Q1. Write a program (using fork() and/or exec() commands) where parent and child execute: (a) same program, same code. (b) same program, different code.

c) before terminating, the parent waits for the child to finish its task

code:-

(a)

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

int main() {
pid_t pid,pr;
pr=fork();
pid=getpid();
if(pr<0)
{
cout<<"Failed!!!\n";
return 1;
}
cout<<"The output of Fork="<<pr<<end1;
cout<<"Process Id= "<<pid<<endl;
return 0;
}
```

(b)

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
using namespace std;

int main(){
    int pid =fork();
    if (pid<0)
    {
        cout<<"Failed"<<endl;
        exit(1);
    }
    else if(pid==0)
```



```
        if(pid==0)
    {
        cout<<"Child Process"<<endl;
        cout<<"Child Process ID = "<<getpid()<<endl;
        exit(0);
    }
    else
    {
        cout<<"Parent Process "<<endl;
        cout<<"Parent Process ID = "<<getpid()<<endl;
        exit(1);
    } }
```

(c)

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
using namespace std;

int main() {
    int pid =fork();
    if (pid<0)
    {
        cout<<"Failed"<<endl;
        exit(1);
    }
    else if(pid==0)
    {
        cout<<"Child Process"<<endl;
        cout<<"Child Process ID = "<<getpid()<<endl;
        exit(0);
    }
    else
    {
        cout<<"Parent Process "<<endl;
        cout<<"Parent Process ID = "<<getpid()<<endl;
        wait(NULL);
        exit(1);
    } }
```


Q2. Write a program to report behavior of Linux kernel including kernel version, CPU type and model. (CPU information)

code:-

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
using namespace std;

int main()
{
    cout<<"The Kernal Version:-\n";
    system("cat/proc/sys/kernel/osrelease");
    cout<<"\nThe CPU info:-\n";
    system("cat/proc/cpuinfo |awk 'NR==3,NR==4{print}'\n");
    return 0;
}
```


Q3. Write a program to report behavior of Linux kernel including information on 19 configured memory, amount of free and used memory. (memory information)

code:-

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
using namespace std;

int main()
{
cout<<"\n The Kernel Version is : \n";
system("cat /proc/sys/kernel/osrelease");
cout<<"\n The CPU Space : \n";
system("cat /proc/cpuinfo | awk 'NR==3, NR==4{print}'\n");
cout<<"\n Amount of CPU time since system was last booted is : ";
system("cat /proc/uptime \n");
cout<<"\n The configured memory is :\n ";
system("cat /proc/meminfo | awk 'NR == 1{print $2}'\n");
cout<<"\n Amount of free memory :\n ";
system("cat /proc/meminfo | awk 'NR == 2{print $2}'\n");
cout<<"\n Amount of used memory is :\n ";
system("cat /proc/meminfo | awk '{if (NR==1) a=$2; if(NR==2) b=$2
}END {print a-b}' \n");
cout<<endl;
return 0;

}
```


4. Write a program to print file details including owner access permissions, file access time, where file name is given as argument.

Code:-

```
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
using namespace std;

int main(int argc,char*argv[])
{
struct stat s;
if(argc<2){
cout<<"\nEnter the filename:-\n";
}
for(int i=1;i<argc;i++)
{
cout<<"File:- "<<argv[i]<<"\n";

if(stat(argv[i],&s)<0)
{
cout<<"Error";
}
else{
cout<<"Owner UID:- "<<s.st_uid<<"\n";
cout<<"Group UID:- "<<s.st_gid<<"\n";
cout<<"Permission:- "<<s.st_mode<<"\n";
cout<<"Access Time:- "<<s.st_atime<<"\n";
cout<<"Size:- "<<s.st_size<<endl;
}
}
return 0;
}
```


5. Write a program to copy files using system calls.

Code:-

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

int main()
{
    int n,fd;
    char buff[50];  // declaring buffer

    //message printing on the display
    printf("Enter text to write in the file:\n");
    //read from keyboard, specifying 0 as fd for std input device
    //Here, n stores the number of characters
    n= read(0, buff, 50);

    // creating a new file using open.
    fd=open("file",O_CREAT | O_RDWR, 0777);

    //writting input data to file (fd)
    write(fd, buff, n);
    //Write to display (1 is standard fd for output device)
    write(1, buff, n);

    //closing the file
    int close(int fd);

    return 0;
}
```

6. Write a program to implement FCFS scheduling algorithm.

Code:-

```
#include <iostream>
using namespace std;

int main()
{
cout<<"--FCFS Scheduling Algorithm--\n";
int turn =0, n,wait,r=0,s=0,w=0,t=0;
cout<<"\nEnter the no. of Processor:- ";
cin>>n;
cout<<endl;
int b[n];
for(int i=0;i<n;i++)
{
cout<<"\nBurst Time for Process P"<<i+1<<" : ";
cin>>b[i];
}
cout<<"\nWaiting time for Process P1 = 0 ms";

for(int i=1;i<n;i++)
{
r=r+b[i-1];
cout<<"\nWaiting time for Process P"<<i+1<<" = "<<r<<"ms";
t=t+r;
}
for(int i=0;i<n;i++)
{
s=s+b[i];
cout<<"\nTurnaround Time for Process P"<<i+1<<" = "<<s<<"ms";
w=w+s;
}
cout<<"\nAverage Turnaround time :-"<<w/n<<"ms";
cout<<"\nAverage Waiting time :-"<<t/n<<"ms";
}
```


7. Write a program to implement Round Robinscheduling algorithm.

Code:-

```
#include <iostream>
using namespace std;

void Wait(int p[],int n,int b[], int w[], int q)
{
int bt[n];
for (int i = 0; i<n; i++)
bt[i] = b[i];
int t = 0;
while(1)
{
bool done = true;
for (int i=0;i<n;i++)
{
if (bt[i]>0)
{
done=false;

if (bt[i]>q)
{
t += q;
bt[i] -= q;
}
else
{
t=t+bt[i];
w[i]=t-b[i];
bt[i]=0;
}
} }
if (done == true)
break;
} }
void Turn(int p[],int n,int b[],int w[],int t[])
{
for (int i=0;i<n;i++)
t[i] = b[i]+w[i];

}
```



```
void avg(int p[],int n,int b[],int q)
{
int w[n],t[n],total_w=0,total_t=0;
Wait(p,n,b,w,q);
Turn(p,n,b,w,t);
cout<<" Processes "<<" Burst time "<<" Waiting time "<<" Turn
around time\n";
for (int i=0;i<n;i++)
{
total_w=total_w+w[i];
total_t=total_t+t[i];
cout<<" "<<i+1<<"\t\t"<<b[i]<<"\t "<<w[i]<<"\t\t "<<t[i]<<endl;
}
cout<<"Average waiting time = "<<(float)total_w/(float)n;
cout<<"\nAverage turn around time = "<<(float)total_t/(float)n;
}

int main()
{
cout<<"--Round Robin Scheduling Algorithm--\n";
int n;
cout<<"\nEnter the no. of processes:- ";
cin>>n;
int p[n];
for (int i=0;i<n;i++)
{
p[i]=i+1;
}
int b[n];
for (int i=0;i<n;i++)
{
cout<<"\nEnter the Burst Time for P"<<i+1<<":- ";
cin>>b[i];
}
int q;
cout<<"\nEnter the Time Quantum:- ";
cin>>q;
avg(p,n,b,q);
return 0;
}
```


8. Write a program to implement SJF schedulingalgorithm.

Code:-

```
##include <iostream>
using namespace std;

int main()
{
cout<<"--SJF Scheduling Algorithm--\n";
int n,tmp,tt=0,min,d,i,j;
float at=0,aw=0,st=0,sw=0;

cout<<"\nEnter no. of processes:- ";
cin>>n;
int a[n],b[n],e[n],t[n],w[n];
for (i=0;i<n;i++)
{
cout<<"Arrival time of P"<<i+1<<":- ";
cin>>a[i];
}

for (i=0;i<n;i++)
{
cout<<"Enter Burst time of P"<<i+1<<":- ";
cin>>b[i];
}
for (i=0;i<n;i++)
{
for (j=i+1;j<n;j++)
{
if (b[i]>b[j])
{
swap(a[i],a[j]);
swap(b[i],b[j]);
}
}
}
min=a[0];
for (i=0;i<n;i++)
{
if (min>a[i])
{
```



```
min=a[i];

d=i;
}
}
tt=min;
e[d]=tt+b[d];
tt=e[d];

for (i=0;i<n;i++)
{
if (a[i]!=min)
{
e[i]=b[i]+tt;
tt=e[i];
}
}
for (i=0;i<n;i++)
{
t[i]=e[i]-a[i];
st=st+t[i];
w[i]=t[i]-b[i];
sw=sw+w[i];
}

at=st/n;
aw=sw/n;
cout<<"Process Arrival-time(ms) Burst-time(ms) Waiting-time(ms)
Turnaround-time(ms)\n";

for(i=0;i<n;i++)
{
cout<<"P"<<i+1<<" "<<a[i]<<" "<<b[i]<<"
"<<w[i]<<" "<<t[i]<<endl;
}
cout<<"\nAverage Waiting Time= "<<aw<<"\nAverage Turnaround Time=
"<<at;
}
```


9. Write a program to implement non-preemptive priority based scheduling algorithm.

Code:-

```
#include <iostream>
using namespace std;

int main()
{
cout<<"--Non-Preemptive Priority Based Scheduling Algorithm--\n";
int b[20],p[20],w[20],t[20],pr[20],i,j,n,total=0,pos,aw,at;
cout<<"\nEnter Number of Processes: ";
cin>>n;
cout<<"\nEnter Burst Time andPriority:- \n";
for (int i=0;i<n;i++)
{
p[i]=i+1;
}
for (i=0;i<n;i++)
{

cout<<"Burst Time of P"<<i+1<<":- ";
cin>>b[i];
}
for (i=0;i<n;i++)
{
cout<<"Priority of P"<<i+1<<":- ";
cin>>pr[i];
}
for (i=0;i<n;i++)
{
pos=i;
for (j=i+1;j<n;j++)
{
if (pr[j]<pr[pos])
pos=j;
}
swap(pr[i],pr[pos]);
swap(b[i],b[pos]);
swap(p[i],p[pos]);
}
w[0]=0;
for (i=1;i<n;i++)
```

```
{
w[i]=0;
for (j=0;j<n;j++)
{
w[i] += b[j];
total += w[i];
}
}
aw=total/n;
total=0;
cout<<"\nProcess\t Burst Time \tWaiting Time\tTurnaround Time";
for (i=0;i<n;i++)
{
t[i]=b[i]+w[i];
total +=t[i];
cout<<"\nP["<<p[i]<<"]\t\t "<<b[i]<<"\t\t
"<<w[i]<<"\t\t\t"<<t[i];
}
at=total/n;
cout<<"\n\nAverage Waiting Time= "<<aw;
cout<<"\nAverage Turnaround Time="<<at;
return 0;
```


10. Write a program to implement preemptive priority based scheduling algorithm.

Code:-

```
#include <iostream>
using namespace std;

int main()
{
cout <<"--Preemptive Priority Based Scheduling Algorithm--\n";
int a[10],b[10],x[10];
int w[10],t[10],c[10],p[10];
int i,j,min,count=0,time,n;
float at=0,aw=0,end;
cout<<"\nEnter the number of Processes:- ";
cin>>n;
for (i=0;i<n;i++)
{
cout<<"\nArrival time of P"<<i+1<<":- ";
cin>>a[i];
}

for (i=0;i<n;i++)
{
cout<<"\nBurst time of P"<<i+1<<":- ";
cin>>b[i];
}
for (i=0;i<n;i++)
{
cout<<"\nPriority time of P"<<i+1<<":- ";
cin>>p[i];
}
for (i=0;i<n;i++)
{
x[i]=b[i];
}
p[9]=-1;
for (time=0;count!=n;time++)
{
min=9;
for(i=0;i<n;i++)
{
if (a[i]<=time&&p[i]>p[min]&&b[i]>0)
```



```
min=i;

}
b[min]--;
if (b[min]==0)
{ count++;
end=time+1;
c[min] = end;
w[min] = end-a[min]-x[min];
t[min] = end-a[min];
}
}

cout<<"Process"<<"\t"<<"Burst-time"<<"\t"<<"Arrival-
time"<<"\t"<<"Waiting-time"<<"\t"<<"Turnaround-
time"<<"\t"<<"Completion-time"<<"\t"<<"Priority-time"<<endl;

for (i=0;i<n;i++)
{
cout<<"P"<<i+1<<"\t\t"<<x[i]<<"\t\t"<<a[i]<<"\t\t"<<w[i]<<"\t\t"<
<t[i]<
<"\t\t"<<c[i]<<"\t\t"<<p[i]<<endl;
aw=aw+w[i];
at=at+t[i];
}
cout<<"\nAverage Waiting time= "<<aw/n;
cout<<"\nAverage Turnaround time= "<<at/n<<endl;
}
```


11. Write a program to implement SRJF scheduling algorithm.

Code:-

```
#include <iostream>
using namespace std;

int main() {
cout <<"--SRJF Scheduling Algorithm--\n"<< endl;
int a[10],b[10],x[10],i,j,small,c=0,time,n;
float avg=0,tt=0,end;
cout<<"Enter the number of Processes:- \n";
cin>>n;
cout<<"Enter arrival time:- \n";
for (i=0;i<n;i++) {
cin>>a[i];
}
cout<<"Enter burst time:- \n";
for (i=0;i<n;i++) {
cin>>b[i];
}
for (i=0;i<n;i++)
{
x[i]=b[i];
}
b[9]=9999;
for (time=0;c!=n;time++) {
small=9;
for (i=0;i<n;i++) {
if (a[i]<=time&&b[i]<b[small]&&b[i]>0)
small=i;
}
b[small]--;
if (b[small]==0) {
c++;
end=time+1;
avg=avg+end-a[small]-x[small];
tt=tt+end-a[small];
} }
cout<<"\nAverage Waiting time= "<<avg/n<<"\n";
cout<<"\nAverage Turnaround time= "<<tt/n<<"\n";
return 0;
}
```


12. Write a program to calculate sum of n numbers using thread library.

Code:-

```
#include <pthread.h>
#include<bits/stdc++.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct data{
int* arr;
int thread_num;
} data;

int arrSize = 10;

void* halfSum(void* p){
data* ptr = (data*)p;
int n = ptr->thread_num;

// Declare sum dynamically to return to join:
int* thread_sum = (int*) calloc(1, sizeof(int));

if(n == 0){
for(int i = 0; i < arrSize/2; i++)
thread_sum[0] = thread_sum[0] + ptr->arr[i];
}
else{
for(int i = arrSize/2; i < arrSize; i++)
thread_sum[0] = thread_sum[0] + ptr->arr[i];
}

pthread_exit(thread_sum);
}

int main(void){

printf("Array :- [1,2,3,4,5,6,7,8,9,10]\n");
int* int_arr = (int*) calloc(arrSize, sizeof(int));
for(int i = 0; i < arrSize; i++)
int_arr[i] = i + 1;

// Declare arguments for both threads:
```



```
data thread_data[2];

thread_data[0].thread_num = 0;
thread_data[0].arr = int_arr;
thread_data[1].thread_num = 1;
thread_data[1].arr = int_arr;

// Declare thread IDs:
pthread_t tid[2];

// Start both threads:
pthread_create(&tid[0], NULL, halfSum, &thread_data[0]);
pthread_create(&tid[1], NULL, halfSum, &thread_data[1]);
// Declare space for sum:
int* sum0;
int* sum1;
// Retrieve sum of threads:
pthread_join(tid[0], (void**)&sum0);
pthread_join(tid[1], (void**)&sum1);

printf("Sum of whole array = %i\n", *sum0 + *sum1);

return 0;
}
```

13. Write a program to implement first-fit, best-fit and worst-fit allocation strategies.

Code:-

first-fit –

```
// C++ implementation of First - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to
// blocks as per First fit algorithm
void firstFit(int blockSize[], int m,
            int processSize[], int n)
{
    // Stores block id of the
    // block allocated to a process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                // allocate block j to p[i] process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];

                break;
            }
        }
    }
```



```
    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t"
             << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver code
int main()
{
    cout << "First-Fit Alogorithm\n";
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {22, 41, 1112, 86};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```


best-fit –

```
// C++ implementation of Best - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per Best fit
// algorithm
void bestFit(int blockSize[], int m, int processSize[], int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process
        int bestIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }

        // If we could find a block for current process
        if (bestIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = bestIdx;

            // Reduce available memory in this block.
            blockSize[bestIdx] -= processSize[i];
```



```
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver code
int main()
{
    cout << "Best-fit  Allocation\n";
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0 ;
}
```


worst-fit –

```
// C++ implementation of worst - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

// Function to allocate memory to blocks as per worst fit
// algorithm
void worstFit(int blockSize[], int m, int processSize[],
                                                int n)
{
    // Stores block id of the block allocated to a
    // process
    int allocation[n];

    // Initially no block is assigned to any process
    memset(allocation, -1, sizeof(allocation));

    // pick each process and find suitable blocks
    // according to its size ad assign to it
    for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process
        int wstIdx = -1;
        for (int j=0; j<m; j++)
        {
            if (blockSize[j] >= processSize[i])
            {
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }

        // If we could find a block for current process
        if (wstIdx != -1)
        {
            // allocate block j to p[i] process
            allocation[i] = wstIdx;

            // Reduce available memory in this block.
```



```
            blockSize[wstIdx] -= processSize[i];
        }
    }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    {
        cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
        if (allocation[i] != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver code
int main()
{
    int blockSize[] = {130, 570, 210, 340, 600};
    int processSize[] = {122, 400, 11, 326};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    worstFit(blockSize, m, processSize, n);

    return 0 ;
}
```