**Name: LALITH KUMAR**

**Reg No.: 192421321**

**CSA0418- Operating Systems for Network Devices- LAB EXPERIMENTS**

**EXP 1: Process Creation**

#include <stdio.h>

#include <windows.h>

int main() {

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

// Create a new process (child process)

if(CreateProcess(

NULL, // No module name (use command line)

"C:\\Windows\\System32\\notepad.exe", // Child process (can be any executable)

NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

{

printf("Parent Process ID: %lu\n", GetCurrentProcessId());

printf("Child Process ID: %lu\n", pi.dwProcessId);

// Wait for child process to exit

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

else {

printf("Failed to create process.\n");

}

return 0;

}

**EXP 2:**

#include <stdio.h>

int main() {

FILE \*src, \*dest;

char ch;

src = fopen("source.txt", "r");

if (src == NULL) {

printf("Error: Cannot open source file.\n");

return 1;

}

dest = fopen("destination.txt", "w");

if (dest == NULL) {

printf("Error: Cannot open destination file.\n");

fclose(src);

return 1;

}

while ((ch = fgetc(src)) != EOF) {

fputc(ch, dest);

}

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

fclose(src);

fclose(dest);

return 0;

}

**EXP 3:**

#include <stdio.h>

int main() {

int n, i;

int burstTime[20], waitingTime[20], turnaroundTime[20];

float totalWaitingTime = 0, totalTurnaroundTime = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

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

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

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

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

}

waitingTime[0] = 0; // First process has 0 waiting time

// Calculate waiting time for each process

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

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

}

// Calculate turnaround time for each process

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

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

totalWaitingTime += waitingTime[i];

totalTurnaroundTime += turnaroundTime[i];

}

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

printf("Average Turnaround Time = %.2f\n", totalTurnaroundTime / n);

return 0;

}

**EXP 4:**

#include <stdio.h>

// Structure to hold process information

struct Process {

int pid;

int burstTime;

int waitingTime;

int turnaroundTime;

};

int main() {

int n, i, j;

struct Process p[20], temp;

float totalWaitingTime = 0, totalTurnaroundTime = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

// Input burst time for each process

for (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].burstTime);

}

// Sort processes based on burst time (using simple bubble sort)

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

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

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

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

// Calculate waiting time

p[0].waitingTime = 0;

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

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

}

// Calculate turnaround time

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

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

}

// Display results

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", p[i].pid, p[i].burstTime, p[i].waitingTime, p[i].turnaroundTime);

totalWaitingTime += p[i].waitingTime;

totalTurnaroundTime += p[i].turnaroundTime;

}

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

printf("Average Turnaround Time = %.2f\n", totalTurnaroundTime / n);

return 0;

}

**EXP 5:**

#include <stdio.h>

// Structure for process

struct Process {

int pid;

int burstTime;

int priority;

int waitingTime;

int turnaroundTime;

};

int main() {

int n, i, j;

struct Process p[20], temp;

float totalWaitingTime = 0, totalTurnaroundTime = 0;

printf("Enter number of processes: ");

scanf("%d", &n);

// Input burst time and priority

for (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].burstTime);

printf("Enter Priority for Process P%d (lower number = higher priority): ", p[i].pid);

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

}

// Sort processes based on priority (ascending order)

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

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

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

temp = p[i];

p[i] = p[j];

p[j] = temp;

}

}

}

// Calculate waiting time

p[0].waitingTime = 0;

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

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

}

// Calculate turnaround time

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

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

}

// Display results

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

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

printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].pid, p[i].priority, p[i].burstTime, p[i].waitingTime, p[i].turnaroundTime);

totalWaitingTime += p[i].waitingTime;

totalTurnaroundTime += p[i].turnaroundTime;

}

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

printf("Average Turnaround Time = %.2f\n", totalTurnaroundTime / n);

return 0;

}