**PROGRAM 1.**

Write a program to demonstrate various process related concept

#include <stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/wait.h>

int main()

{

int pid=fork();

if(pid==0)

{

printf("child=>PPID%d PID %d\n",getppid(),getpid());

exit(EXIT\_SUCCESS);

}

else if(pid>0)

{

printf("parent=>PID %d\n",getpid());

printf("waiting for child process\n");

wait(NULL);

printf("child proces finished");

}

else{

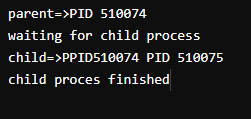
printf("unable to create child process");

}

return EXIT\_SUCCESS;

}

**OUTPUT:-**



**PROGRAM 2**

write a c program to demonstrate thread related concepts

#include<stdio.h>

#include<pthread.h>

void\*routine()

{

printf("test for heading\n");

printf("test for heading\n");

printf("ending\n");

}

int main()

{

pthread\_t t1,t2;

pthread\_create(&t1,NULL,&routine,NULL);

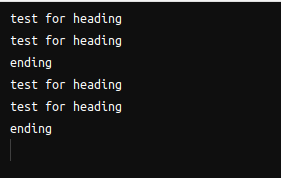
pthread\_create(&t2,NULL,&routine,NULL);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

}

**OUTPUT:-**



PROGRAM 3(B).

write a c program to simulate INTER PROCESS COMMUNICATION (IPC) TECHNIQUE shared memory

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/shm.h>

#include <string.h>

int main() {

int i,shmid;

void\*shared\_memory;

char buff[100];

// id, storge, r/w permission

shmid=shmget((key\_t)2345,1024,0666|IPC\_CREAT);

printf("KEY OF SHARED MEMORY is %d\n",shmid);

shared\_memory=shmat(shmid,NULL,0); // shared memory attached

printf("process attached at %p\n",shared\_memory);

printf("enter some data\n");

read(0,buff,100); //enter input from user

strcpy(shared\_memory,buff);

printf("you wrote %s\n",(char\*)shared\_memory);

printf("KEY OF SHARED MEMORY is %d\n",shmid);

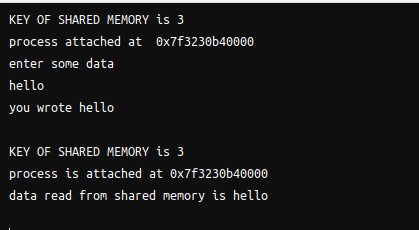
printf("process is attached at %p\n",shared\_memory);

printf("data read from shared memory is %s\n",(char\*)shared\_memory);

return 0;

}

**OUTPUT:-**



**PROGRAM 3(A).**

write a c program to simulate CPU SCHEDULING ALGORITHM .FCFS calculating waiting time

#include <stdio.h>

int main() {

int n, bt[10], wt[10], tat[10], avwt = 0, avtat = 0, i, j;

// Input number of processes

printf("Enter total no. of processes: ");

scanf("%d", &n);

// Input burst time for each process

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

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

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

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

}

wt[0] = 0; // Waiting time for the first process is 0

// Calculate waiting time for each process

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

wt[i] = 0;

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

wt[i] += bt[j]; // Accumulate burst times of previous processes

}

}

// Print header for the table

printf("\nProcess\tBURST TIME\tWAIT.T\tTAT\n");

// Calculate turnaround time and print results

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

tat[i] = bt[i] + wt[i]; // Turnaround time = burst time + waiting time

avwt += wt[i]; // Accumulate waiting time

avtat += tat[i]; // Accumulate turnaround time

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

}

// Calculate average waiting time and turnaround time

avwt=avwt/i;

avtat=avtat/i;

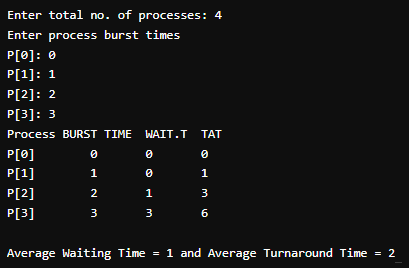
// Print average waiting time and turnaround time

printf("\nAverage Waiting Time = %d and Average Turnaround Time = %d", avwt,avtat);

return 0;

}

**OUTPUT:-**



**PROGRAM 6**

write a c program to simulate BANKER’S ALGORITHM to avoid deadlock

#include <stdio.h>

#define true 1

#define false 0

int available[10], allocation[10][10], max[10][10], need[10][10], work[10], finish[10], maxres[10], safe[10], req[10], m, n;

int find()

{

    int i, j;

    for (i = 0; i < n; i++)

    {

        if (finish[i] == false)

        {

            for (j = 0; j < m; j++)

            {

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

                    break;

            }

            if (j == m)

            {

                finish[i] = true;

                return i;

            }

        }

    }

    return -1;

}

int issafe()

{

    int i, j, k = 0, cnt = n;

    for (j = 0; j < m; j++)

    {

        work[j] = available[j];

    }

    for (i = 0; i < n; i++)

    {

        finish[i] = false;

    }

    while (cnt > 0)

    {

        i = find();

        if (i == -1)

        {

            printf("\nThe system is in an unsafe state\n");

            return 0;

        }

        // Release resources and update work[] array

        for (j = 0; j < m; j++)

        {

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

        }

        safe[k++] = i;

        cnt--;

    }

    printf("\nThe system is in a safe state, safe sequence: ");

    for (i = 0; i < n; i++)

    {

        printf("P%d, ", safe[i]);

    }

    printf("\n");

    return 1;

}

int main()

{

    int i, j, sum;

    printf("\nEnter the number of processes and the number of resources:\n");

    scanf("%d%d", &n, &m);

    printf("Enter maximum instances of resources:\n");

    for (j = 0; j < m; j++)

    {

        scanf("%d", &maxres[j]);

        available[j] = maxres[j];

    }

    printf("\nEnter the Allocated Matrix:\n");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < m; j++)

        {

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

        }

    }

    printf("\nEnter the Max Matrix:\n");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < m; j++)

        {

            scanf("%d", &max[i][j]);

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

        }

    }

    printf("\nThe Need Matrix is:\n");

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < m; j++)

        {

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

        }

        printf("\n");

    }

    // Calculate available resources by subtracting allocated resources from the maximum resources

    for (j = 0; j < m; j++)

    {

        sum = 0;

        for (i = 0; i < n; i++)

        {

            sum += allocation[i][j];

        }

        available[j] -= sum;

    }

    // Call the safety algorithm

    issafe();

    return 0;

}

**OUTPUT:-**

