# NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

Operating System

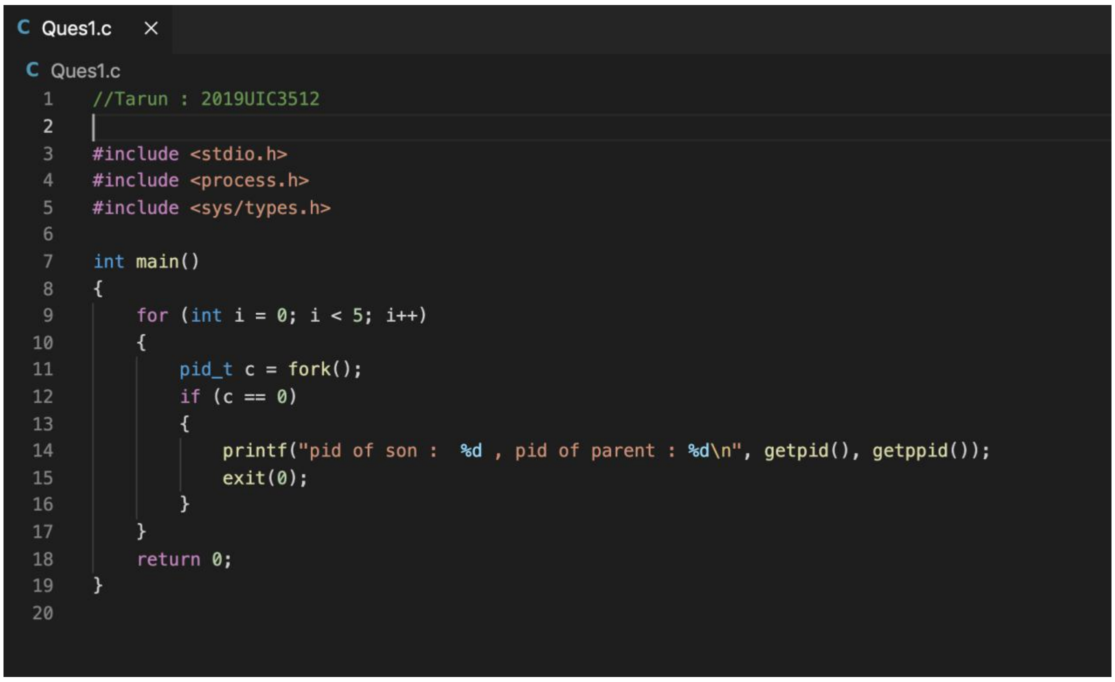
INSTRUMENTATION AND CONTROL ENGINEERING

# OS PROGRAMS

SUBMITTED BY:

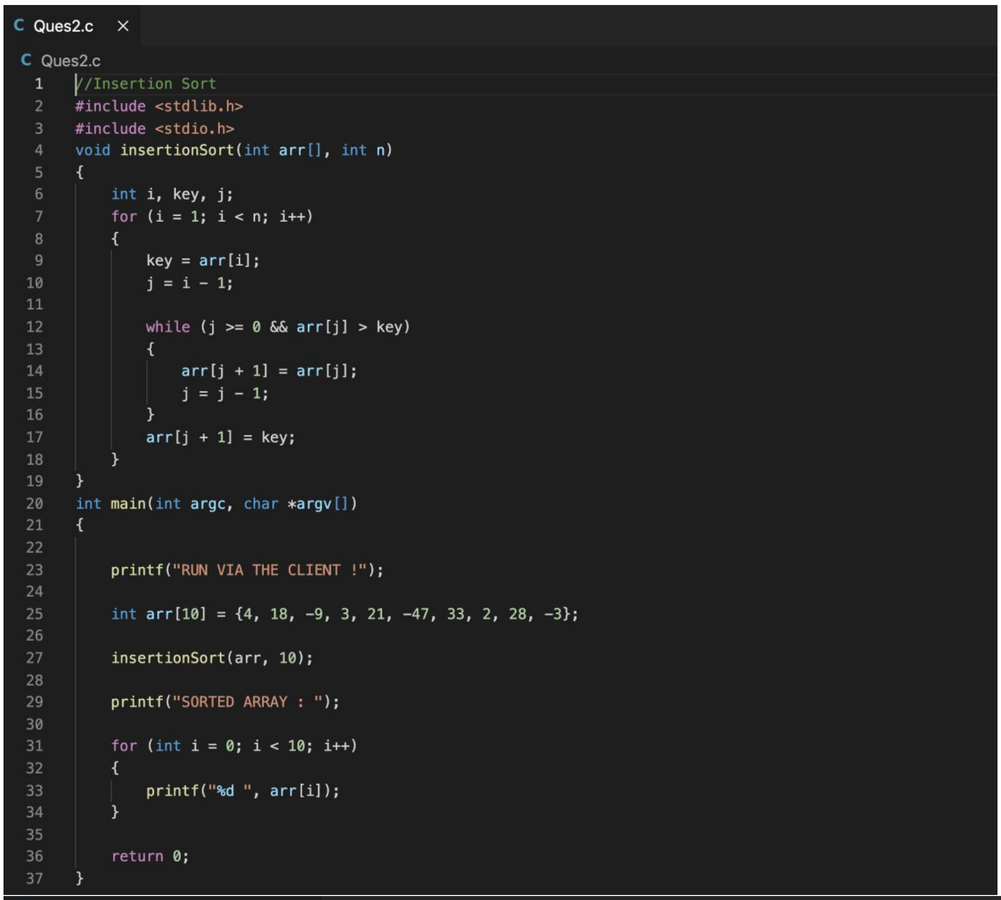
TARUN DHAWAN 2019UIC3512

## Ques 1)



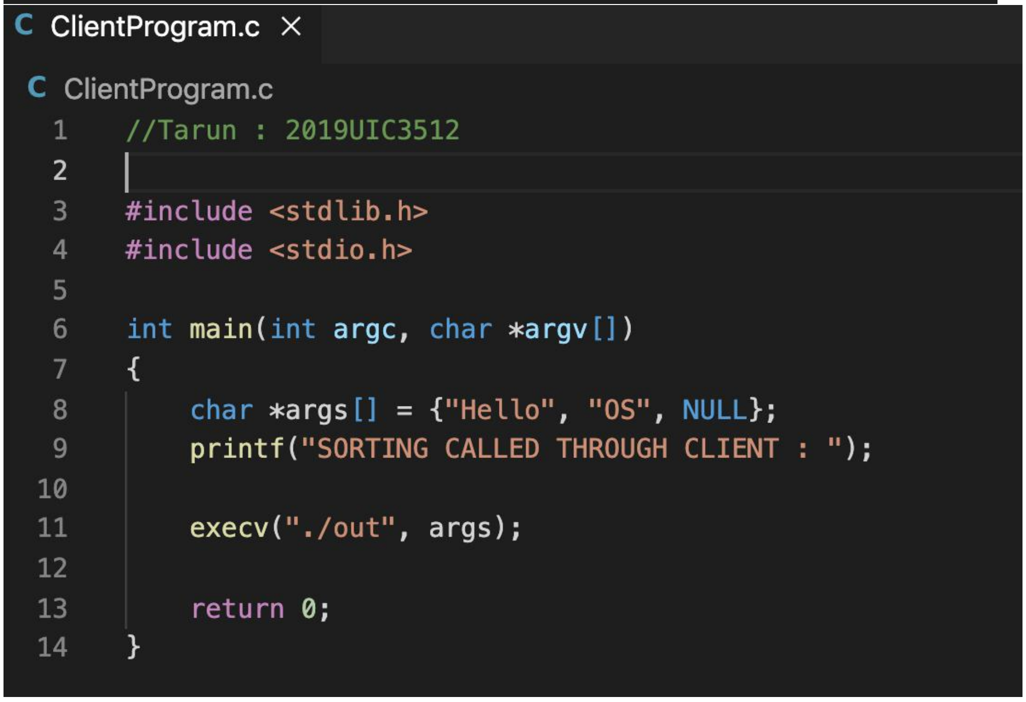
```c
//Tarun : 2019UIC3512

#include <stdio.h>
#include <process.h>
#include <sys/types.h>

int main()
{
    for (int i = 0; i < 5; i++)
    {
        pid_t c = fork();
        if (c == 0)
        {
            printf("pid of son :  %d , pid of parent : %d\n", getpid(), getppid());
            exit(0);
        }
    }
    return 0;
}
```

## Ques 2)



Ques2.c

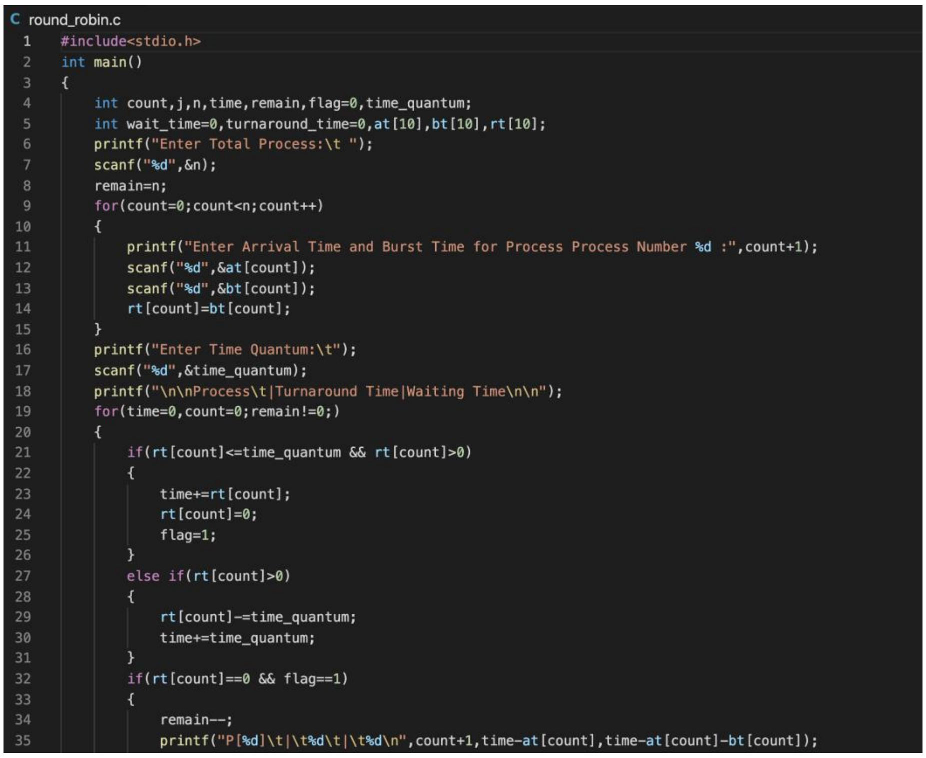
```c
//Insertion Sort
#include <stdlib.h>
#include <stdio.h>
void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
int main(int argc, char *argv[])
{

    printf("RUN VIA THE CLIENT !");

    int arr[10] = {4, 18, -9, 3, 21, -47, 33, 2, 28, -3};

    insertionSort(arr, 10);

    printf("SORTED ARRAY : ");

    for (int i = 0; i < 10; i++)
    {
        printf("%d ", arr[i]);
    }

    return 0;
}
```



ClientProgram.c

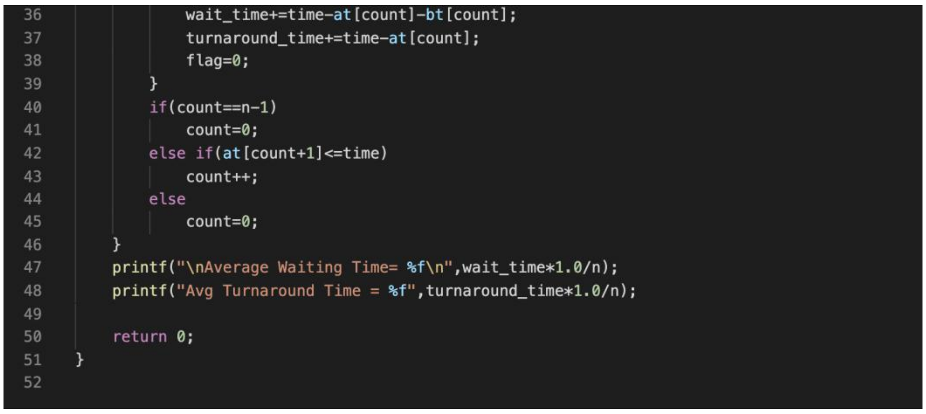
```c
//Tarun : 2019UIC3512

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

int main(int argc, char *argv[])
{
    char *args[] = {"Hello", "OS", NULL};
    printf("SORTING CALLED THROUGH CLIENT : ");

    execv("./out", args);

    return 0;
}
```

## Ques 3)



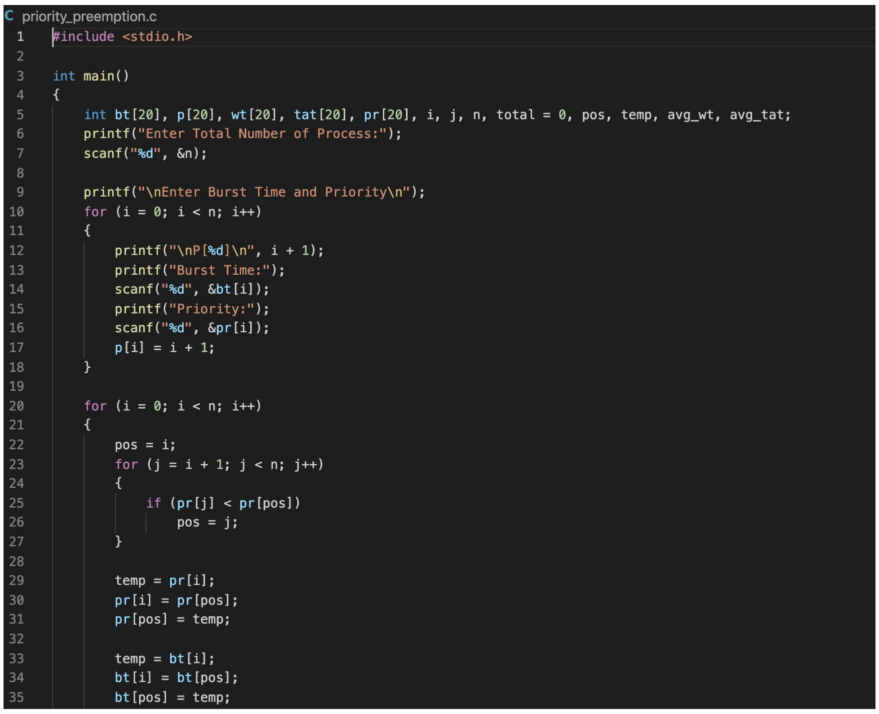
round_robin.c

```c
#include<stdio.h>
int main()
{
    int count,j,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf("Enter Total Process:\t ");
    scanf("%d",&n);
    remain=n;
    for(count=0;count<n;count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
    for(time=0,count=0;remain!=0;)
    {
        if(rt[count]<=time_quantum && rt[count]>0)
        {
            time+=rt[count];
            rt[count]=0;
            flag=1;
        }
        else if(rt[count]>0)
        {
            rt[count]-=time_quantum;
            time+=time_quantum;
        }
        if(rt[count]==0 && flag==1)
        {
            remain--;
            printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
```



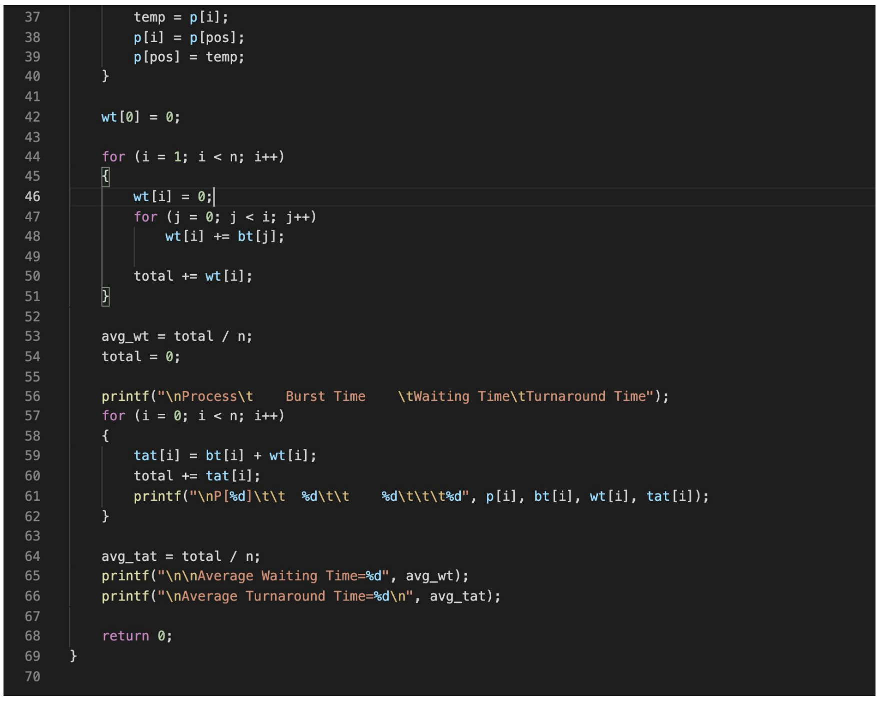
```c
            wait_time+=time-at[count]-bt[count];
            turnaround_time+=time-at[count];
            flag=0;
        }
        if(count==n-1)
            count=0;
        else if(at[count+1]<=time)
            count++;
        else
            count=0;
    }
    printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
    printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

    return 0;
}
```

## Ques 4)



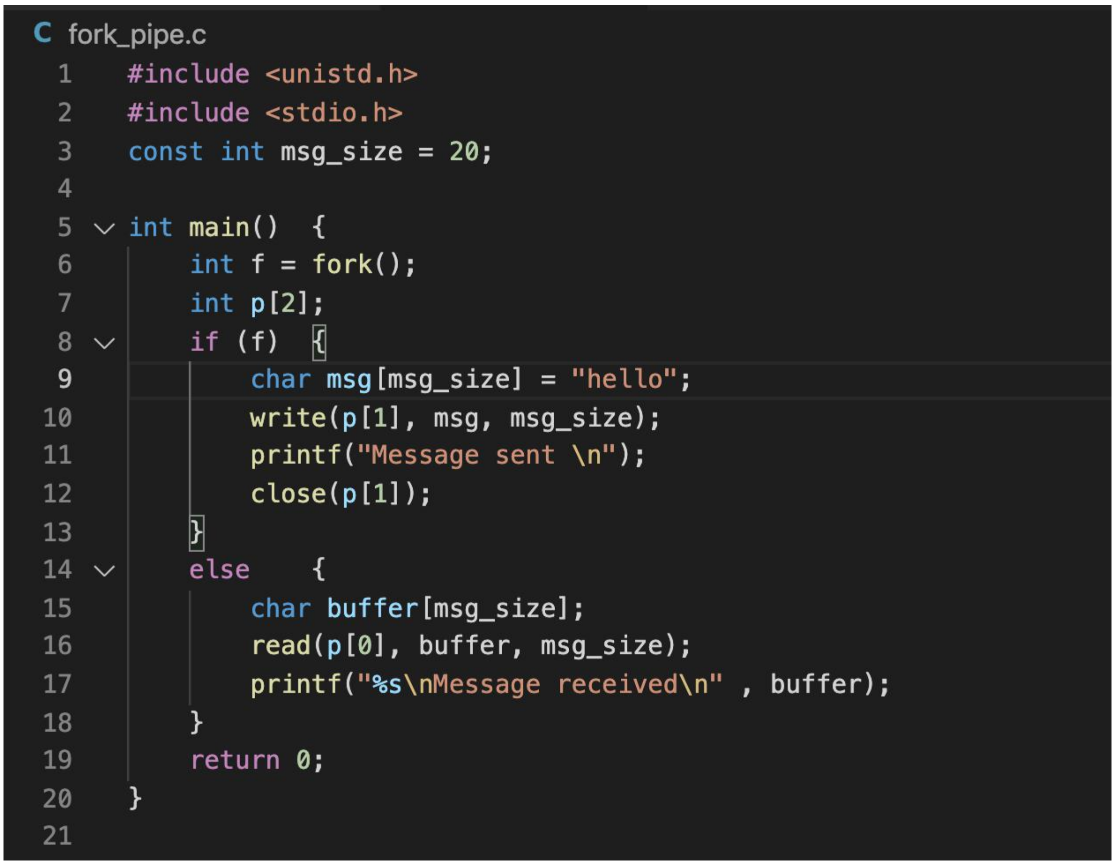
priority_preemption.c

```c
#include <stdio.h>

int main()
{
    int bt[20], p[20], wt[20], tat[20], pr[20], i, j, n, total = 0, pos, temp, avg_wt, avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d", &n);

    printf("\nEnter Burst Time and Priority\n");
    for (i = 0; i < n; i++)
    {
        printf("\nP[%d]\n", i + 1);
        printf("Burst Time:");
        scanf("%d", &bt[i]);
        printf("Priority:");
        scanf("%d", &pr[i]);
        p[i] = i + 1;
    }

    for (i = 0; i < n; i++)
    {
        pos = i;
        for (j = i + 1; j < n; j++)
        {
            if (pr[j] < pr[pos])
                pos = j;
        }

        temp = pr[i];
        pr[i] = pr[pos];
        pr[pos] = temp;

        temp = bt[i];
        bt[i] = bt[pos];
        bt[pos] = temp;
```



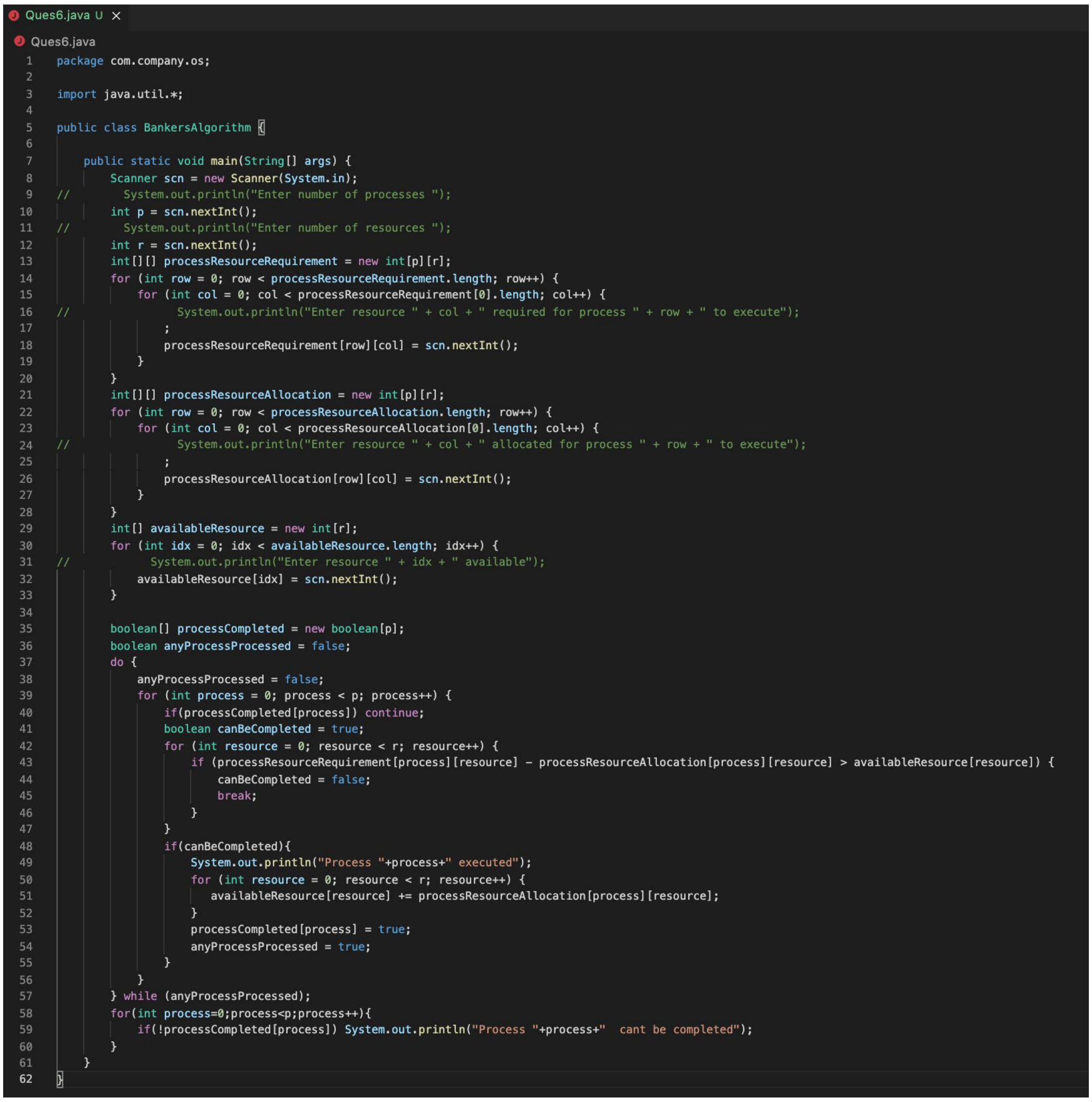
```c
        temp = p[i];
        p[i] = p[pos];
        p[pos] = temp;
    }

    wt[0] = 0;

    for (i = 1; i < n; i++)
    {
        wt[i] = 0;
        for (j = 0; j < i; j++)
            wt[i] += bt[j];

        total += wt[i];
    }

    avg_wt = total / n;
    total = 0;

    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for (i = 0; i < n; i++)
    {
        tat[i] = bt[i] + wt[i];
        total += tat[i];
        printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d", p[i], bt[i], wt[i], tat[i]);
    }

    avg_tat = total / n;
    printf("\n\nAverage Waiting Time=%d", avg_wt);
    printf("\nAverage Turnaround Time=%d\n", avg_tat);

    return 0;
}
```

## Ques 5)


fork_pipe.c

```c
#include <unistd.h>
#include <stdio.h>
const int msg_size = 20;

int main()  {
    int f = fork();
    int p[2];
    if (f)  {
        char msg[msg_size] = "hello";
        write(p[1], msg, msg_size);
        printf("Message sent \n");
        close(p[1]);
    }
    else    {
        char buffer[msg_size];
        read(p[0], buffer, msg_size);
        printf("%s\nMessage received\n" , buffer);
    }
    return 0;
}
```

## Ques 6)



```java
package com.company.os;

import java.util.*;

public class BankersAlgorithm {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
//        System.out.println("Enter number of processes ");
        int p = scn.nextInt();
//        System.out.println("Enter number of resources ");
        int r = scn.nextInt();
        int[][] processResourceRequirement = new int[p][r];
        for (int row = 0; row < processResourceRequirement.length; row++) {
            for (int col = 0; col < processResourceRequirement[0].length; col++) {
//                System.out.println("Enter resource " + col + " required for process " + row + " to execute");
                ;
                processResourceRequirement[row][col] = scn.nextInt();
            }
        }
        int[][] processResourceAllocation = new int[p][r];
        for (int row = 0; row < processResourceAllocation.length; row++) {
            for (int col = 0; col < processResourceAllocation[0].length; col++) {
//                System.out.println("Enter resource " + col + " allocated for process " + row + " to execute");
                ;
                processResourceAllocation[row][col] = scn.nextInt();
            }
        }
        int[] availableResource = new int[r];
        for (int idx = 0; idx < availableResource.length; idx++) {
//            System.out.println("Enter resource " + idx + " available");
            availableResource[idx] = scn.nextInt();
        }

        boolean[] processCompleted = new boolean[p];
        boolean anyProcessProcessed = false;
        do {
            anyProcessProcessed = false;
            for (int process = 0; process < p; process++) {
                if(processCompleted[process]) continue;
                boolean canBeCompleted = true;
                for (int resource = 0; resource < r; resource++) {
                    if (processResourceRequirement[process][resource] - processResourceAllocation[process][resource] > availableResource[resource]) {
                        canBeCompleted = false;
                        break;
                    }
                }
                if(canBeCompleted){
                    System.out.println("Process "+process+" executed");
                    for (int resource = 0; resource < r; resource++) {
                       availableResource[resource] += processResourceAllocation[process][resource];
                    }
                    processCompleted[process] = true;
                    anyProcessProcessed = true;
                }
            }
        } while (anyProcessProcessed);
        for(int process=0;process<p;process++){
            if(!processCompleted[process]) System.out.println("Process "+process+"  cant be completed");
        }
    }
}
```