PROCESS CREATION

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

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        // Fork failed
        fprintf(stderr, "Fork Failed");
        return 1;
    } else if (pid == 0) {
        // Child process
        printf("Hello from the Child process!\n");
    } else {
        // Parent process
        printf("Hello from the Parent process!\n");
    }

    return 0;
}
```

FILE COPYING

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

void copyFile(const char *sourceFile, const char *destinationFile) {
    FILE *source, *dest;
```

```
    char ch;

    // Open source file
    source = fopen(sourceFile, "r");
    if (source == NULL) {
        printf("Source file could not be opened.\n");
        exit(EXIT_FAILURE);
    }

    // Open destination file
    dest = fopen(destinationFile, "w");
    if (dest == NULL) {
        printf("Destination file could not be opened.\n");
        fclose(source);
        exit(EXIT_FAILURE);
    }

    // Copy file content
    while ((ch = fgetc(source)) != EOF) {
        fputc(ch, dest);
    }

    // Close files
    fclose(source);
    fclose(dest);

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

int main() {
    char sourceFile[100], destinationFile[100];
```

```
    printf("Enter source file name: ");
    scanf("%s", sourceFile);

    printf("Enter destination file name: ");
    scanf("%s", destinationFile);

    copyFile(sourceFile, destinationFile);

    return 0;
}
```

FCFS scheduling

```
#include <stdio.h>

struct Process {
    int pid; // Process ID
    int burstTime; // Burst Time
    int waitingTime; // Waiting Time
    int turnaroundTime; // Turnaround Time
};

void calculateWaitingTime(struct Process processes[], int n) {
    processes[0].waitingTime = 0; // First process has 0 waiting time

    // Calculate waiting time for each process
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }
}
```

```
void calculateTurnaroundTime(struct Process processes[], int n) {
    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
    }
}

void printProcesses(struct Process processes[], int n) {
    printf("PID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime,
processes[i].waitingTime, processes[i].turnaroundTime);
    }
}

int main() {
    int n;

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

    struct 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 %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
    }
```

```
    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);

    printf("First-Come, First-Served Scheduling (FCFS):\n");
    printProcesses(processes, n);

    return 0;
}
```

SJF Scheduling

```
#include <stdio.h>

struct Process {
    int pid; // Process ID
    int burstTime; // Burst Time
    int waitingTime; // Waiting Time
    int turnaroundTime; // Turnaround Time
};

void sortProcessesByBurstTime(struct Process processes[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].burstTime > processes[j].burstTime) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
}
```

```
void calculateWaitingTime(struct Process processes[], int n) {
    processes[0].waitingTime = 0; // First process has 0 waiting time

    // Calculate waiting time for each process
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }
}

void calculateTurnaroundTime(struct Process processes[], int n) {
    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
    }
}

void printProcesses(struct Process processes[], int n) {
    printf("PID\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime,
processes[i].waitingTime, processes[i].turnaroundTime);
    }
}

int main() {
    int n;

    printf("Enter the number of processes: ");
    scanf("%d", &n);
```

```
    struct 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 %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
    }

    // Sort processes by burst time
    sortProcessesByBurstTime(processes, n);

    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);

    printf("Shortest Job First Scheduling (SJF):\n");
    printProcesses(processes, n);

    return 0;
}
```

PRIORITY scheduling

```
#include <stdio.h>

struct Process {
    int pid; // Process ID
    int burstTime; // Burst Time
    int priority; // Priority
    int waitingTime; // Waiting Time
    int turnaroundTime; // Turnaround Time
```

```
};

void sortProcessesByPriority(struct Process processes[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].priority > processes[j].priority) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
}

void calculateWaitingTime(struct Process processes[], int n) {
    processes[0].waitingTime = 0; // First process has 0 waiting time

    // Calculate waiting time for each process
    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }
}

void calculateTurnaroundTime(struct Process processes[], int n) {
    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;
    }
}
```

```
void printProcesses(struct Process processes[], int n) {
    printf("PID\tPriority\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].priority,
processes[i].burstTime, processes[i].waitingTime, processes[i].turnaroundTime);
    }
}

int main() {
    int n;

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

    struct Process processes[n];

    // Input burst time and priority for each process
    for (int i = 0; i < n; i++) {
        processes[i].pid = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
        printf("Enter priority for process %d: ", i + 1);
        scanf("%d", &processes[i].priority);
    }

    // Sort processes by priority
    sortProcessesByPriority(processes, n);

    calculateWaitingTime(processes, n);
    calculateTurnaroundTime(processes, n);
```

```
    printf("Priority Scheduling:\n");
    printProcesses(processes, n);

    return 0;
}
```