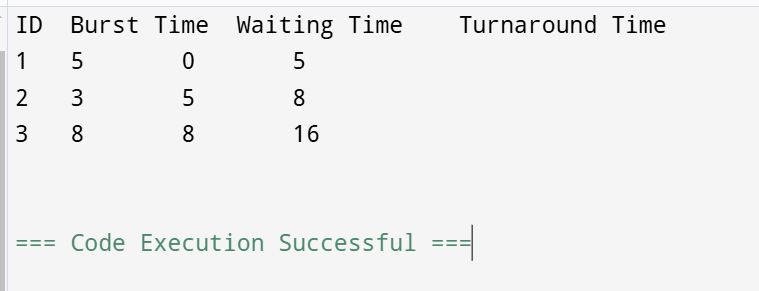
calculateTimes(p, n);

printProcesses(p, n);

return 0;

}

**OUT PUT:**

****

**3)**

#include <stdio.h>

typedef struct {

int id, burst\_time, waiting\_time, turnaround\_time;

} Process;

int main() {

Process p[] = {{1, 5}, {2, 3}, {3, 8}};

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

p[0].waiting\_time = 0;

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

p[i].waiting\_time = p[i - 1].waiting\_time + p[i - 1].burst\_time;

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

p[i].turnaround\_time = p[i].waiting\_time + p[i].burst\_time;

printf("ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

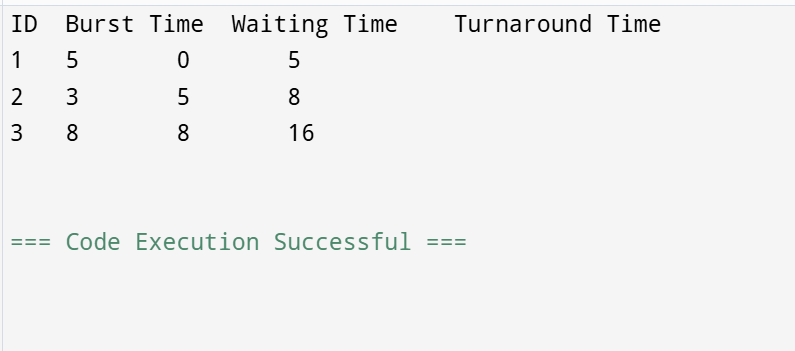
for (int i = 0; i < n; i++)

printf("%d\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burst\_time, p[i].waiting\_time, p[i].turnaround\_time);

return 0;

}

**OUT PUT:**

****

**4)**

#include <stdio.h>

typedef struct

{

int id;

int exec\_time;

} Process;

void schedule(Process proc[], int n)

{

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

{

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

{

if (proc[j].exec\_time > proc[j + 1].exec\_time)

{

Process temp = proc[j];

proc[j] = proc[j + 1];

proc[j + 1] = temp;

}

}

}

}

int main()

{

Process proc[] = {{1, 5}, {2, 2}, {3, 8}, {4, 1}};

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

schedule(proc, n);

printf("Scheduled Processes:\n");

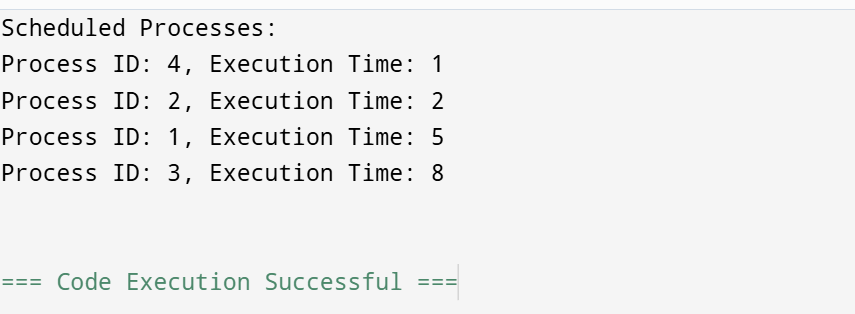
for (int i = 0; i < n; i++) {

printf("Process ID: %d, Execution Time: %d\n", proc[i].id, proc[i].exec\_time);

}

return 0;

}

**OUT PUT:** ****

5)

#include <stdio.h>

#include <stdlib.h>

#define MAX\_PROCESSES 10

typedef struct

{

int id;

int priority;

} Process;

void schedule(Process processes[], int n)

{

int highestPriorityIndex = 0;

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

if (processes[i].priority > processes[highestPriorityIndex].priority)

{

highestPriorityIndex = i;

}

}

printf("Executing Process ID: %d with Priority: %d\n",

processes[highestPriorityIndex].id,

processes[highestPriorityIndex].priority);

}

int main()

{

Process processes[MAX\_PROCESSES] =

{

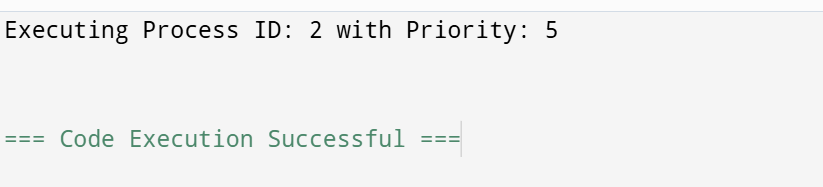
{1, 2}, {2, 5}, {3, 1}, {4, 4}, {5, 3}

};

schedule(processes, 5);

return 0;

}

**OUT PUT:** ****

6)

#include <stdio.h>

struct Process

{

int id, bt, pt, wt, tat;

};

void sort(struct Process p[], int n)

{

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

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

if (p[j].pt > p[j + 1].pt) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

}

int main()

{

struct Process p[] = {{1, 10, 2}, {2, 1, 1}, {3, 2, 3}};

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

sort(p, n);

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

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

{

p[i].wt = total\_wt;

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

total\_wt += p[i].bt;

total\_tat += p[i].tat;

printf("Process %d: WT = %d, TAT = %d\n", p[i].id, p[i].wt, p[i].tat);

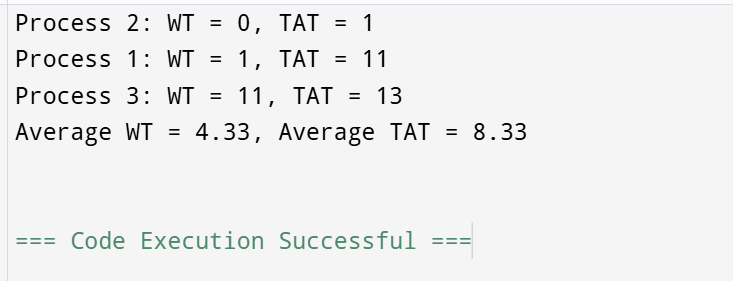
}

printf("Average WT = %.2f, Average TAT = %.2f\n", (float)total\_wt/n, (float)total\_tat/n);

return 0;

}

**OUT PUT:**

****

**7)**

#include <stdio.h>

int main() {

int n, i, j, temp;

printf("Enter number of processes: ");

scanf("%d", &n);

int bt[n], wt[n], tat[n];

printf("Enter burst times:\n");

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

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

}

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

{

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

{

if(bt[i] > bt[j])

{

temp = bt[i]; bt[i] = bt[j]; bt[j] = temp;

}

}

}

wt[0] = 0;

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

{

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

}

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

{

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

}

printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

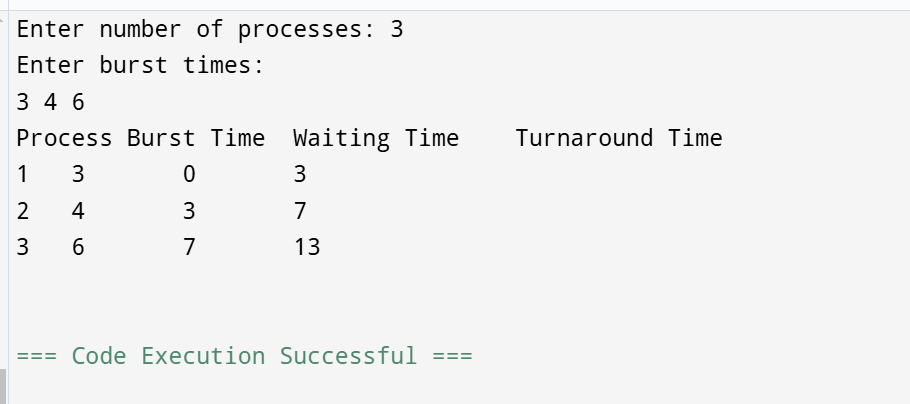
{

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

}

return 0;

}



8)

#include <stdio.h>

int main()

{

int n, i, j, time, quantum;

printf("Enter number of processes: ");

scanf("%d", &n);

int burst[n], wait[n], turn[n];

printf("Enter burst times:\n");

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

{

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

}

printf("Enter time quantum: ");

scanf("%d", &quantum);

for(time = 0; ; time += quantum)

{

int done = 1;

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

{

if(burst[i] > 0)

{

done = 0;

if(burst[i] > quantum)

{

burst[i] -= quantum;

}

else

{

wait[i] = time;

turn[i] = time + burst[i];

burst[i] = 0;

}

}

}

if(done) break;

}

printf("Process\tWaiting Time\tTurnaround Time\n");

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

{

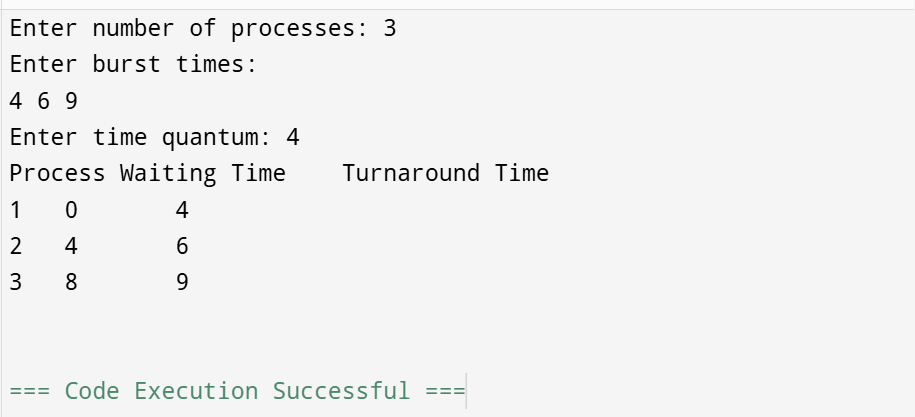
printf("%d\t%d\t\t%d\n", i + 1, wait[i], turn[i]);

}

return 0;

}

**OUT PUT:**

****

**9)**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

int main()

{

int shmid;

key\_t key = 1234;

char \*str;

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

if (shmid < 0)

{

perror("shmget");

exit(1);

}

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

if (str == (char\*) -1)

{

perror("shmat");

exit(1);

}

sprintf(str, "Hello from process %d", getpid());

printf("Data written: %s\n", str);

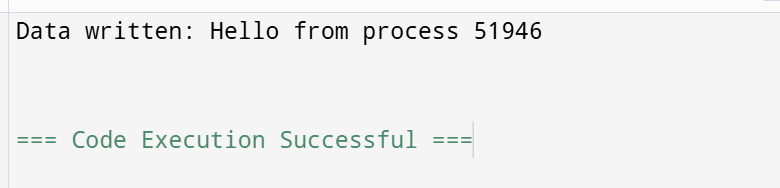
shmdt(str);

shmctl(shmid, IPC\_RMID, NULL);

return 0;

}

**OUT PUT:**

****

**10)**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#include <unistd.h>

#define MSG\_SIZE 100

struct msg\_buffer

{

long msg\_type;

char msg\_text[MSG\_SIZE];

};

int main()

{

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

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

struct msg\_buffer message;

message.msg\_type = 1;

strcpy(message.msg\_text, "Hello, World!");

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

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

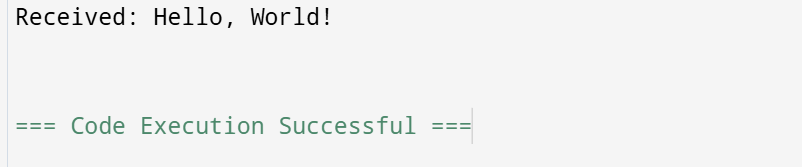
printf("Received: %s\n", message.msg\_text);

msgctl(msgid, IPC\_RMID, NULL);

return 0;

}

**OUTPUT :**

****

**11)**

#include <stdio.h>

#include <pthread.h>

void\* printMessage(void\* msg) {

printf("%s\n", (char\*)msg);

return NULL;

}

int main() {

pthread\_t thread1, thread2;

char\* message1 = "Hello from Thread 1";

char\* message2 = "Hello from Thread 2";

pthread\_create(&thread1, NULL, printMessage, (void\*)message1);

pthread\_create(&thread2, NULL, printMessage, (void\*)message2);

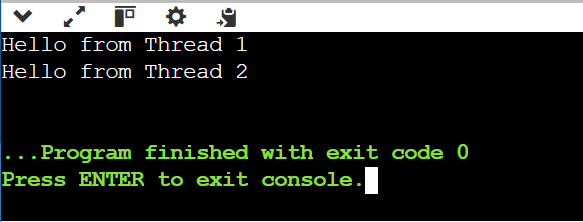
pthread\_join(thread1, NULL);

pthread\_join(thread2, NULL);

return 0;

}

**OUT PUT:**



12)

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#define PHILOSOPHERS 3

pthread\_mutex\_t forks[PHILOSOPHERS];

void\* philosopher(void\* num)

{

int id = \*(int\*)num;

while (1)

{

pthread\_mutex\_lock(&forks[id]);

pthread\_mutex\_lock(&forks[(id + 1) % PHILOSOPHERS]);

printf("Philosopher %d is eating.\n", id);

sleep(1);

pthread\_mutex\_unlock(&forks[id]);

pthread\_mutex\_unlock(&forks[(id + 1) % PHILOSOPHERS]);

printf("Philosopher %d is thinking.\n", id);

sleep(1);

}

}

int main()

{

pthread\_t philosophers[PHILOSOPHERS];

int ids[PHILOSOPHERS];

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

{

pthread\_mutex\_init(&forks[i], NULL);

ids[i] = i;

pthread\_create(&philosophers[i], NULL, philosopher, &ids[i]);

}

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

{

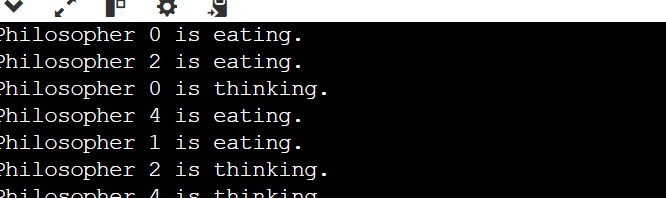
pthread\_join(philosophers[i], NULL);

}

return 0;

}

**OUT PUT:**



13)

#include <stdio.h>

#include <stdlib.h>

void staticAllocation() {

int arr[5] = {1, 2, 3, 4, 5};

printf("Static Allocation: ");

for(int i = 0; i < 5; i++) printf("%d ", arr[i]);

printf("\n");

}

void stackAllocation() {

int arr[5];

for(int i = 0; i < 5; i++) arr[i] = i + 1;

printf("Stack Allocation: ");

for(int i = 0; i < 5; i++) printf("%d ", arr[i]);

printf("\n");

}

void dynamicAllocation() {

int \*arr = (int \*)malloc(5 \* sizeof(int));

for(int i = 0; i < 5; i++) arr[i] = i + 1;

printf("Dynamic Allocation: ");

for(int i = 0; i < 5; i++) printf("%d ", arr[i]);

free(arr);

printf("\n");

}

int main() {

staticAllocation();

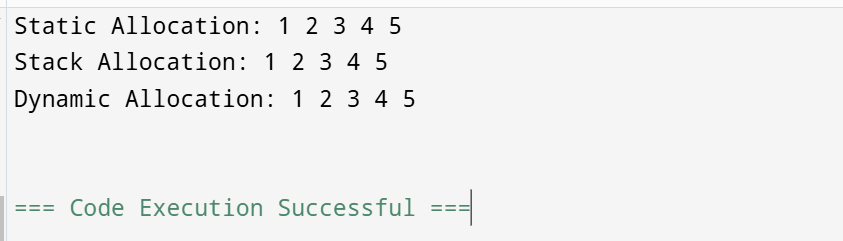
stackAllocation();

dynamicAllocation();

return 0;

}

**OUT PUT:**



14)

#include <stdio.h>

#include <dirent.h>

int main() {

struct dirent \*entry;

DIR \*dp = opendir(".");

if (dp == NULL) {

perror("opendir");

return 1;

}

while ((entry = readdir(dp))) {

if (entry->d\_type == DT\_REG)

printf("%s\n", entry->d\_name);

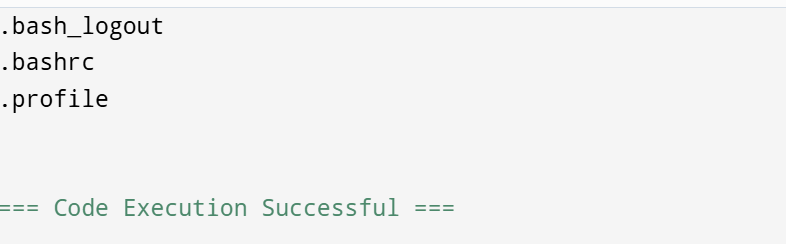
}

closedir(dp);

return 0;

}

**OUT PUT:**



15)

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_FILES 100

#define MAX\_NAME\_LENGTH 50

typedef struct {

char name[MAX\_NAME\_LENGTH];

} File;

typedef struct {

char name[MAX\_NAME\_LENGTH];

File files[MAX\_FILES];

int fileCount;

} Directory;

void createDirectory(Directory \*dir, const char \*name) {

strncpy(dir->name, name, MAX\_NAME\_LENGTH);

dir->fileCount = 0;

}

void addFile(Directory \*dir, const char \*fileName) {

if (dir->fileCount < MAX\_FILES) {

strncpy(dir->files[dir->fileCount].name, fileName, MAX\_NAME\_LENGTH);

dir->fileCount++;

}

}

void displayDirectory(const Directory \*dir) {

printf("Directory: %s\n", dir->name);

for (int i = 0; i < dir->fileCount; i++) {

printf(" File: %s\n", dir->files[i].name);

}

}

int main() {

Directory root, subDir;

createDirectory(&root, "Root");

createDirectory(&subDir, "SubDirectory");

addFile(&root, "file1.txt");

addFile(&root, "file2.txt");

addFile(&subDir, "file3.txt");

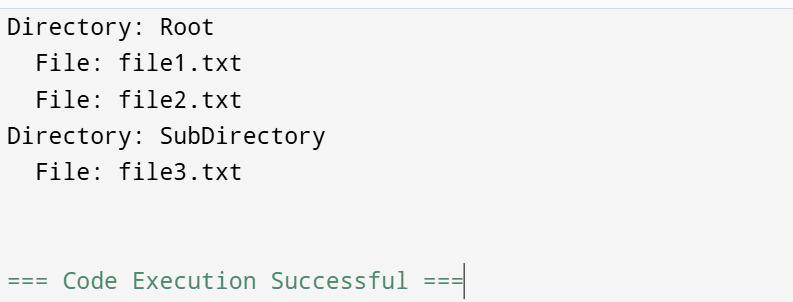
displayDirectory(&root);

displayDirectory(&subDir);

return 0;

}

**OUT PUT:**

****

16)

#include <stdio.h>

#include <stdlib.h>

struct Employee {

int id;

char name[30];

float salary;

};

void writeEmployee(FILE \*file, struct Employee emp, int index) {

fseek(file, index \* sizeof(struct Employee), SEEK\_SET);

fwrite(&emp, sizeof(struct Employee), 1, file);

}

struct Employee readEmployee(FILE \*file, int index) {

struct Employee emp;

fseek(file, index \* sizeof(struct Employee), SEEK\_SET);

fread(&emp, sizeof(struct Employee), 1, file);

return emp;

}

int main() {

FILE \*file = fopen("employees.dat", "r+b");

if (!file) file = fopen("employees.dat", "w+b");

struct Employee emp1 = {1, "John Doe", 50000.0};

writeEmployee(file, emp1, 0);

struct Employee emp2 = readEmployee(file, 0);

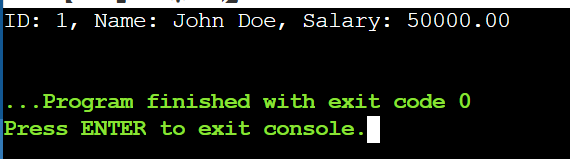
printf("ID: %d, Name: %s, Salary: %.2f\n", emp2.id, emp2.name, emp2.salary);

fclose(file);

return 0;

}

**OUT PUT:**

****

17)

#include <stdio.h>

#include <stdbool.h>

#define P 5 // Number of processes

#define R 3 // Number of resources

int allocation[P][R], max[P][R], need[P][R], available[R];

bool isSafe() {

int work[R], finish[P] = {0};

for (int i = 0; i < R; i++) work[i] = available[i];

for (int found = 1; found;) {

found = 0;

for (int p = 0; p < P; p++) {

if (!finish[p]) {

bool canAllocate = true;

for (int r = 0; r < R; r++)

if (need[p][r] > work[r]) { canAllocate = false; break; }

if (canAllocate) {

for (int r = 0; r < R; r++) work[r] += allocation[p][r];

finish[p] = 1;

found = 1;

}

}

}

}

for (int p = 0; p < P; p++) if (!finish[p]) return false;

return true;

}

int main() {

int totalResources[R] = {10, 5, 7};

int initialAllocation[P][R] = {{0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1}, {0, 0, 2}};

int maxDemand[P][R] = {{7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2}, {4, 3, 3}};

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

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

allocation[i][j] = initialAllocation[i][j];

max[i][j] = maxDemand[i][j];

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

}

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

available[j] = totalResources[j];

for (int i = 0; i < P; i++) available[j] -= allocation[i][j];

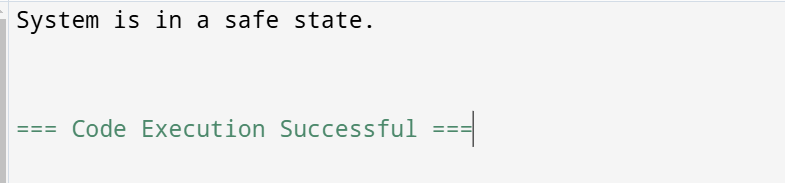
}

printf("System is %s safe state.\n", isSafe() ? "in a" : "not in a");

return 0;

}

**OUT PUT:**

****

18)

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE], in = 0, out = 0;

sem\_t empty, full;

void\* producer(void\* arg) {

for (int i = 0; i < 10; i++) {

sem\_wait(&empty);

buffer[in] = i;

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

in = (in + 1) % BUFFER\_SIZE;

sem\_post(&full);

sleep(1);

}

return NULL;

}

void\* consumer(void\* arg) {

for (int i = 0; i < 10; i++) {

sem\_wait(&full);

int item = buffer[out];

printf("Consumed: %d\n", item);

out = (out + 1) % BUFFER\_SIZE;

sem\_post(&empty);

sleep(1);

}

return NULL;

}

int main() {

pthread\_t prod, cons;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

pthread\_create(&prod, NULL, producer, NULL);

pthread\_create(&cons, NULL, consumer, NULL);

pthread\_join(prod, NULL);

pthread\_join(cons, NULL);

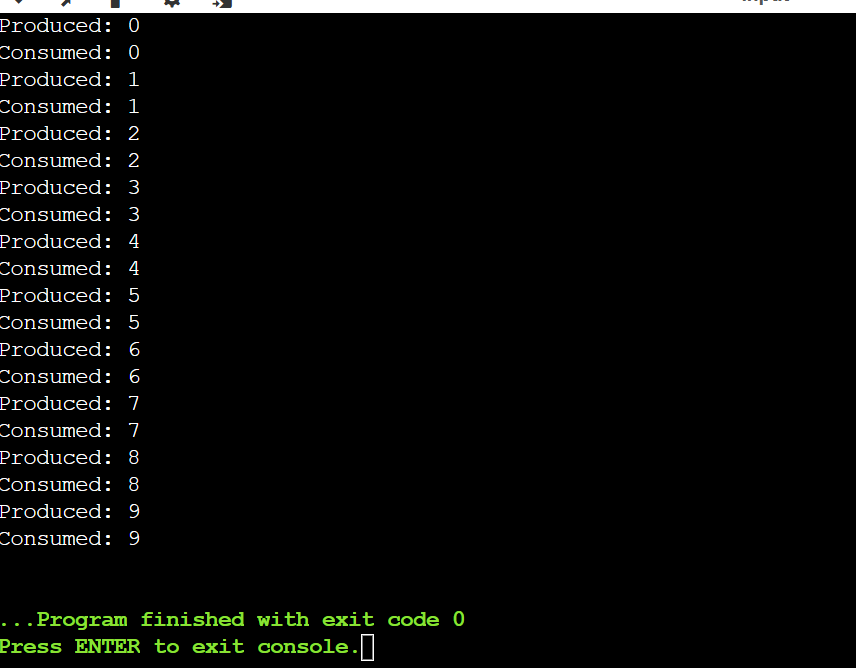
sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

**OUT PUT:**

****