Question 1: (15 marks)

1. A company decided to give a bonus of 5% to an employee if his/her year of service is more than 5 years. Write a shell script to ask users for their salary and year of service and print the net bonus amount in an information dialogue box.

```
#! /bin/sh/

echo "Enter your salary: "
read salary
echo "Enter your year of service: "
read year

if [ $year -gt 5 ]
then
  echo "Employee will get (`expr $salary \* 5`)"
else
  echo "no bounces"
if
```

2. Write a Program for process synchronization using locks. Program tos create two threads: one to increment the value of a shared variable and second to decrement the value of shared variable. Both the threads make use of locks so that only one of the threads is executing in its critical section.

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

void *fun1();
void *fun2();
int shared = 1;
pthread_mutex_t l;

int main()
{

  pthread_mutex_init(&l, NULL);
  pthread_t thread1, thread2;
  pthread_create(&thread1, NULL, fun1, NULL);
  pthread_create(&thread2, NULL, fun2, NULL);

  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);

  printf("Final value of shared is %d\n", shared);
```

```c
}

void *fun1()
{

  int x;
  printf("Thread1 trying to acquire lock\n");
  pthread_mutex_lock(&l);

  printf("Thread1 acquired lock\n");
  x = shared;

  printf("Thread1 reads the value of shared variable as %d\n", x);
  x++;

  printf("Local updation by Thread1: %d\n", x);
  sleep(1);

  shared = x;
  printf("Value of shared variable updated by Thread1 is: %d\n", shared);

  pthread_mutex_unlock(&l);
  printf("Thread1 released the lock\n");
}

void *fun2()
{

  int y;
  printf("Thread2 trying to acquire lock\n");

  pthread_mutex_lock(&l);
  printf("Thread2 acquired lock\n");

  y = shared;
  printf("Thread2 reads the value as %d\n", y);

  y--;
  printf("Local updation by Thread2: %d\n", y);

  sleep(1);
  shared = y;

  printf("Value of shared variable updated by Thread2 is: %d\n", shared);
  pthread_mutex_unlock(&l);
```

```
  printf("Thread2 released the lock\n");
}
```

Question 2: (15 marks)

1. Write a program of zombie process in e programming.

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

int main(){
  pid_t t;
  t = fork();
  if (t == 0){
    printf("Child having id %d\n", getpid());
  }
  else{
    sleep(15);
    printf("Parent having id %d\n", getpid());
  }
}
```

2. Write a C program that takes an array of integers as input. Create a child process (Let's say C1) that checks whether the elements of the array are even or odd. It prints the total even numbers present in the array and if there are any odd numbers present then it multiplies it with 2. Create another child process of C1 (which we call C2) that again checks whether there are any odd numbers present in the array, if there is no odd number it prints the message "No odd Numbers". After that the parent process calculates how many odd numbers were presented in the array after input and before creating any child. There shouldn't be any zombie or orphan process as a result of execution.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main()
{
  int n = 0, odd, even = 0;
  pid_t q, p;
  printf("Enter Limit of Numbers:");
  scanf("%d", &n);
  int array[n];
  for (int i = 0; i < n; i++)
```

```
{
  array[i] = 0;
  printf("Enter Numbers: ");
  scanf("%d", &array[i]);
}
q = fork();
if (q == 0)
{
  printf("odd of child1: %d", odd);
  printf("Child 1\n");
  printf("Child 1 id : %d\n", getpid());
  printf("parent of child 1 is %d\n", getppid());
  printf("%d\n", q);
  for (int i = 0; i < n; i++)
  {
    if (array[i] % 2 == 0)
    {
      even++;
    }
    else
    {
      array[i] *= 2;
    }
    printf("array *2 = %d\n", array[i]);
  }
  printf("even : %d\n", even);
  for (int i = 0; i < n; i++)
  {
    printf("array %d\n", array[i]);
  }
}
else
{
  p = fork();
  if (p == 0)
  {
    printf("odd of child2: %d", odd);
    printf("Child 2\n");
    printf("Child 2 id : %d\n", getpid());
    printf("parent of child 2 is %d\n", getppid());
    printf("%d\n", q);
    if (odd == 0)
    {
      printf("NO odd number");
    }
```

```
      else if (odd > 0)
      {
        printf("Some Odd numbers");
      }
      for (int i = 0; i < n; i++)
      {
        printf("array %d\n", array[i]);
      }
    }
    else
    {
      printf("parent is %d\n", getpid());
      for (int i = 0; i < n; i++)
      {
        if (array[i] % 2 != 0)
        {
          odd++;
        }
      }
      printf("odd : %d\n", odd);
    }
  }
}
```

Question 3: (5 marks)

Write a program that take 3 processes as input, the burst time of that processes and apply FCFS algorithm (without arrival time) on to the processes and calculate their waiting time and turnaround time?

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



void findWaitingTime(int processes[], int n,
                          int bt[], int wt[]){
    wt[0] = 0;
    for (int  i = 1; i < n ; i++ )
        wt[i] =  bt[i-1] + wt[i-1] ;
}
```

```
void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[]){
    for (int  i = 0; i < n ; i++)
        tat[i] = bt[i] + wt[i];
}

void findavgTime( int processes[], int n, int bt[]){
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    findWaitingTime(processes, n, bt, wt);
    findTurnAroundTime(processes, n, bt, wt, tat);

    printf("Processes   Burst time   Waiting time   Turn around time\n");
    for (int  i=0; i<n; i++){
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];

        printf("   %d\t\t%d\t\t%d\t\t%d\n",processes[i], bt[i] ,wt[i] ,tat[i] );
    }
    int s=(float)total_wt / (float)n;
    int t=(float)total_tat / (float)n;

    printf("\n\nAverage waiting time. \t (%d / %d) = %d",total_wt, n, s);
    printf("\nAverage turn around time. (%d / %d) = %d",total_tat, n,t);
}

int FCFS(){
    int n = getNumber(15);

    int processes[n], burst_time[n];

    for(int i = 0; i < n; i++){
        processes[i] = i;
        burst_time[i] = getNumber(10);
    }

    printf("\n\n\t\tFirst Come First Serve\n");


    findavgTime(processes, n,  burst_time);
    return 0;
}
```

Question 4: (15 marks)

1. Write programs to create named pipe and send message "You are expert in linux from one process to the other process".

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

int main(){
        int res,n;
        res=open("fifo1",O_WRONLY);
        write(res,"Message",7);
        printf("Sender Process %d sent the data\n",getpid());

}
```

```
#include<unistd.h>

#include<stdio.h>

#include<fcntl.h>

int main(){

int res,n;

char buffer[100];

res=open("fifo1",O_RDONLY);

n=read(res,buffer,100);

printf("Reader process %d started\n",getpid());

printf("Data received by receiver %d is: %s\n",getpid(), buffer);

}
```

2. Write a program to create 2 threads in Linux Thrend I prims the son of starting five ven numbers while the second thread take 5 numbers as input and sort them and the main process prints numbers from 20-24.

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

void *thread_function(void *arg);
int i, j;
int main()
```

```
{
  pthread_t a_thread;
  pthread_create(&a_thread, NULL, thread_function, NULL);
  pthread_join(a_thread, NULL);
  printf("Inside Main Program\n");
  for (j = 20; j < 25; j++)
  {
    printf("%d\n", j);
    sleep(1);
  }
}
void *thread_function(void *arg){
  printf("Inside Thread\n");
  for (i = 0; i < 5; i++)
  {
    printf("%d\n", i);
    sleep(1);
  }
}

void *thread_function1(void *arg){
  printf("Inside Thread\n");
  for (i = 0; i < 5; i+2){
    printf("%d\n", i);
    sleep(1);
  }
}
```

Question 1: (15 marks).

1. Write a shell script to take 5 numbers as input and sort the numbers.

```
#! /bin/sh/

echo "Enter your salary: "
read salary
echo "Enter your year of service: "
read year

if [ $year -gt 5 ]
then
  echo "Employee will get (`expr $salary \* 5`)"
else
  echo "no bounces"
if
```

2. 2. Write a program to achieve synchronization between multiple threads. to acquire a resource that has two instances.

Question 2: (15 marks)

1. Create a parent-child relationship between two processes. The parent should print two statements:

A) Parent (P) is having ID <PID>

B) ID of P's Child is <PID_of Child>

C) Child is having ID <PID>

D) My Parent ID is <PID of Parent>

Make use of wait() in such a manner that the order of the four statements A, B, C and D is: ACDB

You are free to use any other relevant statement/printf as you desire execution does not matter. Write hierarchy of 3 process such that 2. and their order of a program using fork() system call to create a P2 is the child of P1 and P1 is the child of P Print their ID's.

```
#include<stdio.h
#include<unistd.h>
#include<sys/types.h>
int main()
{

pid_t p;
p=fork();
```

```
if(p==0) //child
{
printf("I am child having PID %d\n",getpid());
printf("My parent PID is %d\n",getppid());
}
else //parent
{
printf("I am parent having PID %d\n",getpid());
printf("My child PID is %d\n",p);
}
}
```

Question 3: (5 marks)

Write a program that take 3 processes as input, the burst time of that processes and apply SJC algorithm (without arrival time) on to the processes and calculate their waiting time and turnaround time?

Question 4: (15 marks)

1. Write a program to create shared memory and send message "You are expert in Linux" from one process to the other process.

```
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main(){
int res,n;
res=open("fifo1",O_WRONLY);
write(res,"Message",7);
printf("Sender Process %d sent the data\n",getpid());

}

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

```
#include<fcntl.h>
int main(){
int res,n;
char buffer[100];
res=open("fifo1",O_RDONLY);
n=read(res,buffer,100);
printf("Reader process %d started\n",getpid());
printf("Data received by receiver %d is: %s\n",getpid(), buffer);
}
```

2. Write a program to create 2 threads in Linux. Thread 1 prints the sum of starting five odd numbers while the second thread take 5 numbers as input and sort them in descending order and the main process prints numbers from table of 5.

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

void *thread_function(void *arg);
int i, j;
int main()
{
  pthread_t a_thread;
  pthread_create(&a_thread, NULL, thread_function, NULL);
  pthread_join(a_thread, NULL);
  printf("Inside Main Program\n");
  for (j = 20; j < 25; j++)
  {
    printf("%d\n", j);
    sleep(1);
  }
}
void *thread_function(void *arg){
  printf("Inside Thread\n");
  for (i = 0; i < 5; i++)
  {
    printf("%d\n", i);
    sleep(1);
  }
}

void *thread_function1(void *arg){
  printf("Inside Thread\n");
  for (i = 0; i < 5; i+2){
    printf("%d\n", i);
    sleep(1);
  }
}
```

RR

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



int RR(){
    int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0;

    float avg_wt, avg_tat;
    printf("Total number of process in the system: ");
    y = NOP = getNumber(8);
    printf("%d \n",y);

    int at[y], bt[y], temp[y];



    for(i=0; i<NOP; i++){
        at[i] = getNumber(3);
        bt[i] = getNumber(6);
        temp[i] = bt[i];
    }

    printf("Enter the Time Quantum for the process: \t");
    scanf("%d", &quant);

    printf("\n| Process No | Arrial Time | Burst Time |\t TAT \t|  Waiting Time
|");

    for(sum=0, i = 0; y!=0; ){
        if(temp[i] <= quant && temp[i] > 0){
        sum = sum + temp[i];
        temp[i] = 0;
        count=1;
        }
        else if(temp[i] > 0){
            temp[i] = temp[i] - quant;
            sum = sum + quant;
        }

        if(temp[i]==0 && count==1){
            y--;
```

```
            printf("\n|      %d      |      %d      |      %d      |\t%d\t|\t%d\t|"
, (i+1),at[i], bt[i], sum-at[i], (bt[i]-sum-at[i]));
            wt = wt+sum-at[i]-bt[i];
            tat = tat+sum-at[i];
            count =0;
        }
        if(i==NOP-1){
            i=0;
        }
        else if(at[i+1]<=sum){
            i++;
        }
        else{
            i=0;
        }
    }

    avg_wt = wt * 1.0/NOP;
    avg_tat = tat * 1.0/NOP;
    printf("\n\nAverage Turn Around Time: \t%f", avg_wt);
    printf("\nAverage Waiting Time: \t%f", avg_tat);
    getch();
}
```