**PROGRAM 1:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main() {

pid\_t pid, child\_pid;

int status;

pid = fork();

if (pid < 0) {

perror("fork failed");

exit(EXIT\_FAILURE);

}

if (pid == 0) {

printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());

execlp("ls", "ls", NULL);

perror("execlp failed");

exit(EXIT\_FAILURE);

} else {

printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);

child\_pid = wait(&status);

if (child\_pid == -1) {

perror("wait failed");

exit(EXIT\_FAILURE);

}

if (WIFEXITED(status)) {

printf("Child process exited with status %d\n", WEXITSTATUS(status));

} else {

printf("Child process did not exit normally\n");

}

}

return 0;

}

**PROGRAM 2:**

#include <stdio.h>

#include <stdlib.h>

typedef struct {

int id;

int burst\_time;

int priority;

} Process;

void fcfs\_scheduling(int n, int burst\_times[]) {

int waiting\_time[n], turnaround\_time[n];

waiting\_time[0] = 0;

turnaround\_time[0] = burst\_times[0];

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

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

turnaround\_time[i] = waiting\_time[i] + burst\_times[i];

}

printf("FCFS Scheduling\n");

printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst\_times[i], waiting\_time[i], turnaround\_time[i]);

}

}

int compare\_sjf(const void \*a, const void \*b) {

return ((Process \*)a)->burst\_time - ((Process \*)b)->burst\_time;

}

void sjf\_scheduling(int n, Process processes[]) {

int waiting\_time[n], turnaround\_time[n];

qsort(processes, n, sizeof(Process), compare\_sjf);

waiting\_time[0] = 0;

turnaround\_time[0] = processes[0].burst\_time;

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

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

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

}

printf("SJF Scheduling\n");

printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

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

}

}

void round\_robin\_scheduling(int n, int burst\_times[], int quantum) {

int remaining\_times[n], waiting\_time[n], turnaround\_time[n];

int t = 0;

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

remaining\_times[i] = burst\_times[i];

}

while (1) {

int done = 1;

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

if (remaining\_times[i] > 0) {

done = 0;

if (remaining\_times[i] > quantum) {

t += quantum;

remaining\_times[i] -= quantum;

} else {

t += remaining\_times[i];

waiting\_time[i] = t - burst\_times[i];

remaining\_times[i] = 0;

}

}

}

if (done) {

break;

}

}

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

turnaround\_time[i] = burst\_times[i] + waiting\_time[i];

}

printf("Round Robin Scheduling\n");

printf("Process ID\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("P%d\t\t%d\t\t%d\t\t%d\n", i + 1, burst\_times[i], waiting\_time[i], turnaround\_time[i]);

}

}

int compare\_priority(const void \*a, const void \*b) {

return ((Process \*)a)->priority - ((Process \*)b)->priority;

}

void priority\_scheduling(int n, Process processes[]) {

int waiting\_time[n], turnaround\_time[n];

qsort(processes, n, sizeof(Process), compare\_priority);

waiting\_time[0] = 0;

turnaround\_time[0] = processes[0].burst\_time;

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

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

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

}

printf("Priority Scheduling\n");

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

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

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

}

}

int main() {

int n, quantum;

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

scanf("%d", &n);

int burst\_times[n];

Process processes[n];

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

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

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

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

processes[i].id = i + 1;

processes[i].burst\_time = burst\_times[i];

}

printf("Enter the quantum time for Round Robin (0 to skip): ");

scanf("%d", &quantum);

if (quantum > 0) {

round\_robin\_scheduling(n, burst\_times, quantum);

}

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

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

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

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

}

fcfs\_scheduling(n, burst\_times);

sjf\_scheduling(n, processes);

priority\_scheduling(n, processes);

return 0;

}