WEEK-3

Ques:sum

```
echo Enter value for n
read n
sum=0
i=1
while [ $i –le $n ]
do
sum=$((sum+i))
i=$((i+2))
done
echo Sum is $sum
```

ques:Write a program to check whether the file has execute permission or not. If not, add the permission.

```
echo -n "Enter file name : "
read file

# find out if file has write permission or not
[ -w $file ] && W="Write = yes" || W="Write = No"

# find out if file has excute permission or not
[ -x $file ] && X="Execute = yes" || X="Execute = No"

# find out if file has read permission or not
[ -r $file ] && R="Read = yes" || R="Read = No"

echo "$file permissions"
echo "$W"
echo "$R"
echo "$X"
```

ques:Write a program to check all the files in the present working directory for a pattern (passed through command line) and display the name of the file followed by a message stating that the pattern is available or not available.

```
if [ -d $1 ]
then
    echo "The provided argument is the directory."

elif [ -f $1 ]
then
    echo "The provided argument is the file."

echo "The given argument does not exist on the file system."
fi
```

ques:Write a shell script to list only the name of sub directories in the present working
directory

```
echo "List of sub-directories present in this Folder - "
```

WEEK-4 process creation

ques:Write a program to print the Child process ID and Parent process ID in both

Child and Parent processes

```
#include <stdio.h>

#include <unistd.h>

int main()

{

  int p_id, p_pid;

    p_id = getpid(); /*process id*/

   p_pid = getpid(); /*parent process id*/

   printf("Process ID: %d\n", p_id);

   printf("Parent Process ID: %d\n", p_pid);

   return 0;

}
```

ques:The child process calculates the sum of odd numbers and the parent process calculates the sum of even numbers up to the number ‘n’. Ensure the Parent process waits for the child process to finish.

```
#include <iostream>
#include <unistd.h>
using namespace std;
int main()
{
   int a[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
   int sumOdd = 0, sumEven = 0, n, i;
   n = fork();

   if (n > 0) {
        for (i = 0; i < 10; i++) {
            if (a[i] % 2 == 0)
                sumEven = sumEven + a[i];
        }
        cout << "Parent process \n";
        cout << "Sum of even no. is " << sumEven << endl;
    }


    else {
        for (i = 0; i < 10; i++) {
            if (a[i] % 2 != 0)
                sumOdd = sumOdd + a[i];
        }
        cout << "Child process \n";
        cout << "\nSum of odd no. is " << sumOdd << endl;
    }
    return 0;
}
```

WEEK 06-process scheduling

FCFS:

```
#include<iostream>
using namespace std;
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);
    cout << "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];
        cout << "   " << i+1 << "\t\t" << bt[i] <<"\t     "
            << wt[i] <<"\t\t  " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
         << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
         << (float)total_tat / (float)n;
}
int main()
{
    //process id's
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    //Burst time of all processes
    int  burst_time[] = {10, 5, 8};

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

ROUND ROBIN:

```
        #include<stdio.h>

        #include<conio.h>

        void main()

        {

            int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

            float avg_wt, avg_tat;
```

```
    printf(" Total number of process in the system: ");
    scanf("%d", &NOP);
    y = NOP;
Time
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");  // Accept arrival time
scanf("%d", &at[i]);
printf(" \nBurst time is: \t"); // Accept the Burst time
scanf("%d", &bt[i]);
temp[i] = bt[i]; // store the burst time in temp array
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0) // define the conditions
{
    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("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i],
sum-at[i]-bt[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 Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
```

```
}
```

WEEK-07 PIPES

```
#include <stdio.h>
#include<unistd.h>
#include<sys/wait.h>
int main()
{
int p[2];
char buff[25];
pipe(p);
if(fork()==0)
{
printf("Child : Writing to pipe \n");
write(p[1],”Welcome”,8);
printf("Child Exiting\n");
}
else
{
wait(NULL);
printf("Parent : Reading from pipe \n");
read(p[0],buff,8);
printf("Pipe content is : %s \n",buff);
}
return 0;
}
```

Q
Write a program to implement the following command line pipe using pipe() and dup()

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

int main()
{
   int fd;
  char * myfifo = "/tmp/myfifo";
```

```
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        fd = open(myfifo, O_WRONLY);

        fgets(arr2, 80, stdin);

        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        fd = open(myfifo, O_RDONLY);

        read(fd, arr1, sizeof(arr1));

        printf("User2: %s\n", arr1);
        close(fd);
    }
    return 0;
}
```

WEEK-08,Message queue

Ques:send a message

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    fgets(message.mesg_text,MAX,stdin);

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

Ques:receive a message

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    key = ftok("progfile", 65);

    msgid = msgget(key, 0666 | IPC_CREAT);
    msgrcv(msgid, &message, sizeof(message), 1, 0);

    printf("Data Received is : %s \n",
                    message.mesg_text);

    msgctl(msgid, IPC_RMID, NULL);

    return 0;
}
```

WEEK-9 shared memory
write

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    key_t key = ftok("shmfile",65);
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    char *str = (char*) shmat(shmid,(void*)0,0);

    cout<<"Write Data : ";
    gets(str);

    printf("Data written in memory: %s\n",str);
    shmdt(str);

    return 0;
}
```

read

```
#include <iostream>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
```

```
{
   // ftok to generate unique key
   key_t key = ftok("shmfile",65);

   // shmget returns an identifier in shmid
   int shmid = shmget(key,1024,0666|IPC_CREAT);

   // shmat to attach to shared memory
   char *str = (char*) shmat(shmid,(void*)0,0);

   printf("Data read from memory: %s\n",str);

   //detach from shared memory
   shmdt(str);

   // destroy the shared memory
   shmctl(shmid,IPC_RMID,NULL);

   return 0;
}
```

WEEK10,overlay concept

```
#include<stdio.h>

#include<unistd.h>


void main() {

  int pid;

  pid = fork();


  /* Child process */

  if (pid == 0) {

    printf("Child process: Running Hello World Program\n");

    execl("./helloworld", "./helloworld", (char *)0);

    printf("This wouldn't print\n");

  } else { /* Parent process */
```

```
    sleep(3);
    printf("Parent process: Running While loop Program\n");
    execl("./while_loop", "./while_loop", (char *)0);
    printf("Won't reach here\n");
  }
  return;
}
```
QUES
```
#include<stdio.h>
#include<string.h>
#include<unistd.h>

void main(int argc, char *argv[0]) {
  int pid;
  int err;
  int num_times;
  char num_times_str[5];

  /* In no command line arguments are passed, then loop maximum count taken as
10 */
  if (argc == 1) {
    printf("Taken loop maximum as 10\n");
    num_times = 10;
    sprintf(num_times_str, "%d", num_times);
  } else {
    strcpy(num_times_str, argv[1]);
    printf("num_times_str is %s\n", num_times_str);
    pid = fork();
  }

  /* Child process */
  if (pid == 0) {
    printf("Child process: Running Hello World Program\n");
    err = execl("./helloworld", "./helloworld", (char *)0);
    printf("Error %d\n", err);
    perror("Execl error: ");
    printf("This wouldn't print\n");
  } else { /* Parent process */
    sleep(3);
    printf("Parent process: Running While loop Program\n");
    execl("./while_loop", "./while_loop", (char *)num_times_str, (char *)0);
    printf("Won't reach here\n");
  }
  return;
}
```

WEEK-11

mutual exclusion using system V semaphore

```
#include<sys/ipc.h>
#include<sys/sem.h>
int main()
{
int pid,semid,val;
struct sembuf sop;
semid=semget((key_t)6,1,IPC_CREAT|0666);
pid=fork();
sop.sem_num=0;
sop.sem_op=0;
sop.sem_flg=0;
if (pid!=0)
{
sleep(1);
printf("The Parent waits for WAIT signal\n");
semop(semid,&sop,1);
printf("The Parent WAKED UP & doing her job\n");
sleep(10);
printf("Parent Over\n");
}
else
{
printf("The Child sets WAIT signal & doing her job\n");
semctl(semid,0,SETVAL,1);
sleep(10);
printf("The Child sets WAKE signal & finished her job\n");
semctl(semid,0,SETVAL,0);
printf("Child Over\n");
}
return 0;
}
```

mutual exclusion using POSIX semaphore and threads.

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
sem_t mutex;
void* thread(void* arg)
{
//wait
sem_wait(&mutex);
printf("\nEntered..\n");
//critical section
sleep(4);
//signal
printf("\nJust Exiting...\n");
sem_post(&mutex);
}


int main()
{
sem_init(&mutex, 0, 1);
pthread_t t1,t2;
pthread_create(&t1,NULL,thread,NULL);
sleep(2);
pthread_create(&t2,NULL,thread,NULL);
pthread_join(t1,NULL);
pthread_join(t2,NULL);
sem_destroy(&mutex);
return 0;
}
```

WEEK-12-reader writer

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

sem_t x,y;
pthread_t tid;
pthread_t writerthreads[100],readerthreads[100];
int readercount;

void *reader(void* param)
{
    sem_wait(&x);
    readercount++;
    if(readercount==1)
    sem_wait(&y);
    sem_post(&x);
    printf("\n%d reader is inside",readercount);
    sem_wait(&x);
    readercount--;
    if(readercount==0)
    {
        sem_post(&y);
    }
    sem_post(&x);
    printf("\n%d Reader is leaving",readercount+1);
}

void *writer(void* param)
{
    printf("\nWriter is trying to enter");
    sem_wait(&y);
    printf("\nWriter has entered");
sem_post(&y);
    printf("\nWriter is leaving");
}

int main()
{
    int n2,i;
    printf("Enter the number of readers:");
    scanf("%d",&n2);
    int n1[n2];
    sem_init(&x,0,1);
    sem_init(&y,0,1);
    for(i=0;i<n2;i++)
    {
        pthread_create(&writerthreads[i],NULL,reader,NULL);
        pthread_create(&readerthreads[i],NULL,writer,NULL);
    }
    for(i=0;i<n2;i++)
    {
        pthread_join(writerthreads[i],NULL);
        pthread_join(readerthreads[i],NULL);
    }
}
```

WEEK-13

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

sem_t room;
sem_t chopstick[5];

void * philosopher(void *);
void eat(int);
int main()
{
int i,a[5];
pthread_t tid[5];

sem_init(&room,0,4);

for(i=0;i<5;i++)
sem_init(&chopstick[i],0,1);

for(i=0;i<5;i++){
a[i]=i;
pthread_create(&tid[i],NULL,philosopher,(void *)&a[i]);
}
for(i=0;i<5;i++)
pthread_join(tid[i],NULL);
}

void * philosopher(void * num)
{
int phil=*(int *)num;

sem_wait(&room);
printf("\nPhilosopher %d has entered room",phil);
sem_wait(&chopstick[phil]);
sem_wait(&chopstick[(phil+1)%5]);

eat(phil);
sleep(2);
printf("\nPhilosopher %d has finished eating",phil);

sem_post(&chopstick[(phil+1)%5]);
sem_post(&chopstick[phil]);
sem_post(&room);
}

void eat(int phil)
{
printf("\nPhilosopher %d is eating",phil);
}
```