**LAB EXPERIMENTS**

1.

Code :

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid;

// Create a new process

pid = fork();

if (pid < 0) {

// fork failed

perror("fork failed");

return 1;

} else if (pid == 0) {

// Child process

printf("Child Process:\n");

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

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

} else {

// Parent process

printf("Parent Process:\n");

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

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

}

return 0;

}

Output : Parent Process:

PID: 323

Child PID: 325

2.

Code :

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#define BUFFER\_SIZE 1024

int main(int argc, char \*argv[]) {

int source\_fd, dest\_fd;

char buffer[BUFFER\_SIZE];

ssize\_t bytes\_read, bytes\_written;

if (argc != 3) {

fprintf(stderr, "Usage: %s <source\_file> <destination\_file>\n", argv[0]);

return 1;

}

*// Open source file*

source\_fd = open(argv[1], O\_RDONLY);

if (source\_fd < 0) {

perror("Failed to open source file");

return 1;

}

*// Open or create destination file with write permissions*

dest\_fd = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (dest\_fd < 0) {

perror("Failed to open/create destination file");

close(source\_fd);

return 1;

}

*// Read from source and write to destination*

while ((bytes\_read = read(source\_fd, buffer, BUFFER\_SIZE)) > 0) {

bytes\_written = write(dest\_fd, buffer, bytes\_read);

if (bytes\_written != bytes\_read) {

perror("Write error");

close(source\_fd);

close(dest\_fd);

return 1;

}

}

if (bytes\_read < 0) {

perror("Read error");

}

*// Close file descriptors*

close(source\_fd);

close(dest\_fd);

printf("File copied successfully.\n");

return 0;

}

Output : Usage: /tmp/6XuDWLGoCO/main.o <source\_file> <destination\_file>

3.

Code :

#include <stdio.h>

typedef struct {

int pid;

int burst\_time;

int waiting\_time;

int turnaround\_time;

} Process;

void calculate\_times(Process processes[], int n) {

processes[0].waiting\_time = 0;

// Calculate waiting time for each process

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

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

}

// Calculate turnaround time for each process

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

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

}

}

void print\_schedule(Process processes[], int n) {

float total\_waiting\_time = 0, total\_turnaround\_time = 0;

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

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

total\_waiting\_time += processes[i].waiting\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

printf("P%d\t%d\t\t%d\t\t%d\n",

processes[i].pid,

processes[i].burst\_time,

processes[i].waiting\_time,

processes[i].turnaround\_time);

}

printf("\nAverage Waiting Time = %.2f", total\_waiting\_time / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_turnaround\_time / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

Process processes[n];

// Input burst time for each process

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

processes[i].pid = i + 1;

printf("Enter burst time for process P%d: ", processes[i].pid);

scanf("%d", &processes[i].burst\_time);

}

// Calculate and display scheduling info

calculate\_times(processes, n);

print\_schedule(processes, n);

return 0;

}

Output :

Enter number of processes: 4

Enter burst time for process P1: 5

Enter burst time for process P3: 6

Enter burst time for process P4: 8

Average Turnaround Time = 12.25

Average Waiting Time = 6.75

4.

#include <stdio.h>

typedef struct {

int pid;

int burst\_time;

int waiting\_time;

int turnaround\_time;

} Process;

// Function to sort processes by burst time

void sort\_by\_burst\_time(Process p[], int n) {

Process temp;

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

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

if (p[i].burst\_time > p[j].burst\_time) {

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

// Function to calculate waiting and turnaround times

void calculate\_times(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;

}

}

// Function to display results

void display(Process p[], int n) {

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

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

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

total\_wt += p[i].waiting\_time;

total\_tat += p[i].turnaround\_time;

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

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

Process p[n];

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

p[i].pid = i + 1;

printf("Enter burst time for process P%d: ", p[i].pid);

scanf("%d", &p[i].burst\_time);

}

sort\_by\_burst\_time(p, n);

calculate\_times(p, n);

display(p, n);

return 0;

}

Output :

Enter number of processes: 4

Enter burst time for process P1: 5

Enter burst time for process P4: 7

Enter burst time for process P3: 6

Enter burst time for process P4: 7

Average Waiting Time = 8.50

Average Turnaround Time = 15.00

5.

#include <stdio.h>

typedef struct {

int pid;

int burst\_time;

int priority;

int waiting\_time;

int turnaround\_time;

} Process;

// Sort by priority (ascending order)

void sort\_by\_priority(Process p[], int n) {

Process temp;

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

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

if (p[i].priority > p[j].priority) {

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

}

void calculate\_times(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 display(Process p[], int n) {

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

printf("\nProcess\tPriority\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

total\_wt += p[i].waiting\_time;

total\_tat += p[i].turnaround\_time;

printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n",

p[i].pid, p[i].priority, p[i].burst\_time,

p[i].waiting\_time, p[i].turnaround\_time);

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

}

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

Process p[n];

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

p[i].pid = i + 1;

printf("Enter burst time and priority for process P%d (lower number = higher priority): ", p[i].pid);

scanf("%d %d", &p[i].burst\_time, &p[i].priority);

}

sort\_by\_priority(p, n);

calculate\_times(p, n);

display(p, n);

return 0;

}

Input :

Enter number of processes: 4

Enter burst time and priority for process P1 (lower number = higher priority): 10 3

Enter burst time and priority for process P2 (lower number = higher priority): 1 1

Enter burst time and priority for process P3 (lower number = higher priority): 2 4

Enter burst time and priority for process P4 (lower number = higher priority): 1 2

Output :

Process Priority Burst Time Waiting Time Turnaround Time

P2 1 1 0 1

P4 2 1 1 2

P1 3 10 2 12

P3 4 2 12 14

Average Waiting Time = 3.75

Average Turnaround Time = 7.25

**6.**

#include <stdio.h>

#define MAX 100

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int remaining\_time;

int priority;

int completion\_time;

int waiting\_time;

int turnaround\_time;

int is\_completed;

} Process;

int main() {

int n, time = 0, completed = 0;

Process p[MAX];

printf("Enter number of processes: ");

scanf("%d", &n);

// Input process details

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

p[i].pid = i + 1;

printf("Enter Arrival Time, Burst Time, and Priority for P%d: ", p[i].pid);

scanf("%d %d %d", &p[i].arrival\_time, &p[i].burst\_time, &p[i].priority);

p[i].remaining\_time = p[i].burst\_time;

p[i].is\_completed = 0;

}

while (completed < n) {

int idx = -1;

int highest\_priority = 9999;

// Select the highest priority process that has arrived

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

if (p[i].arrival\_time <= time && !p[i].is\_completed && p[i].priority < highest\_priority && p[i].remaining\_time > 0) {

highest\_priority = p[i].priority;

idx = i;

}

}

if (idx != -1) {

p[idx].remaining\_time--;

time++;

if (p[idx].remaining\_time == 0) {

p[idx].completion\_time = time;

p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

p[idx].is\_completed = 1;

completed++;

}

} else {

time++; // No process is ready; idle time

}

}

// Display results

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

printf("\nProcess\tArrival\tBurst\tPriority\tWaiting\tTurnaround\n");

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

total\_wt += p[i].waiting\_time;

total\_tat += p[i].turnaround\_time;

printf("P%d\t%d\t%d\t%d\t\t%d\t%d\n",

p[i].pid,

p[i].arrival\_time,

p[i].burst\_time,

p[i].priority,

p[i].waiting\_time,

p[i].turnaround\_time);

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

return 0;

}

Input :

Enter the number of processes: 4

Enter Arrival Time, Burst Time and Priority for P1: 0 5 2

Enter Arrival Time, Burst Time and Priority for P2: 1 3 1

Enter Arrival Time, Burst Time and Priority for P3: 2 8 4

Enter Arrival Time, Burst Time and Priority for P4: 3 6 2

Output :

Process Arrival Burst Priority Waiting Turnaround

P1 0 5 2 5 10

P2 1 3 1 0 3

P3 2 8 4 14 22

P4 3 6 2 6 12

Average Waiting Time = 6.25

Average Turnaround Time = 11.75

**7 .**

#include <stdio.h>

#define MAX 100

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int completion\_time;

int waiting\_time;

int turnaround\_time;

int is\_completed;

} Process;

int main() {

int n, time = 0, completed = 0;

Process p[MAX];

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

scanf("%d", &n);

// Input process details

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

p[i].pid = i + 1;

printf("Enter Arrival Time and Burst Time for P%d: ", p[i].pid);

scanf("%d %d", &p[i].arrival\_time, &p[i].burst\_time);

p[i].is\_completed = 0;

}

while (completed < n) {

int idx = -1;

int shortest\_bt = 9999;

// Select the process with shortest burst time among arrived processes

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

if (p[i].arrival\_time <= time && !p[i].is\_completed && p[i].burst\_time < shortest\_bt) {

shortest\_bt = p[i].burst\_time;

idx = i;

}

}

if (idx != -1) {

time += p[idx].burst\_time;

p[idx].completion\_time = time;

p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

p[idx].is\_completed = 1;

completed++;

} else {

time++; // If no process has arrived yet, move time forward

}

}

// Display results

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

printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");

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

total\_wt += p[i].waiting\_time;

total\_tat += p[i].turnaround\_time;

printf("P%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival\_time,

p[i].burst\_time, p[i].waiting\_time, p[i].turnaround\_time);

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

return 0;

}

Input :

Enter the number of processes: 4

Enter Arrival Time and Burst Time for P1: 0 8

Enter Arrival Time and Burst Time for P2: 1 4

Enter Arrival Time and Burst Time for P3: 2 2

Enter Arrival Time and Burst Time for P4: 3 1

Output :

Process Arrival Burst Waiting Turnaround

P1 0 8 9 17

P2 1 4 5 9

P3 2 2 2 4

P4 3 1 0 1

Average Waiting Time = 4.00

Average Turnaround Time = 7.75

**8**.

#include <stdio.h>

#define MAX 100

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int remaining\_time;

int completion\_time;

int waiting\_time;

int turnaround\_time;

} Process;

int main() {

int n, time = 0, completed = 0, tq;

Process p[MAX];

int queue[MAX], front = 0, rear = 0;

int visited[MAX] = {0};

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

scanf("%d", &n);

// Input process info

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

p[i].pid = i + 1;

printf("Enter Arrival Time and Burst Time for P%d: ", p[i].pid);

scanf("%d %d", &p[i].arrival\_time, &p[i].burst\_time);

p[i].remaining\_time = p[i].burst\_time;

}

printf("Enter Time Quantum: ");

scanf("%d", &tq);

// Enqueue first arrived processes

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

if (p[i].arrival\_time == time && !visited[i]) {

queue[rear++] = i;

visited[i] = 1;

}

}

while (completed < n) {

if (front == rear) {

// CPU Idle

time++;

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

if (p[i].arrival\_time == time && !visited[i]) {

queue[rear++] = i;

visited[i] = 1;

}

}

continue;

}

int idx = queue[front++];

int exec\_time = (p[idx].remaining\_time < tq) ? p[idx].remaining\_time : tq;

p[idx].remaining\_time -= exec\_time;

time += exec\_time;

// Check for new arrivals during execution

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

if (p[i].arrival\_time <= time && !visited[i]) {

queue[rear++] = i;

visited[i] = 1;

}

}

if (p[idx].remaining\_time > 0) {

queue[rear++] = idx; // Requeue

} else {

completed++;

p[idx].completion\_time = time;

p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

}

}

// Output

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

printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");

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

total\_wt += p[i].waiting\_time;

total\_tat += p[i].turnaround\_time;

printf("P%d\t%d\t%d\t%d\t%d\n", p[i].pid, p[i].arrival\_time, p[i].burst\_time,

p[i].waiting\_time, p[i].turnaround\_time);

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n);

printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

return 0;

}

Input :

Enter the number of processes: 4

Enter Arrival Time and Burst Time for P1: 0 5

Enter Arrival Time and Burst Time for P2: 1 4

Enter Arrival Time and Burst Time for P3: 2 2

Enter Arrival Time and Burst Time for P4: 4 1

Enter Time Quantum: 2

Output :

Process Arrival Burst Waiting Turnaround

P1 0 5 7 12

P2 1 4 6 10

P3 2 2 2 4

P4 4 1 0 1

Average Waiting Time = 3.75

Average Turnaround Time = 6.75

**9.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <string.h>

#define SHARED\_MEMORY\_NAME "/shm\_example" // Shared memory object name

#define SIZE 1024 // Size of shared memory

int main() {

// Create shared memory

int shm\_fd = shm\_open(SHARED\_MEMORY\_NAME, O\_CREAT | O\_RDWR, 0666);

if (shm\_fd == -1) {

perror("shm\_open");

exit(1);

}

// Set the size of the shared memory

if (ftruncate(shm\_fd, SIZE) == -1) {

perror("ftruncate");

exit(1);

}

// Map the shared memory to the process's address space

char \*shm\_ptr = mmap(0, SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm\_ptr == MAP\_FAILED) {

perror("mmap");

exit(1);

}

// Write a message to shared memory

const char \*message = "Hello from Writer Process!";

strcpy(shm\_ptr, message);

printf("Writer: Wrote message to shared memory: %s\n", message);

// Wait for the reader process to read the message

printf("Writer: Press Enter to exit...\n");

getchar();

// Clean up

munmap(shm\_ptr, SIZE);

close(shm\_fd);

return 0;

}

output :

Writer: Wrote message to shared memory: Hello from Writer Process!

Writer: Press Enter to exit...

**10.**

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <string.h>

#define MSG\_KEY 1234 // Key for the message queue

// Define the message structure

struct message {

long msg\_type; // Message type (must be > 0)

char msg\_text[100]; // Message content

};

int main() {

int msgid;

struct message msg;

// Create a message queue or get its ID

msgid = msgget(MSG\_KEY, IPC\_CREAT | 0666);

if (msgid == -1) {

perror("msgget");

exit(1);

}

// Send multiple messages

for (int i = 1; i <= 3; i++) {

msg.msg\_type = 1; // Set the message type to 1

sprintf(msg.msg\_text, "Hello from Writer Process, message %d", i);

// Send the message to the queue

if (msgsnd(msgid, &msg, sizeof(msg.msg\_text), 0) == -1) {

perror("msgsnd");

exit(1);

}

printf("Writer: Sent message %d: %s\n", i, msg.msg\_text);

sleep(1); // Delay between messages

}

return 0;

}

output :

Writer: Sent message 1: Hello from Writer Process, message 1

Writer: Sent message 2: Hello from Writer Process, message 2

Writer: Sent message 3: Hello from Writer Process, message 3