1 - Student Database

#!/bin/bash

DB\_FILE="student\_db.txt"

# Function to create database (just clears or creates the file)

create\_database() {

> "$DB\_FILE"

echo "Database created (or cleared)."

}

# Function to view database

view\_database() {

if [[ -s "$DB\_FILE" ]]; then

echo -e "\n----- Student Database -----"

column -t -s "," "$DB\_FILE"

else

echo "Database is empty."

fi

}

# Function to insert a record

insert\_record() {

echo "Enter Roll Number:"

read roll

echo "Enter Name:"

read name

echo "Enter Marks (out of 100):"

read marks

echo "$roll,$name,$marks" >> "$DB\_FILE"

echo "Record inserted."

}

# Function to delete a record

delete\_record() {

echo "Enter Roll Number to delete:"

read roll

grep -v "^$roll," "$DB\_FILE" > temp && mv temp "$DB\_FILE"

echo "Record deleted (if it existed)."

}

# Function to modify a record

modify\_record() {

echo "Enter Roll Number to modify:"

read roll

grep -v "^$roll," "$DB\_FILE" > temp && mv temp "$DB\_FILE"

echo "Enter new Name:"

read name

echo "Enter new Marks (out of 100):"

read marks

echo "$roll,$name,$marks" >> "$DB\_FILE"

echo "Record modified."

}

# Function to display result of a particular student

student\_result() {

echo "Enter Roll Number:"

read roll

record=$(grep "^$roll," "$DB\_FILE")

if [[ -n "$record" ]]; then

IFS=',' read -r r name marks <<< "$record"

echo "Roll Number: $r"

echo "Name: $name"

echo "Marks: $marks"

if (( marks >= 40 )); then

echo "Result: PASS"

else

echo "Result: FAIL"

fi

else

echo "No record found for Roll Number $roll."

fi

}

# Main menu loop

while true; do

echo -e "\n---- Student Database Menu ----"

echo "a) Create Database"

echo "b) View Database"

echo "c) Insert a Record"

echo "d) Delete a Record"

echo "e) Modify a Record"

echo "f) Result of a Particular Student"

echo "g) Exit"

echo -n "Enter your choice: "

read choice

case $choice in

a) create\_database ;;

b) view\_database ;;

c) insert\_record ;;

d) delete\_record ;;

e) modify\_record ;;

f) student\_result ;;

g) echo "Exiting..."; break ;;

\*) echo "Invalid option. Try again." ;;

esac

done

#!/bin/bash

factorial() {

echo -n "Enter a number: "

read n

fact=1

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

fact=$((fact \* i))

done

echo "Factorial of $n is $fact"

}

greatest\_of\_three() {

echo "Enter three numbers:"

read a b c

if (( a >= b && a >= c )); then

echo "$a is greatest"

elif (( b >= c )); then

echo "$b is greatest"

else

echo "$c is greatest"

fi

}

is\_prime() {

echo -n "Enter a number: "

read n

if (( n <= 1 )); then

echo "$n is not a prime number"

return

fi

for (( i=2; i\*i<=n; i++ )); do

if (( n % i == 0 )); then

echo "$n is not a prime number"

return

fi

done

echo "$n is a prime number"

}

number\_palindrome() {

echo -n "Enter a number: "

read num

rev=$(echo $num | rev)

if [[ "$num" == "$rev" ]]; then

echo "$num is a palindrome"

else

echo "$num is not a palindrome"

fi

}

string\_palindrome() {

echo -n "Enter a string: "

read str

rev=$(echo "$str" | rev)

if [[ "$str" == "$rev" ]]; then

echo "\"$str\" is a palindrome"

else

echo "\"$str\" is not a palindrome"

fi

}

while true; do

echo -e "\n---- MENU ----"

echo "a) Find factorial of a number"

echo "b) Find greatest of three numbers"

echo "c) Check prime number"

echo "d) Check if a number is a palindrome"

echo "e) Check if a string is a palindrome"

echo "f) Exit"

echo -n "Enter your choice: "

read choice

case $choice in

a) factorial ;;

b) greatest\_of\_three ;;

c) is\_prime ;;

d) number\_palindrome ;;

e) string\_palindrome ;;

f) echo "Exiting..."; break ;;

\*) echo "Invalid option" ;;

esac

done

2 - Fork sorting

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

void sort\_desc(int arr[], int n) {

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

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

if (arr[i] < arr[j]) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

void print\_array(const char\* label, int arr[], int n) {

printf("%s: ", label);

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

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

printf("\n");

}

int main() {

int arr[] = {5, 2, 8, 4, 1};

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

pid\_t pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

} else if (pid == 0) {

// Child process

sort\_desc(arr, n);

print\_array("Child sorted array", arr, n);

} else {

wait(NULL); // Wait for child to complete

sort\_desc(arr, n);

print\_array("Parent sorted array", arr, n);

}

return 0;

}

3 - vfork()

#include <stdio.h>

#include <unistd.h>

#include <string.h>

#include <ctype.h>

int main() {

char sentence[100];

int \*vowel\_count;

printf("Enter a sentence: ");

fgets(sentence, sizeof(sentence), stdin);

int count = 0;

pid\_t pid = vfork(); // Use vfork() instead of fork()

if (pid < 0) {

perror("vfork failed");

return 1;

} else if (pid == 0) {

// Child process

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

char ch = tolower(sentence[i]);

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

count++;

}

}

printf("Child: Number of vowels = %d\n", count);

\_exit(0); // Must use \_exit() with vfork

} else {

wait(NULL);

printf("Parent: Child completed vowel counting.\n");

}

return 0;

}

4 - FCFS

#include <stdio.h>

int main() {

int n, bt[20], wt[20], tat[20], i;

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

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

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

printf("P[%d]: ", i + 1);

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

}

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

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tBT\tWT\tTAT\n");

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

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

}

printf("Average Waiting Time: %.2f\n", avg\_wt/n);

printf("Average Turnaround Time: %.2f\n", avg\_tat/n);

return 0;

}

5 - SJF - Preemptive

#include <stdio.h>

int main() {

int n, i, j, smallest, time = 0, count = 0;

int at[10], bt[10], rt[10], wt[10], tat[10];

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter arrival time and burst time for each process:\n");

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

printf("P[%d] Arrival Time: ", i+1);

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

printf("P[%d] Burst Time: ", i+1);

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

rt[i] = bt[i]; // Remaining time

}

int complete = 0;

int minm = 9999;

int finish\_time;

int shortest = 0;

int check = 0;

while (complete != n) {

shortest = -1;

minm = 9999;

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

if (at[j] <= time && rt[j] > 0 && rt[j] < minm) {

minm = rt[j];

shortest = j;

check = 1;

}

}

if (check == 0) {

time++;

continue;

}

rt[shortest]--;

if (rt[shortest] == 0) {

complete++;

finish\_time = time + 1;

wt[shortest] = finish\_time - bt[shortest] - at[shortest];

if (wt[shortest] < 0)

wt[shortest] = 0;

}

time++;

}

// Turnaround time

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

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

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tAT\tBT\tWT\tTAT\n");

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

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

}

printf("Average Waiting Time: %.2f\n", avg\_wt / n);

printf("Average Turnaround Time: %.2f\n", avg\_tat / n);

return 0;

}

6 - Round Robin - Preemptive

#include <stdio.h>

int main() {

int n, i, qt, count = 0, temp, sq = 0;

int bt[10], rt[10], wt[10] = {0}, tat[10] = {0};

printf("Enter number of processes: ");

scanf("%d", &n);

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

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

printf("P[%d]: ", i+1);

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

rt[i] = bt[i];

}

printf("Enter time quantum: ");

scanf("%d", &qt);

while(1) {

int done = 1;

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

if(rt[i] > 0) {

done = 0;

if(rt[i] > qt) {

sq += qt;

rt[i] -= qt;

} else {

sq += rt[i];

wt[i] = sq - bt[i];

rt[i] = 0;

}

}

}

if(done) break;

}

float avg\_wt = 0, avg\_tat = 0;

printf("\nProcess\tBT\tWT\tTAT\n");

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

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

avg\_wt += wt[i];

avg\_tat += tat[i];

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

}

printf("Average Waiting Time: %.2f\n", avg\_wt/n);

printf("Average Turnaround Time: %.2f\n", avg\_tat/n);

return 0;

}

7 - Priority Scheduling - Non Preemptive

#include <stdio.h>

int main() {

int n, bt[20], p[20], pr[20], wt[20], tat[20], i, j, temp;

float avg\_wt = 0, avg\_tat = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

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

printf("Enter burst time and priority for P[%d]: ", i+1);

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

p[i] = i+1;

}

// Sort by priority (lower number = higher priority)

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

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

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

temp = pr[i]; pr[i] = pr[j]; pr[j] = temp;

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

temp = p[i]; p[i] = p[j]; p[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] = bt[i] + wt[i];

avg\_wt += wt[i];

avg\_tat += tat[i];

}

printf("\nProcess\tBT\tPriority\tWT\tTAT\n");

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

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

}

printf("Average Waiting Time: %.2f\n", avg\_wt/n);

printf("Average Turnaround Time: %.2f\n", avg\_tat/n);

return 0;

}

8 - Bankers algorithm

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int main() {

int n, m; // n = number of processes, m = number of resources

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

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

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

int avail[MAX\_RESOURCES];

int i, j, k;

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter number of resources: ");

scanf("%d", &m);

printf("Enter Allocation Matrix:\n");

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

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

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

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

}

}

printf("Enter Maximum Requirement Matrix:\n");

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

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

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

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

need[i][j] = max[i][j] - alloc[i][j]; // Calculate need matrix

}

}

printf("Enter Available Resources: ");

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

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

}

bool finish[MAX\_PROCESSES] = {false};

int safeSeq[MAX\_PROCESSES];

int count = 0;

while(count < n) {

bool found = false;

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

if(!finish[i]) {

bool canAllocate = true;

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

if(need[i][j] > avail[j]) {

canAllocate = false;

break;

}

}

if(canAllocate) {

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

avail[k] += alloc[i][k]; // Release resources

}

safeSeq[count++] = i;

finish[i] = true;

found = true;

}

}

}

if(!found) {

printf("\nSystem is NOT in a safe state.\n");

return 1;

}

}

// If here, system is in a safe state

printf("\nSystem is in a SAFE state.\nSafe Sequence: ");

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

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

if(i != n-1)

printf(" -> ");

}

printf("\n");

return 0;

}

6 - Page Replacement Algorithms

FIFO

#include <stdio.h>

int main() {

int frames, pages, i, j, k, page\_faults = 0;

int ref[100], mem[10], pos = 0, flag;

printf("Enter number of pages: ");

scanf("%d", &pages);

printf("Enter reference string: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

for(i = 0; i < frames; i++) mem[i] = -1;

printf("\nFIFO Page Replacement:\n");

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

flag = 0;

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

if(mem[j] == ref[i]) {

flag = 1;

break;

}

}

if(flag == 0) {

mem[pos] = ref[i];

pos = (pos + 1) % frames;

page\_faults++;

}

printf("Frame content: ");

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

printf("%d ", mem[k]);

printf("\n");

}

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

return 0;

}

LRU

#include <stdio.h>

int findLRU(int time[], int n) {

int i, min = time[0], pos = 0;

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

if(time[i] < min) {

min = time[i];

pos = i;

}

}

return pos;

}

int main() {

int frames, pages, i, j, k, time[10], counter = 0, page\_faults = 0, pos;

int mem[10], ref[100], flag;

printf("Enter number of pages: ");

scanf("%d", &pages);

printf("Enter reference string: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

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

mem[i] = -1;

time[i] = 0;

}

printf("\nLRU Page Replacement:\n");

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

flag = 0;

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

if(mem[j] == ref[i]) {

flag = 1;

time[j] = ++counter;

break;

}

}

if(flag == 0) {

if(i < frames) {

mem[i] = ref[i];

time[i] = ++counter;

} else {

pos = findLRU(time, frames);

mem[pos] = ref[i];

time[pos] = ++counter;

}

page\_faults++;

}

printf("Frame content: ");

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

printf("%d ", mem[k]);

printf("\n");

}

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

return 0;

}

Optimal

#include <stdio.h>

int search(int key, int frame[], int frames) {

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

if(frame[i] == key)

return 1;

return 0;

}

int predict(int ref[], int frame[], int index, int n, int frames) {

int res = -1, farthest = index;

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

int j;

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

if(frame[i] == ref[j]) {

if(j > farthest) {

farthest = j;

res = i;

}

break;

}

}

if(j == n)

return i;

}

return (res == -1) ? 0 : res;

}

int main() {

int n, frames;

int ref[100], frame[10], page\_faults = 0;

printf("Enter number of pages: ");

scanf("%d", &n);

printf("Enter reference string: ");

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

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

printf("Enter number of frames: ");

scanf("%d", &frames);

int filled = 0;

printf("\nOptimal Page Replacement:\n");

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

if(search(ref[i], frame, filled)) {

// hit

} else {

if(filled < frames)

frame[filled++] = ref[i];

else {

int pos = predict(ref, frame, i + 1, n, frames);

frame[pos] = ref[i];

}

page\_faults++;

}

printf("Frame content: ");

for(int j = 0; j < filled; j++)

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

printf("\n");

}

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

return 0;

}

File Handling System Calls

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#include <stdlib.h>

struct Student {

int roll;

char name[20];

float marks;

};

#define FILENAME "student\_db.dat"

void createDB() {

int fd = open(FILENAME, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (fd < 0) {

perror("Cannot create file");

return;

}

int n;

printf("Enter number of students: ");

scanf("%d", &n);

struct Student s;

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

printf("Enter roll, name, marks: ");

scanf("%d %s %f", &s.roll, s.name, &s.marks);

write(fd, &s, sizeof(s));

}

close(fd);

}

void viewDB() {

int fd = open(FILENAME, O\_RDONLY);

if (fd < 0) {

perror("Cannot open file");

return;

}

struct Student s;

printf("\n%-5s %-20s %-5s\n", "Roll", "Name", "Marks");

while (read(fd, &s, sizeof(s)) > 0) {

printf("%-5d %-20s %-5.2f\n", s.roll, s.name, s.marks);

}

close(fd);

}

void insertRecord() {

int fd = open(FILENAME, O\_WRONLY | O\_APPEND);

if (fd < 0) {

perror("Cannot open file");

return;

}

struct Student s;

printf("Enter roll, name, marks: ");

scanf("%d %s %f", &s.roll, s.name, &s.marks);

write(fd, &s, sizeof(s));

close(fd);

}

void deleteRecord() {

int fd = open(FILENAME, O\_RDONLY);

if (fd < 0) {

perror("Cannot open file");

return;

}

int tempfd = open("temp.dat", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (tempfd < 0) {

perror("Cannot create temp file");

close(fd);

return;

}

int roll;

printf("Enter roll number to delete: ");

scanf("%d", &roll);

struct Student s;

while (read(fd, &s, sizeof(s)) > 0) {

if (s.roll != roll) {

write(tempfd, &s, sizeof(s));

}

}

close(fd);

close(tempfd);

remove(FILENAME);

rename("temp.dat", FILENAME);

}

void modifyRecord() {

int fd = open(FILENAME, O\_RDWR);

if (fd < 0) {

perror("Cannot open file");

return;

}

int roll;

printf("Enter roll number to modify: ");

scanf("%d", &roll);

struct Student s;

while (read(fd, &s, sizeof(s)) > 0) {

if (s.roll == roll) {

printf("Enter new name and marks: ");

scanf("%s %f", s.name, &s.marks);

lseek(fd, -sizeof(s), SEEK\_CUR);

write(fd, &s, sizeof(s));

printf("Record updated.\n");

close(fd);

return;

}

}

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

close(fd);

}

void searchResult() {

int fd = open(FILENAME, O\_RDONLY);

if (fd < 0) {

perror("Cannot open file");

return;

}

int roll;

printf("Enter roll number to search: ");

scanf("%d", &roll);

struct Student s;

while (read(fd, &s, sizeof(s)) > 0) {

if (s.roll == roll) {

printf("Name: %s, Marks: %.2f\n", s.name, s.marks);

close(fd);

return;

}

}

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

close(fd);

}

int main() {

int choice;

do {

printf("\n--- Student Database Menu ---\n");

printf("1. Create Database\n2. View Database\n3. Insert Record\n4. Delete Record\n5. Modify Record\n6. Student Result\n7. Exit\n");

printf("Enter choice: ");

scanf("%d", &choice);

switch(choice) {

case 1: createDB(); break;

case 2: viewDB(); break;

case 3: insertRecord(); break;

case 4: deleteRecord(); break;

case 5: modifyRecord(); break;

case 6: searchResult(); break;

case 7: printf("Exiting.\n"); break;

default: printf("Invalid choice.\n");

}

} while(choice != 7);

return 0;

}

IPC using Pipe

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

int main() {

int fd[2];

pid\_t pid;

char buffer[100];

// Create pipe

if (pipe(fd) == -1) {

perror("Pipe failed");

exit(1);

}

// Create child process

pid = fork();

if (pid < 0) {

perror("Fork failed");

exit(1);

}

else if (pid == 0) {

// Consumer Process

close(fd[1]); // Close write end

printf("Consumer: Waiting for data...\n");

read(fd[0], buffer, sizeof(buffer));

printf("Consumer: Received -> %s\n", buffer);

close(fd[0]); // Close read end

exit(0);

}

else {

// Producer Process

close(fd[0]); // Close read end

char data[100];

printf("Producer: Enter data to send: ");

fgets(data, sizeof(data), stdin);

write(fd[1], data, strlen(data) + 1);

printf("Producer: Data sent to consumer.\n");

close(fd[1]); // Close write end

wait(NULL); // Wait for child to finish

}

return 0;

}

Multithreading

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define MAX 10

int A[MAX][MAX], B[MAX][MAX];

int rowA, colA, rowB, colB;

typedef struct {

int row;

int col;

} Position;

void \*multiply(void \*arg) {

Position \*pos = (Position \*)arg;

int i, sum = 0;

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

sum += A[pos->row][i] \* B[i][pos->col];

}

int \*result = malloc(sizeof(int));

\*result = sum;

pthread\_exit(result);

}

int main() {

printf("Enter rows and columns of Matrix A: ");

scanf("%d %d", &rowA, &colA);

printf("Enter rows and columns of Matrix B: ");

scanf("%d %d", &rowB, &colB);

if (colA != rowB) {

printf("Matrix multiplication not possible!\n");

return -1;

}

printf("Enter elements of Matrix A:\n");

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

for (int j = 0; j < colA; j++)

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

printf("Enter elements of Matrix B:\n");

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

for (int j = 0; j < colB; j++)

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

pthread\_t threads[MAX \* MAX];

Position positions[MAX \* MAX];

int result[MAX][MAX];

int thread\_count = 0;

// Create threads for each element in result matrix

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

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

positions[thread\_count].row = i;

positions[thread\_count].col = j;

pthread\_create(&threads[thread\_count], NULL, multiply, &positions[thread\_count]);

thread\_count++;

}

}

// Collect results from each thread

for (int i = 0; i < thread\_count; i++) {

int \*value;

pthread\_join(threads[i], (void \*\*)&value);

int r = positions[i].row;

int c = positions[i].col;

result[r][c] = \*value;

free(value);

}

// Display Result

printf("\nResultant Matrix:\n");

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

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

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

}

printf("\n");

}

return 0;

}

threads

#include<pthread.h>

void \*thread(void \*vargp);

int main()

{

pthread\_t tid;

pthread\_create(&tid,NULL,thread,NULL);

exit(0);

}

void \*thread(void \*vargp)

{

printf(“Hello World!\n”);

return NULL;

}

Example 2:

void \*thread(void \*vargp);

int main()

{

pthread\_t tid;

pthread\_create(&tid,NULL,thread,NULL);

pthread\_join(tid,NULL);

}

/\* Thread Routine\*/

void \*thread(void \*vargp)

{

printf(“Hello World!\n”);

return NULL;

}

Example 3:

typedef struct

{

int num1;

int num2;

}NUM;

void\* sum\_function(void \*argp);

int main()

{

pthread\_t th1;

NUM N1;

int n1, n2;

int ret\_val;

printf("Enter num1\n");

scanf("%d",&n1);

printf("Enter num2\n");

scanf("%d",&n2);

N1.num1 = n1;

N1.num2 = n2;

pthread\_create(&th1, NULL, sum\_function, (void\*) &N1);

pthread\_join(th1, (void\*)&ret\_val);

printf("sum = %d\n",ret\_val);

return 0;

}

void\* sum\_function(void \*argp)

{

NUM \*N2 = (NUM\*) argp;

int a = N2->num1;

int b = N2->num2;

int sum = a + b;

return (void\*)sum;

}

multiple operations using Multithreading

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

typedef struct {

int a;

int b;

} Operands;

void \*add(void \*args) {

Operands \*op = (Operands \*)args;

int \*res = malloc(sizeof(int));

\*res = op->a + op->b;

pthread\_exit(res);

}

void \*subtract(void \*args) {

Operands \*op = (Operands \*)args;

int \*res = malloc(sizeof(int));

\*res = op->a - op->b;

pthread\_exit(res);

}

void \*multiply(void \*args) {

Operands \*op = (Operands \*)args;

int \*res = malloc(sizeof(int));

\*res = op->a \* op->b;

pthread\_exit(res);

}

void \*divide(void \*args) {

Operands \*op = (Operands \*)args;

float \*res = malloc(sizeof(float));

if (op->b == 0) {

printf("Division by zero error.\n");

\*res = 0.0;

} else {

\*res = (float)op->a / op->b;

}

pthread\_exit(res);

}

int main() {

pthread\_t tid1, tid2, tid3, tid4;

Operands op;

printf("Enter two integers: ");

scanf("%d %d", &op.a, &op.b);

int \*sum, \*diff, \*prod;

float \*quot;

pthread\_create(&tid1, NULL, add, &op);

pthread\_create(&tid2, NULL, subtract, &op);

pthread\_create(&tid3, NULL, multiply, &op);

pthread\_create(&tid4, NULL, divide, &op);

pthread\_join(tid1, (void \*\*)&sum);

pthread\_join(tid2, (void \*\*)&diff);

pthread\_join(tid3, (void \*\*)&prod);

pthread\_join(tid4, (void \*\*)&quot);

printf("\nResults:\n");

printf("Addition: %d + %d = %d\n", op.a, op.b, \*sum);

printf("Subtraction: %d - %d = %d\n", op.a, op.b, \*diff);

printf("Multiplication: %d \* %d = %d\n", op.a, op.b, \*prod);

printf("Division: %d / %d = %.2f\n", op.a, op.b, \*quot);

// Free dynamically allocated memory

free(sum);

free(diff);

free(prod);

free(quot);

return 0;

}