

Term work

**on**

**Operating Systems**

**(PCS 502) 2023-24**

# Submitted to: Submitted by:

Ms. Manika Manwal Priyanshu Tariyal Asst. Professor University Roll. No.: 2118963

GEHU, D.Dun Class Roll. No./Section: 34/O

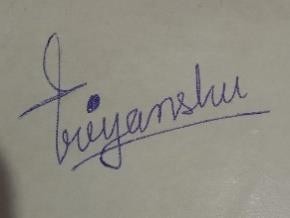
**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# GRAPHIC ERA HILL UNIVERSITY, DEHRADUN

**ACKNOWLEDGMENT**

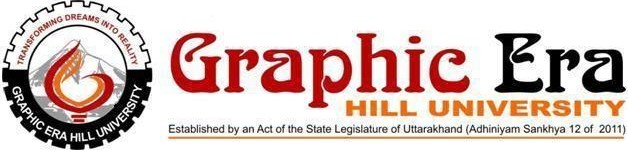
I would like to particularly thank my Operating Systems Lab Faculty **Ms. Manika Manwal** for his patience, support and encouragement throughout the completion of this Term work.

Last but not least I am greatly indebted to all other persons who directly or indirectly helped me during this course.

**Priyanshu Tariyal**

**University. Roll No.- 2118963 B.Tech CSE-O-V Sem Session: 2023-24**

**GEHU, Dehradun**

Table of Contents

|  |  |  |
| --- | --- | --- |
| **Program No.** | **Program Name** | **Page No** |
| 1 | C Program to demonstrate the working of fork() system call |  |
| 2 | C Program in which Parent Process Computes the SUM OF EVEN NUMBERS and Child Process Computes the sum of ODD NUMBERS stored in an array using fork (). First, the child process should print its answer i.e., sum of odd numbers, then the parent should print its answer, i.e., sum of even numbers. |  |
| 3 | C program to Implement the Orphan Process and Zombie Process. |  |
| 4 | C program to Implement FCFS CPU Scheduling Algorithm |  |
| 5 | C program to implement SRTF algorithm. |  |
| 6 | C program to Implement Round Robin CPU scheduling algo. |  |
| 7 | C program to Implement Preemptive Priority CPU scheduling algo |  |
| 8 | C program to Implement Interprocess Communication using PIPE |  |
| 9 | C program to Implement Interprocess Communication using shared memory |  |
| 10 | C program to demonstrate working of excel() where parent process executes "ls" command and child process executes "date" command |  |
| 11 | C program to Implement Banker’s Algo for Deadlock Avoidance to check for the safe and unsafe state. |  |
| 12 | C program to Implement First In First Out page replacement policy |  |

|  |  |  |
| --- | --- | --- |
| 13 | C program to Implement Least recently used page replacement policy |  |
| 14 | C program to Implement FCFS Disk Scheduling Algorithm |  |
| 15. | C program to Implement SCAN Disk Scheduling Algorithm |  |

# Program 1

### Q. Write a C program to demonstrate the use of fork() System call .

***fork()***: It is a system call in a Unix-based system. The Fork system call is used for creating a new process in Linux, and Unix systems, which is called the child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call.

# Source Code:

#include<stdio.h> int main()

{

If(fork() &&(!fork())

{

if(fork() || fork())

{

fork();

printf(“Hello world”);

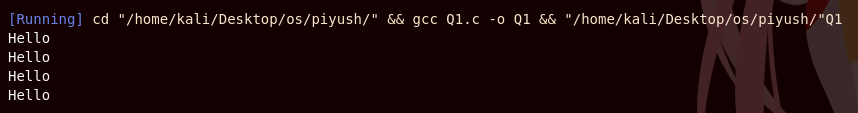
}

}

return 0;

}

# Output



**Program 2**

**Q. C Program in which Parent Process Computes the SUM OF EVEN NUMBERS and Child Process Computes the sum of ODD NUMBERS stored in an array using fork (). First, the child process should print its answer i.e., sum of odd numbers, then the parent should print its answer, i.e., sum of even**

**numbers.**

***Parent Process Computes***: This refers to the computations or tasks that the parent process performs after the fork. The parent process continues its execution from the point of the fork and can perform any necessary calculations or operations.

***Child Process Computes***: This refers to the computations or tasks that the child process performs after the fork. Like the parent process, the child process can also perform its own set of calculations or operations independently.

# Source Code:

#include <stdio.h> #include

<unistd.h> int main()

{

int arr[] = {1, 2, 3, 4, 5, 6, 7};

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

int sum\_odd = 0; int pid = fork(); if (pid == 0)

{

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

{

if (arr[i] % 2 != 0)

{

sum\_odd += arr[i];

}

}

printf("Child Process: Sum of odd numbers = %d\n", sum\_odd);

}

else if (pid > 0)

{

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

{

if (arr[i] % 2 == 0)

{

sum\_even += arr[i];

}

}

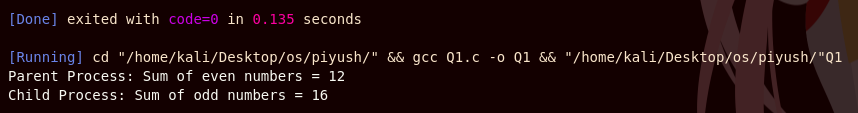
printf("Parent Process: Sum of even numbers = %d\n", sum\_even);

}

return 0;

}

# Output



**Program 3**

### Q. C program to Implement the Orphan Process and Zombie Process.

***Orphan Process:*** A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

***Zombie Process:*** A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process.

# Source Code:

#### Orphan Process:

#include <stdio.h>

#include <stdlib.h> #include

<unistd.h> int main()

{

pid\_t child\_pid = fork(); if (child\_pid < 0)

{

perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child\_pid == 0)

{

printf("| %-10s | %-5d | %-15s |\n", "Child", getpid(), "Running"); sleep(2);

printf("| %-10s | %-5d | %-15s |\n", "Child", getpid(), "Exiting");

}

else

{

printf("| %-10s | %-5d | %-15s |\n", "Parent", getpid(), "Running");

printf("| %-10s | %-5d | %-15s |\n", "Parent", getpid(), "Exiting"); exit(EXIT\_SUCCESS);

}

return 0;

#### Zombie Process:

#include <stdio.h> #include <stdlib.h> #include <unistd.h>

#include <sys/types.h> #include

<sys/wait.h> int main()

{

pid\_t child\_pid = fork(); if (child\_pid < 0)

{

perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child\_pid == 0)

{

printf("| %-12s | %-5s | %-12s |\n", "Process", "PID", "Status"); printf("| | | |\n");

printf("| %-12s | %-5d | %-12s |\n", "Child", getpid(), "Running"); sleep(2);

printf("| %-12s | %-5d | %-12s |\n", "Child", getpid(), "Exiting");

}

else

{

printf("| %-12s | %-5s | %-12s |\n", "Process", "PID", "Status");

printf("| | | |\n");

printf("| %-12s | %-5d | %-12s |\n", "Parent", getpid(), "Running"); printf("| %-12s | %-5d | %-12s |\n", "Parent", getpid(), "Not Waiting");

sleep(5);

printf("| %-12s | %-5d | %-12s |\n", "Parent", getpid(), "Exiting");

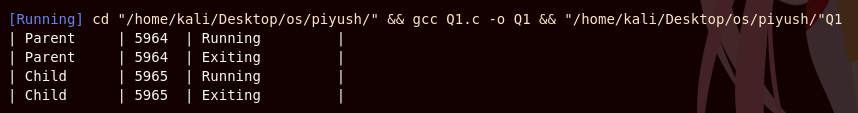
}

return 0;

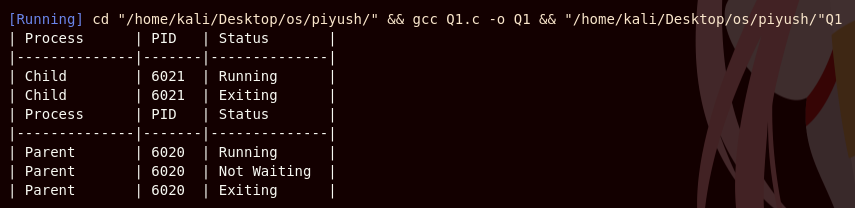
}

# Output

#### Orphan Process:



***Zombie Process:***



# Program 4

##### Q. Write a C program to Implement FCFS CPU Scheduling Algorithm

***FCFS:*** First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue.

In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

# Source Code:

#include <stdio.h> void swap(int \*a, int \*b)

{ int t =

\*a; \*a =

\*b;

\*b = t;

}

void sort\_by\_arrival\_time(int n, int at[], int bt[])

{

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

{

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

{

if (at[j] > at[j + 1])

{

swap(&at[j], &at[j + 1]);

swap(&bt[j], &bt[j + 1]);

}

}

}

}

void fcfs\_scheduling(int n, int at[], int bt[])

{

sort\_by\_arrival\_time(n, at, bt); int current\_time = 0; int ct[n];

float wt[n], total\_wt = 0; float tat[n], total\_tat = 0; int total\_ct = 0;

printf("\nProcesses\t Arrival Time\t Brust Time \tCompeletation Time\t Turnaround Time\t Wating Time");

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

{

if (current\_time < at[i])

{

current\_time = at[i];

}

ct[i] = current\_time + bt[i]; tat[i] = ct[i] - at[i];

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

total\_ct += ct[i]; total\_tat += tat[i]; total\_wt += wt[i];

printf("\n%d\t\t\t%d\t\t%d\t\t%d\t\t\t%.2f\t\t\t%.2f", i + 1, at[i], bt[i], ct[i], tat[i], wt[i]); current\_time = ct[i];

}

printf("\n\n");

printf("\nTotal Waiting Time: %f", total\_wt); printf("\nTotal Turnaround Time: %f", total\_tat); printf("\nTotal CPU Completion Time: %d", total\_ct);

}

int main()

{ int

n;

printf("\nEnter the total no. of processes: "); scanf("%d", &n); int at[n], bt[n];

printf("\nEnter the Arrival time & Burst time: \n"); for (int i = 0; i < n; i++)

{

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

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

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

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

}

printf("\nOriginal Table:\n");

printf("\nProcess\t Arrival Time\t Burst Time\n"); for (int i = 0; i < n; i++)

{

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

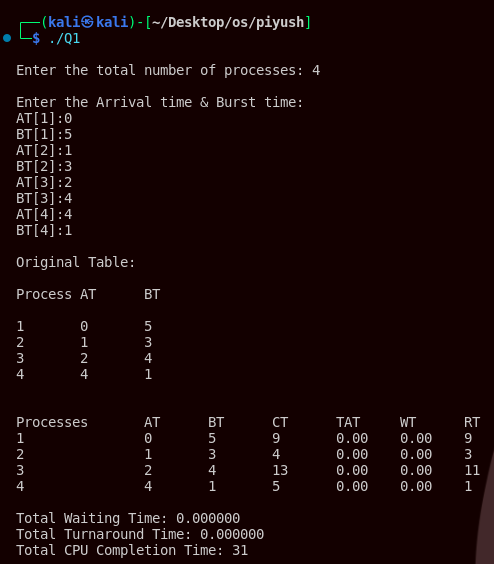
}

printf("\n\n"); fcfs\_scheduling(n, at, bt);

return 0;

}

# Output



**Program 5**

##### Q. Write a C program to implement SRTF algorithm.

***SRTF(Shortest Remaining Time First):*** In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining, and since that time should only reduce as execution progresses, processes will always run until they complete, or a new process is added that requires a smaller amount of time.

***Source Code:***

#include <stdio.h> #include

<limits.h>

void srtf\_scheduling(int n, int at[], int bt[])

{

int current\_time = 0; int remaining\_time[n]; int completed = 0; int ct[n], wt[n], tat[n];

int total\_ct = 0, total\_wt = 0, total\_tat = 0; int response\_time[n];

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

{

remaining\_time[i] = bt[i]; response\_time[i] = INT\_MAX;

}

printf("\nProcesses\tAT\tBT\tCT\tTAT\tWT\tRT"); while (completed < n)

{

int shortest = -1;

int shortest\_time = INT\_MAX;

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

{

if (at[i] <= current\_time && remaining\_time[i] > 0 && remaining\_time[i] < shortest\_time)

{

shortest = i;

shortest\_time = remaining\_time[i];

}

}

if (shortest == -1)

{

}

else

{

current\_time++;

remaining\_time[shortest]--;

if (response\_time[shortest] == INT\_MAX)

{

response\_time[shortest] = current\_time - at[shortest];

}

if (remaining\_time[shortest] == 0)

{

ct[shortest] = current\_time + 1; tat[shortest] = ct[shortest] - at[shortest]; wt[shortest] = tat[shortest] - bt[shortest]; total\_ct += ct[shortest]; total\_tat += tat[shortest]; total\_wt += wt[shortest]; completed++;

}

current\_time++;

}

}

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

{

printf("\n%d\t\t%d\t%d\t%d\t%.2f\t%.2f\t%d", i + 1, at[i], bt[i], ct[i], tat[i], wt[i], response\_time[i]);

}

printf("\n\n");

printf("Total Waiting Time: %f\n", total\_wt); printf("Total Turnaround Time: %f\n", total\_tat); printf("Total CPU Completion Time: %d\n", total\_ct);

}

int main()

{ int n;

printf("\nEnter the total number of processes: "); scanf("%d", &n); int at[n], bt[n];

printf("\nEnter the Arrival time & Burst time: \n"); for (int i = 0; i < n; i++)

{

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

scanf("%d", &at[i]); printf("BT[%d]:", i + 1); scanf("%d", &bt[i]);

}

printf("\nOriginal Table:\n"); printf("\nProcess\tAT\tBT\n"); for (int i = 0; i < n; i++)

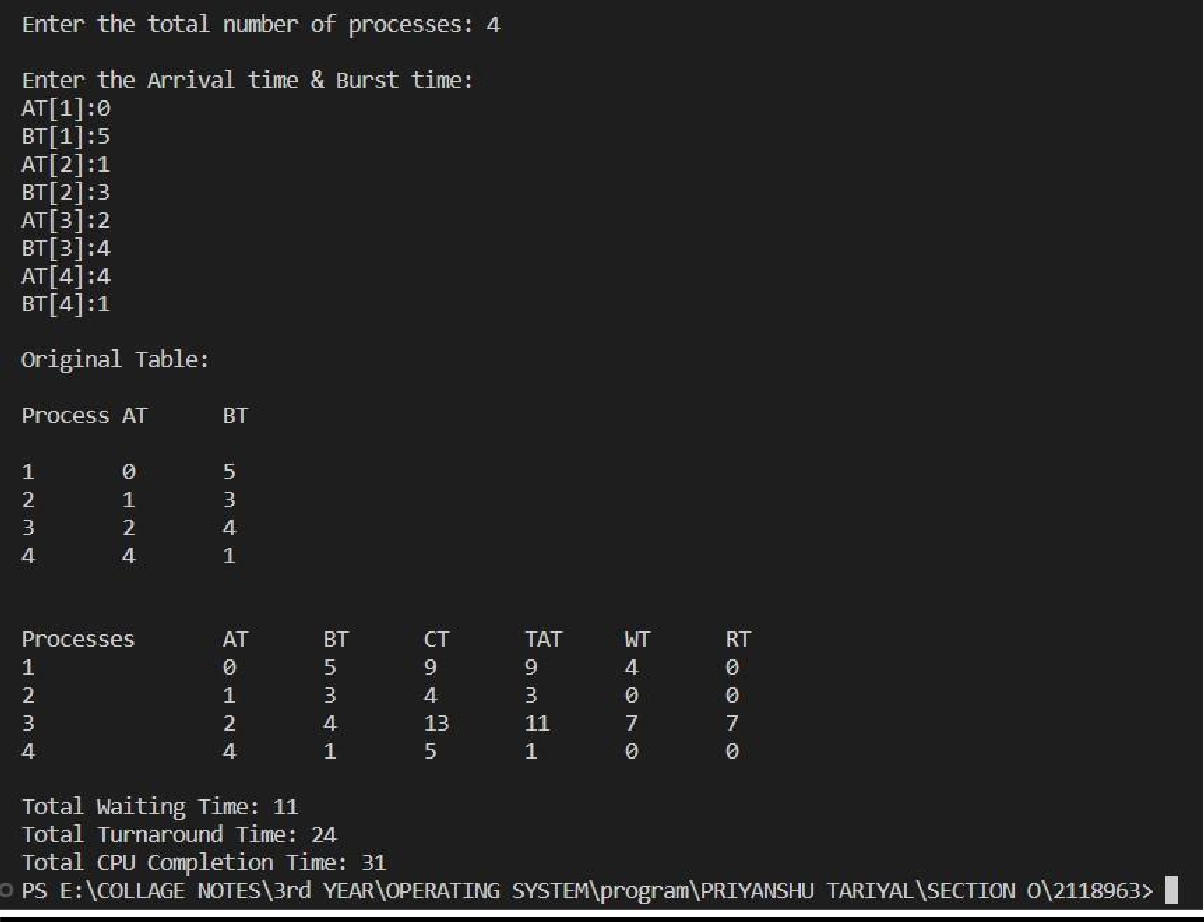
{

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

} printf("\n\n"); srtf\_scheduling(n, at, bt); return 0;

}

# Output



**Program 6**

**Q. Write a C program to Implement Round Robin CPU scheduling algorithm. *Round Robin*** is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is basically the preemptive version of First come First Serve CPU Scheduling algorithm.

## Source Code:

#include <stdio.h>

void RoundRobin(int n, int at[], int bt[], int q)

{

int ct[n], tat[n], wt[n], rt[n], remaining\_time[n]; int current\_time = 0;

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

{

remaining\_time[i] = bt[i];

}

printf("\nProceses\tAT\tBT\tCT\tTAT\tWT\tRT"); while (compeleted < n)

{

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

{

if (remaining\_time[i] > 0)

{

if (at[i] <= current\_time)

{

if (remaining\_time[i] > q)

{

current\_time += q; remaining\_time[i] -= q;

}

else

{

current\_time += remaining\_time[i]; remaining\_time[i] = 0;

ct[i] = current\_time; tat[i] = ct[i] - at[i];

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

rt[i] = ct[i] - at[i] - bt[i];

printf("\n%d\t\t%d\t%d\t%d\t%d\t%d\t%d", i, at[i], bt[i], ct[i], tat[i], wt[i], rt[i]); compeleted++;

}

}

}

}

}

}

int main()

{

int n;

printf("\nEnter the no. of processes: "); scanf("%d", &n);

int at[n], bt[n];

printf("\nEnter the arrival time and burst time: \n"); for (int i = 0; i < n; i++)

{

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

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

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

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

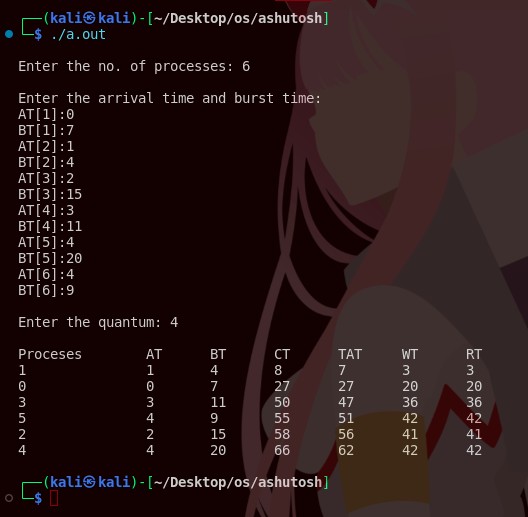
}

int q;

printf("\nEnter the quantum: "); scanf("%d", &q); RoundRobin(n, at, bt, q); return 0;

}

# Output



**Program 7**

**Q. C program to Implement Preemptive Priority CPU scheduling algorithm.**

***Pre-emptive Priority CPU Scheduling Algorithm*** is a pre-emptive method of CPU scheduling algorithm that works based on the priority of a process. In this algorithm, the scheduler schedules the tasks to work as per the priority, which means that a higher priority process should be executed first.

***Source Code:***

#include <stdio.h> #include <limits.h>

void preemptive\_priority\_scheduling(int n, int at[], int bt[], int priority[])

{

int current\_time = 0; int remaining\_time[n]; int completed = 0;

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

int total\_ct = 0, total\_wt = 0, total\_tat = 0; int response\_time[n];

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

{

remaining\_time[i] = bt[i]; response\_time[i] = INT\_MAX;

}

printf("\nProcesses\tAT\tBT\tPriority\tCT\tTAT\tWT\tRT"); while (completed < n)

{

int selected\_process = -1;

int highest\_priority = INT\_MAX; for (int i = 0; i < n; i++)

{

if (at[i] <= current\_time && remaining\_time[i] > 0 && priority[i] < highest\_priority)

{

selected\_process = i; highest\_priority = priority[i];

}

}

if (selected\_process == -1)

{

current\_time++;

}

else

{

remaining\_time[selected\_process]--;

if (response\_time[selected\_process] == INT\_MAX)

{

response\_time[selected\_process] = current\_time - at[selected\_process];

if (remaining\_time[selected\_process] == 0)

{

ct[selected\_process] = current\_time + 1;

tat[selected\_process] = ct[selected\_process] - at[selected\_process]; wt[selected\_process] = tat[selected\_process] - bt[selected\_process]; total\_ct += ct[selected\_process];

total\_tat += tat[selected\_process]; total\_wt += wt[selected\_process]; completed++;

current\_time++;

}

}

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

{

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

}

printf("\n\n");

printf("Total Waiting Time: %d\n", total\_wt); printf("Total Turnaround Time: %d\n", total\_tat); printf("Total CPU Completion Time: %d\n", total\_ct);

}

}

}

int main()

{

int n;

printf("\nEnter the total number of processes: "); scanf("%d", &n);

int at[n], bt[n], priority[n];

printf("\nEnter the Arrival time, Burst time, and Priority: \n"); for (int i = 0; i < n; i++)

{

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

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

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

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

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

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

}

printf("\nOriginal Table:\n"); printf("\nProcess\tAT\tBT\tPriority\n"); for (int i = 0; i < n; i++)

{

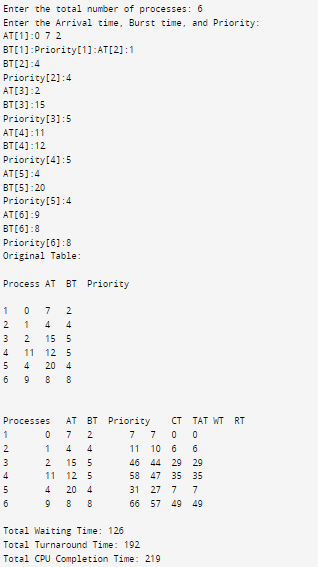
printf("\n%d\t%d\t%d\t%d", i + 1, at[i], bt[i], priority[i]);

}

printf("\n\n");

preemptive\_priority\_scheduling(n, at, bt, priority); return 0;}

# Output



**Program 8**

##### Q. C program to Implement Interprocess Communication using PIPE

***pipe()***: Conceptually, a pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

***Source Code:***

#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <string.h>

int main()

{

int pipe\_fd[2];

pid\_t child\_pid = fork(); if (pipe(pipe\_fd) == -1)

{

perror("Pipe creation failed"); exit(EXIT\_FAILURE);

}

if (child\_pid == -1)

{

perror("Fork failed"); exit(EXIT\_FAILURE);

}

if (child\_pid == 0)

{

close(pipe\_fd[1]); char message[100]; ssize\_t bytes\_read;

bytes\_read = read(pipe\_fd[0], message, sizeof(message)); if (bytes\_read == -1)

{

perror("Read from pipe failed"); exit(EXIT\_FAILURE);

}

message[bytes\_read] = '\0';

printf("\nChild Process: Received message from parent: %s\n", message); close(pipe\_fd[0]);

}

else

{

close(pipe\_fd[0]);

char message[] = "Hello, Child Process!"; ssize\_t bytes\_written;

bytes\_written = write(pipe\_fd[1], message, strlen(message)); if (bytes\_written == -1)

{

perror("Write to pipe failed"); exit(EXIT\_FAILURE);

}

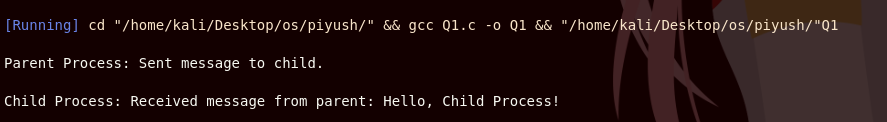
printf("\nParent Process: Sent message to child.\n"); close(pipe\_fd[1]);

}

return 0;

}

# Output



**Program 11**

##### Q. C program to Implement Banker’s Algorithm for Deadlock Avoidance to check for the safe and unsafe state.

***Banker’s algorithm*** is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for the predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue.

***Source code:***

#include <stdio.h>

int main()

{

int n, m, i, j, k; n = 5;

m = 3;

int alloc[5][3] = {{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}};

int max[5][3] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int avail[3] = {3, 3, 2};

int f[n], ans[n], ind = 0; for (k = 0; k < n; k++)

{

f[k] = 0;

}

int need[n][m];

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

{

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

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

}

int y = 0;

for (k = 0; k < 5; k++)

{

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

{

if (f[i] == 0)

{

int flag = 0;

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

{

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

{

flag = 1; break;

}

}

if (flag == 0)

{

ans[ind++] = i;

for (y = 0; y < m; y++) avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

// Print the Banker's Algorithm table printf("\nBanker's Algorithm Table:\n"); printf("Process | Allocation (A B C) | Max Need (A C)

| Available (A B C) | Remaining Need (A B C)\n");

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

{

printf(" P%d |", i); for (j = 0; j < m; j++)

{

printf(" %d", alloc[i][j]); if (j < m - 1)

printf("");

}

printf(" |");

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

{

printf(" %d", max[i][j]); if (j < m - 1)

printf(" ");

}

printf(" |"); if (i == 0)

{

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

{

printf(" %d", avail[j]); if (j < m - 1)

printf(" ");

}

printf(" |");

}

else

{

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

{

printf(" -"); if (j < m - 1) printf(" ");

}

printf(" |");

}

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

{

printf(" %d", need[i][j]); if (j < m - 1)

printf(" ");

}

printf("\n");

}

int flag = 1;

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

{

if (f[i] == 0)

{

flag = 0;

printf("\nThe following system is not safe"); break;

}

}

if (flag == 1)

{

printf("\nFollowing is the SAFE Sequence\n"); for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

printf(" P%d\n", ans[n - 1]);

}

return 0;

}

# Output

# 

**Program 12**

##### Q. C program to Implement First In First Out page replacement policy.

***Source code:***

#include <stdio.h>

int isPagePresent(int page, int frames[], int frameCount) {

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

if (frames[i] == page) {

return 1;

}

}

return 0;

}

void displayFrames(int frames[], int frameCount) {

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

if (frames[i] == -1) {

printf("[ ] ");

} else {

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

}

}

printf("\n");

}

void fifoPageReplacement(int pages[], int pageCount, int maxFrames) {

int frames[maxFrames];

int frameCount = 0;

int pageFaults = 0;

int hits = 0;

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

frames[i] = -1;

}

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

if (isPagePresent(pages[i], frames, frameCount)) {

hits++;

} else {

if (frameCount < maxFrames) {

frames[frameCount] = pages[i];

frameCount++;

} else {

for (int j = 0; j < maxFrames - 1; j++) {

frames[j] = frames[j + 1];

}

frames[maxFrames - 1] = pages[i];

}

pageFaults++;

displayFrames(frames, maxFrames);

}

}

float hitRatio = (float)hits / pageCount;

float missRatio = 1 - hitRatio;

printf("\nHit Ratio: %.2f%%\n", hitRatio \* 100);

printf("Miss Ratio: %.2f%%\n", missRatio \* 100);

}

int main() {

int maxFrames, pageCount; // local var for maxFrame and pagecount

printf("Enter the maximum number of frames: ");

scanf("%d", &maxFrames);

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

scanf("%d", &pageCount);

int pages[pageCount];

printf("Enter the sequence of page numbers:\n");

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

printf("Page %d: ", i + 1);

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

}

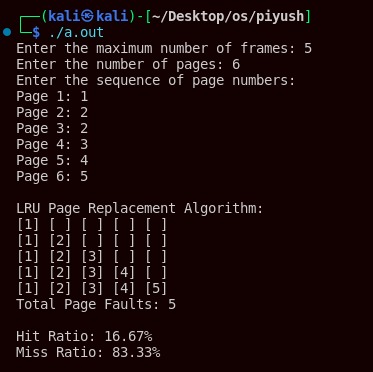
printf("\nFIFO Page Replacement Algorithm:\n");

fifoPageReplacement(pages, pageCount, maxFrames);

return 0;

}

# Output



**Program 13**

##### Q. C program to Implement Least recently used page replacement policy

***Source code:***

##### #include <stdio.h>

##### #include <stdlib.h>

##### int isPagePresent(int page, int frames[], int frameCount) {

##### for (int i = 0; i < frameCount; i++) {

##### if (frames[i] == page) {

##### return 1;

##### }

##### }

##### return 0;

##### }

##### int findLRUIndex(int counters[], int frameCount) {

##### int minIndex = 0;

##### for (int i = 1; i < frameCount; i++) {

##### if (counters[i] < counters[minIndex]) {

##### minIndex = i;

##### }

##### }

##### return minIndex;

##### }

##### void displayFrames(int frames[], int frameCount) {

##### for (int i = 0; i < frameCount; i++) {

##### if (frames[i] == -1) {

##### printf("[ ] ");

##### } else {

##### printf("[%d] ", frames[i]);

##### }

##### }

##### printf("\n");

##### }

##### void lruPageReplacement(int pages[], int pageCount, int maxFrames)

##### {

##### int frames[maxFrames];

##### int counters[maxFrames];

##### int frameCount = 0;

##### int pageFaults = 0;

##### int time = 0;

##### int hits = 0;

##### for (int i = 0; i < maxFrames; i++) {

##### frames[i] = -1;

##### counters[i] = 0;

##### }

##### // Iterate through each page

##### for (int i = 0; i < pageCount; i++) {

##### time++;

##### if (isPagePresent(pages[i], frames, frameCount)) {

##### hits++;

##### for (int j = 0; j < frameCount; j++) {

##### if (frames[j] == pages[i]) {

##### counters[j] = time;

##### break;

##### }

##### }

##### } else {

##### if (frameCount < maxFrames) {

##### frames[frameCount] = pages[i];

##### counters[frameCount] = time;

##### frameCount++;

##### } else {

##### int lruIndex = findLRUIndex(counters, frameCount);

##### frames[lruIndex] = pages[i];

##### counters[lruIndex] = time;

##### }

##### pageFaults++;

##### displayFrames(frames, maxFrames);

##### }

##### }

##### printf("Total Page Faults: %d\n", pageFaults);

##### float hitRatio = (float)hits / pageCount;

##### float missRatio = 1 - hitRatio;

##### printf("\nHit Ratio: %.2f%%\n", hitRatio \* 100);

##### printf("Miss Ratio: %.2f%%\n", missRatio \* 100);

##### }

##### int main()

##### {

##### int maxFrames, pageCount;

##### printf("Enter the maximum number of frames: ");

##### scanf("%d", &maxFrames);

##### printf("Enter the number of pages: ");

##### scanf("%d", &pageCount);

##### int pages[pageCount];

##### printf("Enter the sequence of page numbers:\n");

##### for (int i = 0; i < pageCount; i++) {

##### printf("Page %d: ", i + 1);

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

##### }

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

##### lruPageReplacement(pages, pageCount, maxFrames);

##### return 0;

##### }

# Output

##### 

**Program 9**

##### Q. C program to Implement Interprocess Communication using shared memory

***Source code:***

##### Sender:

##### #include <stdio.h>

##### #include <stdlib.h>

##### #include <sys/types.h>

##### #include <unistd.h>

##### #include <sys/shm.h>

##### #include <string.h>

##### int main(int arg, char\* args[])

##### {

##### void \*SHM;

##### char buff[100];

##### int shmid;

##### shmid = shmget((key\_t)1122,1024,0666|IPC\_CREAT);

##### printf("Key of shmid : %d ", shmid );

##### SHM = shmat(shmid,NULL,0);

##### printf("process attached at %p\n",SHM);

##### printf("Enter the data :");

##### read(0,buff,100);

##### strcpy(SHM, buff);

##### printf("you wrote : %s",(char\*)SHM);

##### return 0;

##### program 2

##### #include <stdio.h>

##### #include <stdlib.h>

##### #include <sys/types.h>

##### #include <unistd.h>

##### #include <sys/shm.h>

##### #include <string.h>

##### int main(int arg, char\* args[])

##### {

##### void \*SHM;

##### char buff[100];

##### int shmid;

##### shmid = shmget((key\_t)1122,1024,0666);

##### printf("Key of shmid : %d ", shmid );

##### SHM = shmat(shmid, NULL, 0);

##### printf("Process attached at %p\n", SHM);

##### printf("Data from sender : %s ",(char\*)SHM);

##### return 0;

##### }

# Output

##### 

**Program 10**

##### Q. C program to demonstrate working of excel() where parent process executes "ls" command and child process executes "date" command

***Source code:***

#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h>

#include <unistd.h>

void child(){

system("ls -1");

printf("\n");

}

void par(){

system("date");

}

int main(int var, char\* varg[])

{

pid\_t pid = fork();

if(pid < 0)

{

printf("fork faild");

return 1;

}

else if(pid == 0)

child();

else{wait(NULL); par();}

return 0;

##### }

# Output

##### A screen shot of a computer Description automatically generated

**Program 14**

##### Q. C program to Implement FCFS Disk Scheduling Algorithm

***Source code:***

##### #include <stdio.h>

##### #include <stdlib.h>

##### int calculateTotalHeadMovement(int requestQueue[], int head, int n) {

##### int totalMovement = 0;

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

##### totalMovement += abs(head - requestQueue[i]);

##### head = requestQueue[i];

##### }

##### return totalMovement;

##### }

##### int main() {

##### int n;

##### printf("Enter the number of disk requests: ");

##### scanf("%d", &n);

##### int requestQueue[n];

##### printf("Enter the disk request queue:\n");

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

##### printf("Request %d: ", i + 1);

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

##### }

##### int initialHead;

##### printf("Enter the initial position of the disk head: ");

##### scanf("%d", &initialHead);

##### int totalHeadMovement = calculateTotalHeadMovement(requestQueue, initialHead, n);

##### printf("Total head movement using FCFS: %d\n", totalHeadMovement);

##### return 0;

##### }

# Output

##### 

**Program 15**

##### Q. C program to Implement SCAN Disk Scheduling Algorithm

***Source code:***

##### #include <stdio.h>

##### #include <stdlib.h>

##### #define SIZE 8

##### #define DISK\_SIZE 200

##### void SCAN(int arr[], int head, char direction) {

##### int seek\_count = 0;

##### int distance, cur\_track;

##### int left[SIZE], right[SIZE];

##### int left\_count = 0, right\_count = 0;

##### int seek\_sequence[SIZE \* 2]; // Maximum possible size for seek sequence

##### // Appending end values which have to be visited before reversing the direction

##### if (direction == 'l') {

##### left[left\_count++] = 0;

##### } else if (direction == 'r') {

##### right[right\_count++] = DISK\_SIZE - 1;

##### }

##### for (int i = 0; i < SIZE; i++) {

##### if (arr[i] < head) {

##### left[left\_count++] = arr[i];

##### }

##### if (arr[i] > head) {

##### right[right\_count++] = arr[i];

##### }

##### }

##### // Sorting left and right arrays

##### for (int i = 0; i < left\_count - 1; i++) {

##### for (int j = 0; j < left\_count - i - 1; j++) {

##### if (left[j] < left[j + 1]) {

##### int temp = left[j];

##### left[j] = left[j + 1];

##### left[j + 1] = temp;

##### }

##### }

##### }

##### for (int i = 0; i < right\_count - 1; i++) {

##### for (int j = 0; j < right\_count - i - 1; j++) {

##### if (right[j] > right[j + 1]) {

##### int temp = right[j];

##### right[j] = right[j + 1];

##### right[j + 1] = temp;

##### }

##### }

##### }

##### // Run the while loop two times, scanning right and left of the head

##### int run = 2;

##### while (run--) {

##### if (direction == 'l') {

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

##### cur\_track = left[i];

##### // Appending current track to seek sequence

##### seek\_sequence[seek\_count++] = cur\_track;

##### // Calculate absolute distance

##### distance = abs(cur\_track - head);

##### // Increase the total count

##### seek\_count += distance;

##### // Accessed track is now the new head

##### head = cur\_track;

##### }

##### direction = 'r';

##### } else if (direction == 'r') {

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

##### cur\_track = right[i];

##### // Appending current track to seek sequence

##### seek\_sequence[seek\_count++] = cur\_track;

##### // Calculate absolute distance

##### distance = abs(cur\_track - head);

##### // Increase the total count

##### seek\_count += distance;

##### // Accessed track is now the new head

##### head = cur\_track;

##### }

##### direction = 'l';

##### }

##### }

##### printf("Total number of seek operations = %d\n", seek\_count);

##### printf("Seek Sequence is\n");

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

##### printf("%d\n", seek\_sequence[i]);

##### }

##### }

##### int main() {

##### // Request array

##### int arr[SIZE] = {176, 79, 34, 60, 92, 11, 41, 114};

##### int head = 50;

##### char direction = 'l';

##### SCAN(arr, head, direction);

##### return 0;

##### }

##### 

# Output

##### 