**Week 9a(factorial)**

echo enter a number

read a

i=2

fact=1

if [ $a -ge 2 ]

then

while [ $i -le $a ]

do

fact=`expr $fact \\* $i`

i=`expr $i + 1`

done

fi

echo factorial of $a=$fact

output:

[latha@localhost ~]$ sh fact.sh enter a number 5 factorial of 5=120

**Week9b**

NAME OF THE EXPERIMENT:Creating a child process and allow the parent to display ‘parent’.

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<string.h>

int global = 10;

int main()

{

int local=20;

pid\_t pid;

printf("before fork\n");

printf("pid= %d,global= %d,local= %d\n",getpid(),global,local);

pid=fork();

if(pid<0)

printf("failed to create child");

else if(pid==0)

{

printf("after fork\n");

global++;

local++;

}

else if(pid>0)

sleep(2);

printf("cid=%d,global= %d,local= %d\n",getpid(),global,local);

exit(0);

}

**Weeek9c:** AIM:Write a C program in which a parent writes a message to a pipe and the child reads themessage.

SOURCE CODE:

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

Int main(void) {

int pid;

int status;

printf("Hello World!\n");

pid = fork( );

if(pid == -1)

{

perror("bad fork");

exit(1);

} i

f (pid == 0)

printf("I am the child process.\n");

else {

wait(&status);

printf("I am the parent process.\n");

return 0; } }

**OUTPUT:** student@202.sys14:~$ ./a.out

Hello World!

I am the child process

I am the parent process.

**#10a(fcfs)**

#include<stdio.h>

void main()

{

int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i;

float awt,atat;

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

scanf("%d",&n);

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

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

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

wt[0]=0;

tat[0]=bt[0];

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

wt[i]=tat[i-1];

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

}

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

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID \t BT \t WT \t TAT");

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

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

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

}

**#10b(Priority)**

#include<stdio.h>

void main()

{

int pid[10],bt[10],pr[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

float awt,atat;

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

scanf("%d",&n);

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

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

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

printf("\n Enter PID:");

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

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

printf("\n Enter Priorities:");

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

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

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

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

if(pr[i]>pr[j]){

t=pr[i];

pr[i]=pr[j];

pr[j]=t;

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

t=pid[i];

pid[i]=pid[j];

pid[j]=t;

}}}

wt[0]=0;

tat[0]=bt[0];

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

wt[i]=tat[i-1];

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

}

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

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID PRIORITY \t BT \t WT \t TAT");

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

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

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

}

**Week11a:**

#include<stdio.h>

void main(){

int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

float awt,atat;

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

scanf("%d",&n);

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

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

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

printf("\n Enter PID:");

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

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

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

{

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

{

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

{

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

t=pid[i];

pid[i]=pid[j];

pid[j]=t;

}}}

wt[0]=0;

tat[0]=bt[0];

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

{

wt[i]=tat[i-1];

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

}

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

{

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID \t BT \t WT \t TAT");

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

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

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

}

**#11b(RR)**

#include<stdio.h>

void main()

{

int ts,bt1[10],wt[10],tat[10],i,j=0,n,bt[10],ttat=0,twt=0,tot=0;

float awt,atat;

printf("Enter the number of Processes \n");

scanf("%d",&n);

printf("\n Enter the Timeslice \n");

scanf("%d",&ts);

printf("\n Enter the Burst Time for the process");

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

scanf("%d",&bt1[i]);

bt[i]=bt1[i];

}

while(j<n){

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

if(bt[i]>0){

if(bt[i]>=ts){

tot+=ts;

bt[i]=bt[i]-ts;

if(bt[i]==0){

j++;

tat[i]=tot;

}}

else{

tot+=bt[i];

bt[i]=0;

j++;

tat[i]=tot;

}}}}

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

wt[i]=tat[i]-bt1[i];

twt=twt+wt[i];

ttat=ttat+tat[i];

}

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\n PID \t BT \t WT \t TAT\n");

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

printf("\n %d \t %d \t %d \t %d \t\n",i,bt1[i],wt[i],tat[i]);

}

printf("\n The average Waiting Time=%f",awt);

printf("\n The average Turn around Time=%f",atat);

}

**#12a(sequential)**

#include<stdio.h>

int main(){

int i,j,n,b[20],sb[20],t[20],x,c[20][20];

printf("enter no.of files:");

scanf("%d",&n);

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

printf("enter no.of blocks occupied by file%d:",i+1);

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

printf("enter no.of starting blocks of file%d:",i+1);

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

t[i]=sb[i];

for(j=0;j<b[i];j++)

c[i][j]=sb[i]++;

}

printf("filename\tstartblock\tlength\n");

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

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

printf("\nenter file number to display:");

scanf("%d",&x);

if(x>0 && x<=n){

printf("\n filename is %d",x);

printf("\n length is %d",b[x-1]);

printf("\n blocks occupied:");

for(i=0;i<b[x-1];i++)

printf("%d",c[x-1][i]);

printf("\n");

}else{

printf("Invalid file number!\n");

}

return 0;

}

**#13a(paging)**

#include<stdio.h>

void main()

{

int i,j,temp,framearr[20],pages,pageno,frames,memsize,log,pagesize,prosize,base;

printf("Enter the process size: ");

scanf("%d",&prosize);

printf("\nEnter the main memory size:");

scanf("%d",&memsize);

printf("\nEnter the page size:");

scanf("%d",&pagesize);

pages=prosize/pagesize;

printf("\nThe process is divided into %d pages",pages);

frames = memsize/pagesize;

printf("\nThe main memory divided into %d frames\n",frames);

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

framearr[i]=-I;

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

pos: printf("\nEnter the frame number of page%d:",i);

scanf("%d",&temp);

if(temp>=frames)

{

printf("\n\t\*Invalid frame number\*\n");

goto pos;

}

/\* storing pageno (i.e temp ) index \*/

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

if(temp==j)

framearr[temp]=i;

j = log%pagesize; /\* here 'j' is dispalcement\*/

temp = base + (temp\*pagesize)+j;

printf("\nphysical address is : %d" ,temp);

}

}

**#14a(FIFO)**

#include<stdio.h>

void main()

{

int i,j,n,a[50],frame[10],fno,k,avail,pagefault=0;

printf("\nEnter the number of Frames : ");

scanf("%d",&fno);

printf("\nEnter number of reference string :");

scanf("%d",&n);

printf("\n Enter the Reference string :\n");

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

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

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

frame[i]= -1;

j=0;

printf("\n FIFO Page Replacement Algorithm\n\n The given reference string is:\n\n");

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

{

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

}

printf("\n");

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

{

printf("\nReference No %d-> ",a[i]);

avail=0;

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

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j = (j+1) % fno;

pagefault++;

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

if(frame[k]!=-1)

printf(" %2d",frame[k]);

}

printf("\n");

}

printf("\nPage Fault Is %d",pagefault);

}

**CN LAB:**

**Week1:**

**character stuffing:**

#include <stdio.h>

#include <string.h>

int main() {

char a[100], c[200]; // Input and stuffed strings

int i = 0, k = 0;

printf("Enter the string: ");

scanf("%s", a);

// Character stuffing logic

for (i = 0; a[i] != '\0'; i++)

{

if (a[i] == 'D' && a[i+1] == 'L' && a[i+2] == 'E')

{

// Check for DLE sequence

c[k++] = 'D';

c[k++] = 'L';

c[k++] = 'E';

c[k++] = 'D';

i += 2;

} else {

c[k++] = a[i];

}

}

c[k] = '\0'; // Null-terminate the stuffed string

// Printing the results

printf("DLESTX%sDLEETX\n", c); // Add protocol framing

return 0;

}

Output:







BIT :

**bit stuffing:**

#include <stdio.h>

#include <string.h>

void bitStuffing(char input[], char output[]) {

int count = 0;

int j = 0;

for (int i = 0; input[i] != '\0'; i++)

{

output[j++] = input[i];

if (input[i] == '1') {

count++;

} else {

count = 0; // Reset count if 0 is encountered

}

// If 5 consecutive 1's are found, stuff a 0

if (count == 5) {

output[j++] = '0';

count = 0; // Reset count after stuffing

}

}

output[j] = '\0'; // Null-terminate the output string

}

int main() {

char input[100], output[200];

printf("Enter the input bit stream: ");

scanf("%s", input);

bitStuffing(input, output);

printf("Bit-stuffed output: %s\n", output);

return 0;

}

Output:







Week2 :

CRC:

#include <stdio.h>

#include <string.h>

int main() {

int i, j, keylen, msglen;

char input[100], key[30], temp[30], quot[100], key1[30], rem[30];

printf("Enter Data: ");

scanf("%s", input);

printf("Enter Key: ");

scanf("%s", key);

keylen = strlen(key);

msglen = strlen(input);

strcpy(key1, key);

for (i = 0; i < keylen - 1; i++) {

input[msglen + i] = '0';

}

input[msglen + keylen - 1] = '\0';

strncpy(temp, input, keylen);

temp[keylen] = '\0';

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

quot[i] = temp[0];

if (quot[i] == '0') {

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

key[j] = '0';

}

} else {

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

key[j] = key1[j];

}

}

for (j = 1; j < keylen; j++) {

temp[j - 1] = (temp[j] == key[j]) ? '0' : '1';

}

if (i + keylen < msglen + keylen - 1) {

temp[keylen - 1] = input[i + keylen];

}

}

strncpy(rem, temp, keylen - 1);

rem[keylen - 1] = '\0';

quot[msglen] = '\0';

printf("Quotient : %s\n", quot);

printf("Remainder : %s\n", rem);

return 0;

}

Week 3:

Stop and wait protocol:

Implement Stop and Wait Protocol.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> // For sleep function

#include <time.h> // For random number generation

void sender();

void receiver(int frame);

int main() {

srand(time(0)); // Seed for random number generation

sender();

return 0;

}

void send3er() {

int frame = 0; // Initial frame number

char ack;

int timeout;

while (frame < 5) { // Simulating 5 frames to be sent

printf("Sender: Sending frame %d\n", frame);

// Simulating packet loss with a 30% probability

if (rand() % 10 < 3) {

printf("Sender: Frame %d lost! Resending...\n\n", frame);

sleep(1);

continue;

}

receiver(frame); // Call receiver function

// Simulating waiting for acknowledgment with timeout

timeout = rand() % 10; // Random timeout event (0-9)

printf("Sender: Waiting for acknowledgment...\n");

sleep(1);

if (timeout < 2) { // Simulating timeout with 20% probability

printf("Sender: Timeout occurred! Resending frame %d\n\n", frame);

continue;

}

printf("Receiver: Enter 'y' to acknowledge frame %d: ", frame);

scanf(" %c", &ack);

if (ack == 'y' || ack == 'Y') {

printf("Sender: Acknowledgment received for frame %d\n\n", frame);

frame++; // Send next frame

} else {

printf("Sender: Acknowledgment not received. Resending frame %d\n\n", frame);

}

}

}

void receiver(int frame) {

printf("Receiver: Frame %d received successfully.\n", frame);

}

Week 4:

Sliding window protocol:

Implement Sliding Window Protocol.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> // For sleep function

#define WINDOW\_SIZE 4 // Sliding window size

#define TOTAL\_FRAMES 10 // Total frames to send

void sender();

void receiver(int frame);

int main() {

sender();

return 0;

}

void sender() {

int base = 0, next\_frame = 0;

char ack;

while (base < TOTAL\_FRAMES) {

// Sending frames within the window size

for (int i = 0; i < WINDOW\_SIZE && (next\_frame < TOTAL\_FRAMES); i++) {

printf("Sender: Sending frame %d\n", next\_frame);

receiver(next\_frame);

next\_frame++;

}

// Simulating waiting for acknowledgment

for (int i = base; i < next\_frame; i++) {

printf("Receiver: Enter 'y' to acknowledge frame %d (or 'n' to reject): ", i);

scanf(" %c", &ack);

if (ack == 'y' || ack == 'Y') {

printf("Sender: Acknowledgment received for frame %d\n", i);

base++; // Move window forward

} else {

printf("Sender: Acknowledgment not received for frame %d. Resending from frame %d\n", i, base);

next\_frame = base; // Reset next frame to resend from base

break; // Exit the acknowledgment loop and start resending

}

}

printf("\n");

}

}

void receiver(int frame) {

printf("Receiver: Frame %d received successfully.\n", frame);

}

Week 5:

Shortes path:

**Experiment 5:**

**Implement Dijkstra's shortest path algorithm through a graph.**

#include <stdio.h>

#define INFINITY 9999

#define MAX 20

// Function to find the minimum of two values

int minimum(int a, int b) {

return (a <= b) ? a : b;

}

int main() {

int i, j, k, n, start, end, adj[MAX][MAX], path[MAX][MAX];

// Input number of vertices

printf("Enter number of vertices: ");

scanf("%d", &n);

// Input adjacency matrix

printf("Enter weighted matrix (0 for no path):\n");

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

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

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

// Initialize path matrix

path[i][j] = (i != j && adj[i][j] == 0) ? INFINITY : adj[i][j];

}

}

// Floyd-Warshall algorithm

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

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

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

if (path[i][k] != INFINITY && path[k][j] != INFINITY) {

path[i][j] = minimum(path[i][j], path[i][k] + path[k][j]);

}

}

}

}

// Print shortest path matrix

printf("\nShortest path matrix:\n");

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

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

if (path[i][j] == INFINITY)

printf("%6s", "INF");

else

printf("%6d", path[i][j]);

}

printf("\n");

}

// Query shortest path between two vertices

printf("\nEnter start vertex: ");

scanf("%d", &start);

printf("Enter end vertex: ");

scanf("%d", &end);

if (path[start][end] == INFINITY)

printf("No path exists between %d and %d\n", start, end);

else

printf("The minimum cost between %d and %d is %d\n", start, end, path[start][end]);

return 0;

}

**Output:**

Enter number of vertices: 4

Enter weighted matrix (0 for no path):

0 3 0 7

3 0 1 5

0 1 0 2

7 5 2 0

Shortest path matrix:

0 3 4 6

3 0 1 3

4 1 0 2

6 3 2 0

Enter start vertex: 0

Enter end vertex: 3

The minimum cost between 0 and 3 is 6

Week 6:

#include <stdio.h>

int main() {

int a[10][10], n;

int i, j, root;

printf("Enter no. of nodes: ");

scanf("%d", &n);

printf("Enter adjacency matrix\n");

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

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

printf("Enter connection of %d --> %d: ", i, j);

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

}

}

printf("Enter root node (1 to %d): ", n);

scanf("%d", &root);

printf("\nAdjacent nodes of root node %d:\n", root);

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

if (a[root][i] == 1 || a[i][root] == 1) {

printf("%d\t", i);

}

}

printf("\n");

return 0;

}

Week 7:

**Experiment 7: Implementation of bit map protocol**

#include <stdio.h>

#define MAX\_BLOCKS 32

void displayBitmap(int bitmap[], int n) {

printf("Current Bitmap: ");

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

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

}

printf("\n");

}

int allocateBlock(int bitmap[], int n) {

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

if (bitmap[i] == 0) {

bitmap[i] = 1;

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

return i;

}

}

printf("No Free Blocks Available.\n");

return -1;

}

void deallocateBlock(int bitmap[], int block) {

if (bitmap[block] == 1) {

bitmap[block] = 0;

printf("Block %d deallocated.\n", block);

} else {

printf("Block %d is already free.\n", block);

}

}

int main() {

int bitmap[MAX\_BLOCKS] = {0}; // Initialize all blocks as free

int choice, block;

// Display the menu only once at the beginning

printf("\*\*\*\*\*\*\*\*\*\* Bit Map Protocol \*\*\*\*\*\*\*\*\*\*\n");

printf("Menu:\n");

printf("1. Allocate Block\n");

printf("2. Deallocate Block\n");

printf("3. Display Bitmap\n");

printf("4. Show Menu Again\n");

printf("5. Exit\n");

while (1) {

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

allocateBlock(bitmap, MAX\_BLOCKS);

break;

case 2:

printf("Enter block number to deallocate (0 - %d): ", MAX\_BLOCKS - 1);

scanf("%d", &block);

if (block >= 0 && block < MAX\_BLOCKS)

deallocateBlock(bitmap, block);

else

printf("Invalid block number.\n");

break;

case 3:

displayBitmap(bitmap, MAX\_BLOCKS);

break;

case 4:

// Option to display menu again if user wants to see it

printf("\nMenu:\n");

printf("1. Allocate Block\n");

printf("2. Deallocate Block\n");

printf("3. Display Bitmap\n");

printf("4. Show Menu Again\n");

printf("5. Exit\n");

break;

case 5:

printf("Exiting...\n");

return 0;

default:

printf("Invalid choice. Try again.\n");

}

}

return 0;

}

**Sample Output:**

\*\*\*\*\*\*\*\*\*\* Bit Map Protocol \*\*\*\*\*\*\*\*\*\*

Menu:

1. Allocate Block

2. Deallocate Block

3. Display Bitmap

4. Show Menu Again

5. Exit

Enter your choice: 3

Current Bitmap: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Enter your choice: 1

Allocated Block: 0

Enter your choice: 3

Current Bitmap: 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Enter your choice: 2

Enter block number to deallocate (0 - 31): 0

Block 0 deallocated.

Enter your choice: 4

Menu:

1. Allocate Block

2. Deallocate Block

3. Display Bitmap

4. Show Menu Again

5. Exit

Enter your choice: 5

Exiting...