**EXPERIMENT -1**

1.Create a new process by invoking the appropriate system call. Get the process identifier of the

currently running process and its respective parent using system calls and display the same

using a C program.

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

pid\_t pid;

pid = fork();

if (pid < 0) {

perror("Fork failed");

return 1;

} else if (pid == 0) {

printf("Child Process:\n");

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

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

} else {

printf("Parent Process:\n");

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

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

}

return 0;

}

**SAMPLE OUTPUT**

Parent Process:

Parent PID: 12345

Child PID: 12346

Child Process:

Child PID: 12346

Parent PID: 12345

EXPERIMENT-2

2.Identify the system calls to copy the content of one file to another and illustrate the same using

a C program.

#include <stdio.h>

#include <fcntl.h>

#include <unistd.h>

#include <stdlib.h>

#define BUFFER\_SIZE 1024

int main() {

int sourceFile, destFile;

ssize\_t bytesRead, bytesWritten;

char buffer[BUFFER\_SIZE];

sourceFile = open("source.txt", O\_RDONLY);

if (sourceFile < 0) {

perror("Error opening source file");

exit(1);

}

destFile = open("destination.txt", O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

if (destFile < 0) {

perror("Error opening/creating destination file");

close(sourceFile);

exit(1);

}

while ((bytesRead = read(sourceFile, buffer, BUFFER\_SIZE)) > 0) {

bytesWritten = write(destFile, buffer, bytesRead);

if (bytesWritten != bytesRead) {

perror("Error writing to destination file");

close(sourceFile);

close(destFile);

exit(1);

}

}

close(sourceFile);

close(destFile);

printf("File copied successfully using system calls.\n");

return 0;

}

**EXPERIMENT-3**

3. Design a CPU scheduling program with C using First Come First Served technique with the

following considerations.

a. All processes are activated at time 0.

b. Assume that no process waits on I/O devices.

#include <stdio.h>

int main() {

int n, i;

int burstTime[100], waitingTime[100], turnaroundTime[100];

int totalWaitingTime = 0, totalTurnaroundTime = 0;

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

scanf("%d", &n);

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

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

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

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

}

waitingTime[0] = 0;

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

waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];

}

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

turnaroundTime[i] = waitingTime[i] + burstTime[i];

totalWaitingTime += waitingTime[i];

totalTurnaroundTime += turnaroundTime[i];

}

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

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

printf("P%d\t%d\t\t%d\t\t%d\n", i + 1, burstTime[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = (float)totalWaitingTime / n;

float avgTurnaroundTime = (float)totalTurnaroundTime / n;

printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);

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

return 0;

}

**SAMPLE INPUT**

Enter the number of processes: 3

Enter burst time for each process:

Process 1: 5

Process 2: 3

Process 3: 8

**SAMPLE OUTPUT**

Process Burst Time Waiting Time Turnaround Time

P1 5 0 5

P2 3 5 8

P3 8 8 16

Average Waiting Time: 4.33

Average Turnaround Time: 9.67

**EXPERIMENT-4**

4. Construct a scheduling program with C that selects the waiting process with the smallest

execution time to execute next.

#include <stdio.h>

int main() {

int n, i, j;

int burstTime[100], process[100], waitingTime[100], turnaroundTime[100];

int totalWaitingTime = 0, totalTurnaroundTime = 0;

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

scanf("%d", &n);

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

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

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

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

process[i] = i + 1;

}

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

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

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

// Swap burst times

int temp = burstTime[i];

burstTime[i] = burstTime[j];

burstTime[j] = temp;

int t = process[i];

process[i] = process[j];

process[j] = t;

}

}

}

waitingTime[0] = 0;

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

waitingTime[i] = waitingTime[i - 1] + burstTime[i - 1];

}

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

turnaroundTime[i] = waitingTime[i] + burstTime[i];

totalWaitingTime += waitingTime[i];

totalTurnaroundTime += turnaroundTime[i];

}

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

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

printf("P%d\t%d\t\t%d\t\t%d\n", process[i], burstTime[i], waitingTime[i], turnaroundTime[i]);

}

float avgWaitingTime = (float)totalWaitingTime / n;

float avgTurnaroundTime = (float)totalTurnaroundTime / n;

printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);

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

return 0;

}

**SAMPLE INPUT**

Enter the number of processes: 4

Enter burst time for each process:

Process 1: 6

Process 2: 8

Process 3: 7

Process 4: 3

SAMPLE OUTPUT

Process Burst Time Waiting Time Turnaround Time

P4 3 0 3

P1 6 3 9

P3 7 9 16

P2 8 16 24

Average Waiting Time: 7.00

Average Turnaround Time: 13.00

**EXPERIMENT-5**

5. Construct a scheduling program with C that selects the waiting process with the highest

priority to execute next.

#include <stdio.h>

int main() {

int n;

printf("Enter number of processes: ");

scanf("%d", &n);

int burst[n], priority[n], pid[n];

int waiting[n], turnaround[n];

int i, j;

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

pid[i] = i + 1;

printf("Process %d burst time: ", pid[i]);

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

printf("Process %d priority (lower = higher): ", pid[i]);

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

}

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

int best = i;

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

if (priority[j] < priority[best]) {

best = j;

}

}

int tmp = priority[i]; priority[i] = priority[best]; priority[best] = tmp;

tmp = burst[i]; burst[i] = burst[best]; burst[best] = tmp;

tmp = pid[i]; pid[i] = pid[best]; pid[best] = tmp;

}

waiting[0] = 0;

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

waiting[i] = waiting[i-1] + burst[i-1];

}

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

turnaround[i] = waiting[i] + burst[i];

}

int totalW = 0, totalT = 0;

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

totalW += waiting[i];

totalT += turnaround[i];

}

printf("\nPID\tPri\tBurst\tWait\tTurnaround\n");

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

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

pid[i], priority[i], burst[i], waiting[i], turnaround[i]);

}

printf("\nAverage waiting time: %.2f\n", (float)totalW / n);

printf("Average turnaround time: %.2f\n", (float)totalT / n);

return 0;

}

**SAMPLE**

Enter number of processes: 4

Process 1 burst time: 5

Process 1 priority (lower = higher): 2

Process 2 burst time: 3

Process 2 priority (lower = higher): 1

Process 3 burst time: 8

Process 3 priority (lower = higher): 4

Process 4 burst time: 6

Process 4 priority (lower = higher): 3

PID Pri Burst Wait Turnaround

P2 1 3 0 3

P1 2 5 3 8

P4 3 6 8 14

P3 4 8 14 22

Average waiting time: 6.25

Average turnaround time: 11.75