Question : 1

Create a program in C to implement a multi-process environment and print Hello from all the processes.

Code

```c
#include<stdio.h>
#include<unistd.h>
int main(){
    pid_t pid  = fork();
    if(pid == 0){
        //child processs
        printf("hello from child process(pid : %d) \n",getpid());
    }
    else
    {
        printf("hello from parent process (pid : %d)\n",getpid());
    }
    return 0;
}
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc question1.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
hello from parent process (pid : 7613)
hello from child process(pid : 7614)
```

Question : 2

Create a program in C to implement a multi-process environment and print Hello from the parent process and Hi from the child process.

Code

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

int main(){
    pid_t pid = fork();
    if(pid==0){
        printf("hi from child process (pid : %d) \n", getpid());
    }
    else{
        printf("hello from parent process (pid :%d) \n ", getpid());
    }
    return 0;
}
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc question2.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
hello from parent process (pid :8408)
 hi from child process (pid : 8409)
```

Question : 3

Create a program in C to implement a multi-process environment and print Hello from the parent process and Hi from the child process and before exiting they have to say goodbye. But the parent must wait for the child to complete its execution.

Code

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

int id;
int i;

int main() {
    for (i = 1; i <= 5; i++) {
        id = fork();

        if (id == 0) {
            // Child process
            printf("Hi from child process (pid = %d) and (iteration = %d)\n", getpid(), i);
            printf("Goodbye from child process (pid = %d)\n", getpid());
            exit(0);  // Ensure the child process terminates
        }
        else if (id > 0) {

            wait(NULL);  // Wait for the child process to finish
            printf("Hello from parent process (pid = %d) and (iteration = %d)\n", getpid(), i);
            printf("Goodbye from parent process (pid = %d)\n", getpid());
        }
        else {

            perror("Fork failed");
            exit(1);
        }
    }

    return 0;
}
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc question3.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
Hi from child process (pid = 9051) and (iteration = 1)
Goodbye from child process (pid = 9051)
Hello from parent process (pid = 9050) and (iteration = 1)
Goodbye from parent process (pid = 9050)
Hi from child process (pid = 9052) and (iteration = 2)
Goodbye from child process (pid = 9052)
Hello from parent process (pid = 9050) and (iteration = 2)
Goodbye from parent process (pid = 9050)
Hi from child process (pid = 9053) and (iteration = 3)
Goodbye from child process (pid = 9053)
Hello from parent process (pid = 9050) and (iteration = 3)
Goodbye from parent process (pid = 9050)
Hi from child process (pid = 9054) and (iteration = 4)
Goodbye from child process (pid = 9054)
Hello from parent process (pid = 9050) and (iteration = 4)
Goodbye from parent process (pid = 9050)
Hi from child process (pid = 9055) and (iteration = 5)
Goodbye from child process (pid = 9055)
Hello from parent process (pid = 9050) and (iteration = 5)
Goodbye from parent process (pid = 9050)
```

Question : 4

Create a program in C to implement a multi-process environment and pass messages among them using pipe(). Print from 1 to 100 by printing alternative numbers by the processes.

Code

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

int main() {
    int parent_to_child[2], child_to_parent[2];
    if(pipe(parent_to_child) < 0 || pipe(child_to_parent) < 0) {
        perror("Pipe Creation Failed!");
        return 1;
    }

    pid_t id = fork();
    if(id < 0) {
        perror("Fork Failed");
        return 2;
    } else if(id == 0) {
        // Child Process
        close(parent_to_child[1]);
        close(child_to_parent[0]);

        int num;
        while(1) {
            read(parent_to_child[0], &num, sizeof(int));
            if(num > 100)
                break;

            printf("Child  Printed: %d\t\t", num);
            num++;

            write(child_to_parent[1], &num, sizeof(int));
        }

        close(parent_to_child[0]);
        close(child_to_parent[1]);
    } else {
        // Parent Process
        close(parent_to_child[0]);
        close(child_to_parent[1]);

```

```
        int num = 1;
        while(1) {
            write(parent_to_child[1], &num, sizeof(int));
            read(child_to_parent[0], &num, sizeof(int));
            if(num > 100)
                break;

            printf("Parent Printed: %d\t\t", num);
            num++;
        }

        close(parent_to_child[1]);
        close(child_to_parent[0]);
        wait(NULL);
        printf("\n");
    }

    return 0;
}
```

Output

```
Child  Printed: 1              Child  Printed: 3              Child  Printed: 5              Child  Printed: 7              Child  Printed: 9
hild  Printed: 11             Child  Printed: 13             Child  Printed: 15             Child  Printed: 17             Child  Printed: 19
hild  Printed: 21             Child  Printed: 23             Child  Printed: 25             Child  Printed: 27             Child  Printed: 29
hild  Printed: 31             Child  Printed: 33             Child  Printed: 35             Child  Printed: 37             Child  Printed: 39
hild  Printed: 41             Child  Printed: 43             Child  Printed: 45             Child  Printed: 47             Child  Printed: 49
hild  Printed: 51             Child  Printed: 53             Child  Printed: 55             Child  Printed: 57             Child  Printed: 59
hild  Printed: 61             Child  Printed: 63             Child  Printed: 65             Child  Printed: 67             Child  Printed: 69
hild  Printed: 71             Child  Printed: 73             Child  Printed: 75             Child  Printed: 77             Child  Printed: 79
hild  Printed: 81             Child  Printed: 83             Child  Printed: 85             Child  Printed: 87             Child  Printed: 89
hild  Printed: 91             Child  Printed: 93             Child  Printed: 95             Child  Printed: 97             Child  Printed: 99
arent Printed: 2              Parent Printed: 4              Parent Printed: 6              Parent Printed: 8              Parent Printed: 10
arent Printed: 12             Parent Printed: 14             Parent Printed: 16             Parent Printed: 18             Parent Printed: 20
arent Printed: 22             Parent Printed: 24             Parent Printed: 26             Parent Printed: 28             Parent Printed: 30
arent Printed: 32             Parent Printed: 34             Parent Printed: 36             Parent Printed: 38             Parent Printed: 40
arent Printed: 42             Parent Printed: 44             Parent Printed: 46             Parent Printed: 48             Parent Printed: 50
arent Printed: 52             Parent Printed: 54             Parent Printed: 56             Parent Printed: 58             Parent Printed: 60
arent Printed: 62             Parent Printed: 64             Parent Printed: 66             Parent Printed: 68             Parent Printed: 70
arent Printed: 72             Parent Printed: 74             Parent Printed: 76             Parent Printed: 78             Parent Printed: 80
arent Printed: 82             Parent Printed: 84             Parent Printed: 86             Parent Printed: 88             Parent Printed: 90
arent Printed: 92             Parent Printed: 94             Parent Printed: 96             Parent Printed: 98             Parent Printed: 100
```

Question : 5

Create a C program to solve the producer consumer problem where producer and consumers are two processes. Producer will produce in a theoretically infinite sized buffer and the consumer will consume from it. Incorporate empty buffer scenario and some waiting time for each process so that their production and consumption can be observed.

Code

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

struct Node {
    int product;
    struct Node* next;
};

int sem_empty = 1;
int sem_full = 0;
int mutex = 1;

void wait(int* x) {
    if (*x > 0) {
        (*x)--;
    }
}

void signal(int* x) {
    (*x)++;
}

struct Node* create_node(int product) {
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->product = product;
    new_node->next = NULL;
    return new_node;
}

void produce(struct Node** buffer) {
    int product;
    printf("Enter the product value: ");
    scanf("%d", &product);

    wait(&sem_empty);
    wait(&mutex);

```

```
    struct Node* new_node = create_node(product);
    new_node->next = *buffer;
    *buffer = new_node;

    signal(&mutex);
    signal(&sem_full);
    printf("Produced: %d\n", product);
}

void consume(struct Node** buffer) {
    if (*buffer == NULL) {
        printf("Buffer is empty!\n");
        return;
    }

    wait(&sem_full);
    wait(&mutex);

    struct Node* temp = *buffer;
    *buffer = (*buffer)->next;
    printf("Consumed: %d\n", temp->product);

    free(temp);
    signal(&mutex);
    signal(&sem_empty);
}

int main() {
    struct Node* buffer = NULL;
    int choice;

    while (1) {
        printf("\nEnter 1 to produce a product, 2 to consume a product, or
    any other key to exit: ");
        scanf("%d", &choice);

        if (choice == 1) {
            produce(&buffer);
        } else if (choice == 2) {
            consume(&buffer);
        } else {
            break;
        }
    }

    while (buffer != NULL) {
        struct Node* temp = buffer;
        buffer = buffer->next;
        free(temp);
    }
```

```
87
88      return 0;
89  }
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc question5_pro_consume.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 1
Enter the product value: 45
Produced: 45

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 1
Enter the product value: 67
Produced: 67

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 1
Enter the product value: 87
Produced: 87

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 2
Consumed: 87

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 2
Consumed: 67

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 2
Consumed: 45

Enter 1 to produce a product, 2 to consume a product, or any other key to exit: 2
Buffer is empty!

Enter 1 to produce a product, 2 to consume a product, or any other key to exit:
```

Question : 6

Using the following processes (with burst time given in brackets) P(8), Q(13), R(10), S(12) and T(15) have arrived at the same time. Write a C codes to depict their completion, turnaround and waiting time in tabular form when the processes are scheduled using :

- FCFS method.
- SJF method.
- Round Robin method with the time quantum of 3 units.

FCFS method

```c
#include<stdio.h>

struct process {
    int Id;
    int Arrival_Time;
    int Brust_Time;
    int Completion_Time;
    int Turn_Around_Time;
    int Waiting_Time;
};

void Find_Completion_Time(struct process proc[], int n) {
    // First process starts execution immediately after its arrival
    proc[0].Completion_Time = proc[0].Arrival_Time + proc[0].Brust_Time;

    // For the subsequent processes
    for (int i = 1; i < n; i++) {
        // If the next process arrives after the previous process completes, the CPU stays idle
        if (proc[i].Arrival_Time > proc[i - 1].Completion_Time) {
            proc[i].Completion_Time = proc[i].Arrival_Time + proc[i].Brust_Time;
        } else {
            proc[i].Completion_Time = proc[i - 1].Completion_Time + proc[i].Brust_Time;
        }
    }
}

void TAT_AND_WT(struct process proc[], int n) {
    for (int i = 0; i < n; i++) {
        proc[i].Turn_Around_Time = proc[i].Completion_Time - proc[i].Arrival_Time;
```

```
        proc[i].Waiting_Time = proc[i].Turn_Around_Time - proc[i].Brust_Time;
    }
}

void Calculate_Scheduling_Length_and_Idleness(struct process proc[], int n, int *idle_time, float *L) {
    int max_completion_time = proc[0].Completion_Time;
    int min_arrival_time = proc[0].Arrival_Time;

    *idle_time = 0;

    // Find the max completion time and min arrival time, also calculate idle time
    for (int i = 0; i < n; i++) {
        if (proc[i].Completion_Time > max_completion_time) {
            max_completion_time = proc[i].Completion_Time;
        }
        if (proc[i].Arrival_Time < min_arrival_time) {
            min_arrival_time = proc[i].Arrival_Time;
        }

        // Check idle time between processes
        if (i > 0 && proc[i].Arrival_Time > proc[i - 1].Completion_Time) {
            *idle_time += proc[i].Arrival_Time - proc[i - 1].Completion_Time;
        }
    }

    // Calculate scheduling length
    *L = max_completion_time - min_arrival_time;

    // Handle CPU idle time at the start
    if (proc[0].Arrival_Time > 0) {
        *L = max_completion_time - proc[0].Arrival_Time;
    }
}

void Average_Time(struct process proc[], int n) {
    int Total_TAT = 0, Total_WT = 0;
    for (int i = 0; i < n; i++) {
        Total_TAT += proc[i].Turn_Around_Time;
        Total_WT += proc[i].Waiting_Time;
    }
    printf("\nAverage Turnaround Time: %.2f", (float)Total_TAT / n);
    printf("\nAverage Waiting Time: %.2f\n", (float)Total_WT / n);
}

void Throughput_and_Idleness(int n, float L, int idle_time) {
    // Throughput
```

```
    float throughput = (float)n / L;
    printf("\nThroughput: %.2f", throughput);

    // Idle time percentage
    float idle_percentage = (float)idle_time / L * 100;
    printf("\nIdle Time Percentage: %.2f%%\n", idle_percentage);
}

void Display_process(struct process proc[], int n) {
    printf("\nProcess ID\tArrival Time\tBurst Time\tCompletion Time\
    tTurnaround Time\tWaiting Time");
    for (int i = 0; i < n; i++) {
        printf("\nP%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d", proc[i].Id, proc[i].
    Arrival_Time, proc[i].Brust_Time, proc[i].Completion_Time, proc[i].
    Turn_Around_Time, proc[i].Waiting_Time);
    }
}

int main() {
    int n;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    struct process proc[n];

    // Input details for processes
    for (int i = 0; i < n; i++) {
        proc[i].Id = i + 1;
        printf("\nEnter the Arrival and Burst time for Process P%d: ", i +
    1);
        scanf("%d %d", &proc[i].Arrival_Time, &proc[i].Brust_Time);
    }

    // Sort the processes by their arrival time (if not already sorted)
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].Arrival_Time > proc[j].Arrival_Time) {
                struct process temp = proc[i];
                proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }

    Find_Completion_Time(proc, n);
    TAT_AND_WT(proc, n);

    int idle_time = 0;
    float L = 0;
    Calculate_Scheduling_Length_and_Idleness(proc, n, &idle_time, &L);

```

```
    Display_process(proc, n);
    Average_Time(proc, n);
    Throughput_and_Idleness(n, L, idle_time);

    return 0;
}
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc fcfs.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
Enter number of processes: 5

Enter the Arrival and Burst time for Process P1: 0 8

Enter the Arrival and Burst time for Process P2: 0 13

Enter the Arrival and Burst time for Process P3: 0 10

Enter the Arrival and Burst time for Process P4: 0 12

Enter the Arrival and Burst time for Process P5: 0 15

Process ID      Arrival Time    Burst Time      Completion Time Turnaround Time Waiting Time
P1              0               8               8               8               0
P2              0               13              21              21              8
P3              0               10              31              31              21
P4              0               12              43              43              31
P5              0               15              58              58              43
Average Turnaround Time: 32.20
Average Waiting Time: 20.60

Throughput: 0.09
Idle Time Percentage: 0.00%
```

SJF method

```c
#include <stdio.h>

typedef struct {
    int burst_time;
    int arrival_time;
    int completion_time;
    int turnaround_time;
    int waiting_time;
    char process_id[12];  // Increased size to 12 to handle larger process IDs like "P1000000000"
} Process;
void calculate_times(Process processes[], int n) {
    int start_time = 0;
    int total_turnaround_time = 0;
    int total_waiting_time = 0;

    // Sorting processes by arrival time
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].arrival_time > processes[j].arrival_time) {
                Process temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
    printf("\n%-12s%-16s%-16s%-20s%-20s%-20s\n", "Process ID", "Arrival Time", "Burst Time", "Completion Time", "Turnaround Time", "Waiting Time");

    // Calculate and print times for each process
    for (int i = 0; i < n; i++) {
        // If CPU is idle before the current process starts
        if (processes[i].arrival_time > start_time) {
            printf("Idle         %-16d%-16d%-20d%-20d%-20d\n", start_time, processes[i].arrival_time, 0, 0, 0);
            start_time = processes[i].arrival_time; // Set start time to process arrival time
        }
        processes[i].completion_time = start_time + processes[i].burst_time;
        processes[i].turnaround_time = processes[i].completion_time - processes[i].arrival_time;
        processes[i].waiting_time = processes[i].turnaround_time - processes[i].burst_time;
        total_turnaround_time += processes[i].turnaround_time;
        total_waiting_time += processes[i].waiting_time;

```

```
        // Print process information
        printf("%-12s%-16d%-16d%-20d%-20d%-20d\n", processes[i].process_id,
            processes[i].arrival_time, processes[i].burst_time, processes[i
    ].completion_time,
            processes[i].turnaround_time, processes[i].waiting_time);

        // Update start time for next process
        start_time = processes[i].completion_time;
    }
    double avg_turnaround_time = (double)total_turnaround_time / n;
    double avg_waiting_time = (double)total_waiting_time / n;

    printf("\nAverage Turnaround Time: %.2f\n", avg_turnaround_time);
    printf("Average Waiting Time: %.2f\n", avg_waiting_time);
    printf("\nGantt Chart:\n");
    printf("Time: ");
    for (int i = 0; i < n; i++) {
        printf("%-5d", processes[i].completion_time); // Print end time of
    each process
    }
    printf("\n");

    printf("Process ID: ");
    for (int i = 0; i < n; i++) {
        printf("%-5s", processes[i].process_id); // Print process ID
    }
    printf("\n");
}
idleness)
void print_metrics(Process processes[], int n) {
    int total_burst_time = 0;
    int total_idle_time = 0;
    int scheduling_length;
    // Calculate total burst time
    for (int i = 0; i < n; i++) {
        total_burst_time += processes[i].burst_time;
    }
    // Calculate total idle time
    for (int i = 1; i < n; i++) {
        // Calculate idle time between processes
        if (processes[i].arrival_time > processes[i - 1].completion_time) {
            total_idle_time += processes[i].arrival_time - processes[i -
    1].completion_time;
        }
    }

    // Scheduling length is the time taken from the first process arrival
    to the last process completion
    scheduling_length = processes[n - 1].completion_time - processes[0].
    arrival_time;
```

```
    double cpu_idleness = ((double)total_idle_time / scheduling_length) * 100;
    double throughput = (double)n / scheduling_length;

    // Print scheduling metrics
    printf("\nScheduling Metrics:\n");
    printf("Scheduling Length: %-10d\n", scheduling_length);
    printf("Throughput: %-10.2f processes/unit time\n", throughput);
    printf("CPU Idleness: %-10.2f%%\n", cpu_idleness);
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);
    Process processes[n];
    for (int i = 0; i < n; i++) {
        printf("Enter Arrival Time and Burst Time for Process P%d: ", i + 1);
        scanf("%d %d", &processes[i].arrival_time, &processes[i].burst_time);
        snprintf(processes[i].process_id, sizeof(processes[i].process_id), "P%d", i + 1);  // Assign process ID
    }
    calculate_times(processes, n);
    print_metrics(processes, n);

    return 0;
}
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc sjf.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
Enter the number of processes: 5
Enter Arrival Time and Burst Time for Process P1: 0 8
Enter Arrival Time and Burst Time for Process P2: 0 13
Enter Arrival Time and Burst Time for Process P3: 0 10
Enter Arrival Time and Burst Time for Process P4: 0 12
Enter Arrival Time and Burst Time for Process P5: 0 15

Process ID  Arrival Time    Burst Time      Completion Time     Turnaround Time     Waiting Time
P1          0               8               8                   8                   0
P2          0               13              21                  21                  8
P3          0               10              31                  31                  21
P4          0               12              43                  43                  31
P5          0               15              58                  58                  43

Average Turnaround Time: 32.20
Average Waiting Time: 20.60

Gantt Chart:
Time: 8    21   31   43   58
Process ID: P1   P2   P3   P4   P5

Scheduling Metrics:
Scheduling Length: 58
Throughput: 0.09       processes/unit time
CPU Idleness: 0.00      %
```

Round Robin method with the time quantum of 3 units.

```c
#include <stdio.h>

int main() {
    int n, cur_proc, total_time = 0, remaining_procs, quantum;
    int wait_time = 0, turnaround_time = 0;
    int at[10], bt[10], rt[10], ct[10];
    int idle = 0;

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

    remaining_procs = n;

    for (int i = 0; i < n; i++) {
        printf("Enter arrival time and burst time for process %d: ", i + 1);
        scanf("%d %d", &at[i], &bt[i]);
        rt[i] = bt[i];
    }

    printf("Enter time quantum: ");
    scanf("%d", &quantum);

    printf("\n\nProcess\t| Turnaround Time\t| Waiting Time\n\n");

    for (int time = 0, i = 0; remaining_procs != 0;) {
        if (rt[i] <= quantum && rt[i] > 0) {
            if (time < at[i]) {
                idle += at[i] - time;
                time = at[i];
            }
            time += rt[i];
            rt[i] = 0;
            ct[i] = time;

            remaining_procs--;
            printf("P[%d]\t|\t%d\t\t|\t%d\n", i + 1, ct[i] - at[i], ct[i] - at[i] - bt[i]);
            wait_time += ct[i] - at[i] - bt[i];
            turnaround_time += ct[i] - at[i];
        } else if (rt[i] > 0) {
            if (time < at[i]) {
                idle += at[i] - time;
                time = at[i];
            }
            rt[i] -= quantum;
            time += quantum;
        }
```

```
47
48          if (i == n - 1)
49              i = 0;
50          else if (at[i + 1] <= time)
51              i++;
52          else
53              i = 0;
54      }
55
56      int schedule_len = ct[0];
57      for (int i = 1; i < n; i++) {
58          if (ct[i] > schedule_len)
59              schedule_len = ct[i];
60      }
61      schedule_len -= at[0];
62
63      double throughput = (double)n / schedule_len;
64
65      printf("\nAverage Waiting Time = %.2f\n", (double)wait_time / n);
66      printf("Average Turnaround Time = %.2f\n", (double)turnaround_time / n)
        ;
67      printf("Scheduling Length = %d\n", schedule_len);
68      printf("Throughput = %.2f processes/unit time\n", throughput);
69      printf("Idle Time = %d\n", idle);
70
71      return 0;
72  }
```

Output

```
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ gcc rr.c
somnath@somnath-HP-Pavilion-Laptop-14-ec1xxx:~/Documents/oslab$ ./a.out
Enter Total Number of Processes:        5
Enter Arrival Time and Burst Time for Process 1: 0 8
Enter Arrival Time and Burst Time for Process 2: 0 13
Enter Arrival Time and Burst Time for Process 3: 0 10
Enter Arrival Time and Burst Time for Process 4: 0 12
Enter Arrival Time and Burst Time for Process 5: 0 15
Enter Time Quantum:     3


Process | Turnaround Time       | Waiting Time

P[1]    |       32              |       24
P[3]    |       48              |       38
P[4]    |       51              |       39
P[2]    |       55              |       42
P[5]    |       58              |       43

Average Waiting Time = 37.20
Average Turnaround Time = 48.80
Scheduling Length = 58
Throughput = 0.09 processes/unit time
Idle Time = 0
```