Qs. Implement First Come First Serve Scheduling Algorithm (FCFS)

Ans.

```
#include <stdio.h>

struct Process {
    int process_id;
    int arrival_time;
    int burst_time;
};

void fcfs(struct Process processes[], int n) {
    int waiting_time[n];
    int turnaround_time[n];

    // Waiting time for the first process is 0
    waiting_time[0] = 0;

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

    // Calculate turnaround time for each process
    for (int i = 0; i < n; i++) {
        turnaround_time[i] = waiting_time[i] + processes[i].burst_time;
    }

    // Calculate average waiting time and average turnaround time
    float avg_waiting_time = 0;
    float avg_turnaround_time = 0;
    for (int i = 0; i < n; i++) {
```

```
        avg_waiting_time += waiting_time[i];
        avg_turnaround_time += turnaround_time[i];
    }
    avg_waiting_time /= n;
    avg_turnaround_time /= n;

    // Print the results
    printf("Process\tArrival Time\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].process_id, processes[i].arrival_time,
               processes[i].burst_time, waiting_time[i], turnaround_time[i]);
    }

    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
    printf("Average Turnaround Time: %.2f\n", avg_turnaround_time);
}

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

    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].process_id = i + 1;
        printf("Enter Arrival Time for Process %d: ", i + 1);
        scanf("%d", &processes[i].arrival_time);
        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &processes[i].burst_time);
```

```
    }

    fcfs(processes, n);

    return 0;
}
```

Qs . Implement Orphan Process

Ans.

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

int main() {
   printf("ORPHAN PROCESS\n");

   int p_id = fork(); // Create a new process

   if (p_id == 0) {
      // Code in the child process
      sleep(10); // Child sleeps for 10 seconds
      printf("Child process\n");
   }

   if (p_id > 0) {
      // Code in the parent process
      printf("Parent process\n");
   }

   return 0;
}
```

Qs. Implement pipe communication.

Ans.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    int fd[2], n, p;
    char buffer[50];

    if (pipe(fd) == -1) {
        perror("Pipe creation failed");
        return 1;
    }

    p = fork();

    if (p < 0) {
        perror("Fork failed");
        return 1;
    }

    if (p > 0) {
        close(fd[0]); // Close the read end of the pipe in the parent process
        printf("\nPassing Values to child (PID=%d)\n", getpid());
        write(fd[1], "Deemed Geu\n", 11); // Write data to the pipe
```

```
        close(fd[1]); // Close the write end of the pipe in the parent process
    } else {
        close(fd[1]); // Close the write end of the pipe in the child process
        printf("\nChild received the data (PID=%d)\n", getpid());
        n = read(fd[0], buffer, sizeof(buffer)); // Read data from the pipe
        close(fd[0]); // Close the read end of the pipe in the child process
        write(1, buffer, n); // Write the received data to the standard output (stdout)
    }
    return 0;
}
```

Qs. Zombie process

Ans.

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

int main() {
   printf("ZOMBIE PROCESS\n");

   int p_id = fork(); // Create a new process

   if (p_id > 0) {
      // Code in the parent process
      sleep(10); // Parent sleeps for 10 seconds
      printf("Parent process\n");
   } else {
      // Code in the child process
      exit(0); // Child exits immediately
   }

   return 0;
}
```

Qs. Avoiding Zombie Process

Ans.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
int main()
{
        int i;
        int p = fork();
        if (p==0)
        {
                for (i=0; i<10; i++)
                        printf("I am Child Process\n");
        }
        else
        {
                wait(NULL);
                printf("I am Parent Process\n");
                while(1);
        }
}
```