**Aman Malik**

**Section-E**

**Roll number -11**

**University roll number -2014555**

PROBLEM 1:- Demonstration of FORK() System Call.

PROGRAM:-

#include<stdio.h>

#include<unistd.h>

int main()

{

printf("today\n");

fork();

printf("yesterday\n");

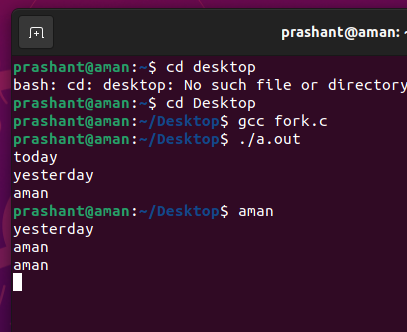
fork();

printf("aman\n");

return 0;

}

OUTPUT



PROBLEM 2:- Parent Process Computes the SUM OF EVEN and Child Process Computes the sum of ODD NUMBERS using fork.

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

int a[20],n,sum;

pid\_t pid;

scanf("%d",&n);

int i;

for(i=0;i<n;i++)

scanf("%d",&a[i]);

pid=fork();

if(pid==0)

{

sum=0;

for(i=0;i<n;i++)

{

if(a[i]%2==0)

sum+=a[i];

}

printf("sum of even %d \n",sum);

}

else

{

sum=0;

for(i=0;i<n;i++)

{

if(a[i]%2!=0)

sum+=a[i];

}

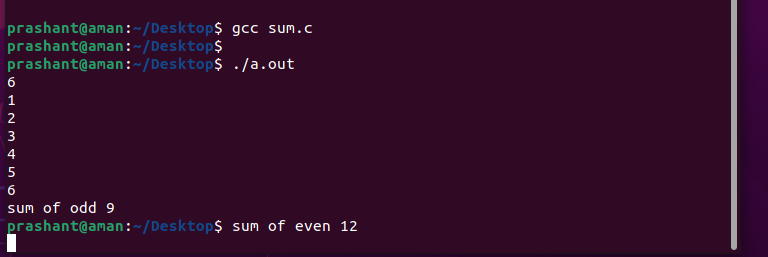
printf("sum of odd %d \n",sum);

}

return 0;

}

OUTPUT



PROBLEM 3:- Demonstration of WAIT() System Call.

PROGRAM:

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

pid\_t pid;

int status;

pid=fork();

if(pid==0)

{

printf("i am child\n");

exit(0);

}

else

{

wait(&status);

printf("i am parent\n");

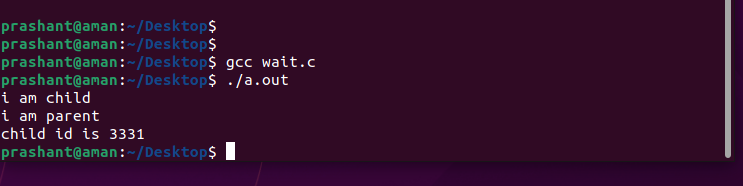
printf("child id is %d\n",pid);

}

return 0;

}

OUTPUT



PROBLEM 4:- Implementation of ORPHAN PROCESS & ZOMBIE PROCESS.

PROGRAM:

PART 1: ORPHAN PROCESS

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid==0)

{

sleep(6);

printf("i am child my pid is = %d and my parent id is = %d\n",getpid(),getppid());

}

else

{

printf("i am parent my child pid is = %d and my pid is = %d\n",pid,getpid());

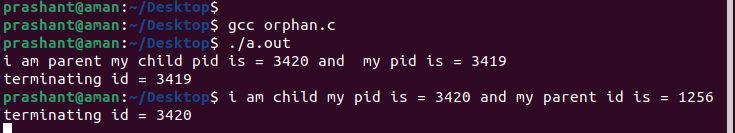
}

printf("terminating id = %d\n",getpid());

return 0;

}

OUTPUT



PART 2: ZOMBIE PROCESS=>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

pid\_t pid;

pid=fork();

if(pid!=0)

{

sleep(6);

}

else

{

exit(0);

}

return 0;

}

OUTPUT



PROBLEM 5:- Implementation of PIPE.

PROGRAM:

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

#include<string.h>

int main()

{

int pi[2];

pid\_t pid;

char a[100],str[11];

int nbr,nbw;

pipe(pi);

pid=fork();

if(pid==0)

{

printf("enter string: ");

fgets(str,12,stdin);

nbr=write(pi[1],str,strlen(str));

printf("child write %d bytes: \n",nbr);

exit(0);

}

else

{

nbw=read(pi[0],a,sizeof(a));

a[nbw]='\0';

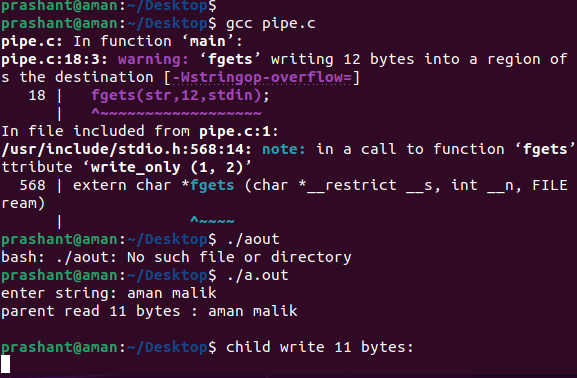
printf("parent read %d bytes : %s \n",nbw,a);

}

return 0;

}

OUTPUT



PROBLEM 6:- Implementation of FIFO

PROGRAM:

WRITER:

#include<sys/stat.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<sys/types.h>

#include<string.h>

int main()

{

int nbw,fd;

char str[100];

mknod("myfifo",S\_IFIFO|0666,0);

printf("writing for reader process: \n");

fd=open("myfifo",O\_WRONLY);

while(fgets(str,20,stdin))

{

nbw=write(fd,str,strlen(str));

printf("writer process write %d bytes\n %s",nbw,str);

}

return 0;

}

READER:

#include<sys/stat.h>

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<fcntl.h>

#include<sys/types.h>

#include<string.h>

int main()

{

int nbr,fd;

char a[100];

mknod("myfifo",S\_IFIFO|0666,0);

fd=open("myfifo",O\_RDONLY);

printf("if you get some data type something\n");

do

{

nbr=read(fd,a,sizeof(a));

a[nbr]='\0';

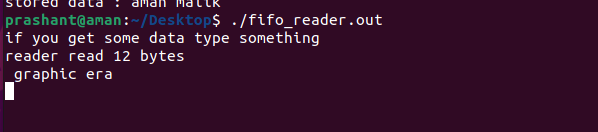
printf("reader read %d bytes\n %s",nbr,a);

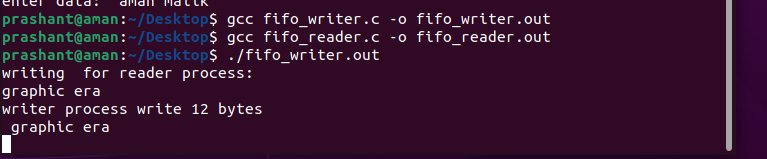
}while(nbr>0);

return 0;

}

OUTPUT





PROBLEM 7:- IMPLEMENT OF MESSAGE QUEUE

PROGRAM:

WRITER:

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/types.h>

#include<sys/msg.h>

#include<string.h>

struct msgbuf{

long t;

char str[100];

}svrname;

int main()

{

key\_t key;

key=ftok("progfile",'A');

int msgid,c;

msgid=msgget(key,0666|IPC\_CREAT);

svrname.t=1;

printf("enter string: ");

fgets(svrname.str,20,stdin);

c=msgsnd(msgid,&svrname,sizeof(svrname),0);

printf("sender wrote msg: %s",svrname.str);

return 0;

}

READER:

#include<stdio.h>

#include<sys/ipc.h>

#include<sys/types.h>

#include<sys/msg.h>

#include<string.h>

struct msgbuf{

long t;

char str[100];

}svrname;

int main()

{

key\_t key;

key=ftok("progfile",'A');

int msgid,c;

msgid=msgget(key,0666|IPC\_CREAT);

msgrcv(msgid,&svrname,sizeof(svrname),1,0);

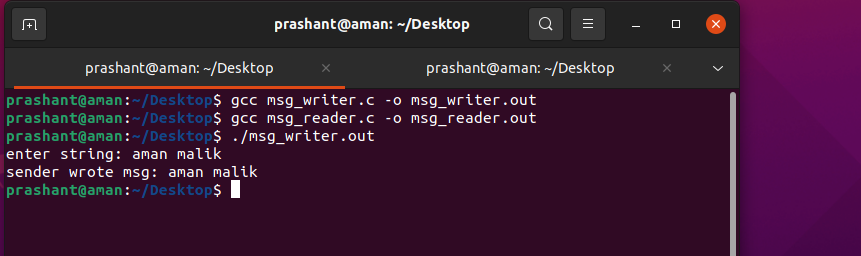
printf("reciver recive message : %s",svrname.str);

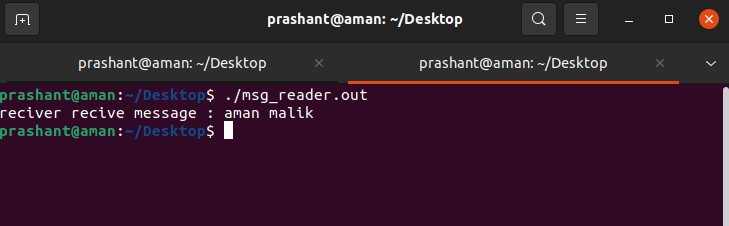
msgctl(msgid,IPC\_RMID,NULL);

return 0;

}

OUTPUT





PROBLEM 8:- IMPLEMENT OF SHARED MEMORY.

PROGRM:

WRITER:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<sys/types.h>

#include<sys/shm.h>

#include<sys/ipc.h>

int main()

{

char \*ptr;

key\_t key;

int shmid;

key=ftok("shmfile",'A');

shmid=shmget(key,1024,0666|IPC\_CREAT);

ptr=(char\*)shmat(shmid,(void\*)0,0);

printf("enter data: ");

fgets(ptr,100,stdin);

shmdt(ptr);

return 0;

}

READER:

#include<stdio.h>

#include<string.h>

#include<sys/types.h>

#include<sys/shm.h>

#include<sys/ipc.h>

int main()

{

char \*ptr;

int shmid;

key\_t key;

key=ftok("shmfile",'A');

shmid=shmget(key,1024,0666|IPC\_CREAT);

ptr=(char\*)shmat(shmid,(void\*)0,0);

printf("stored data : %s",ptr);

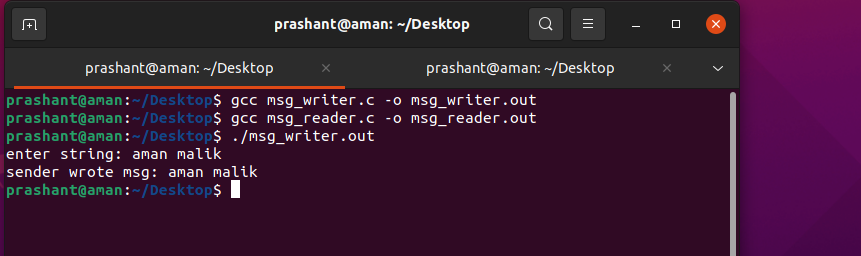
shmdt(ptr);

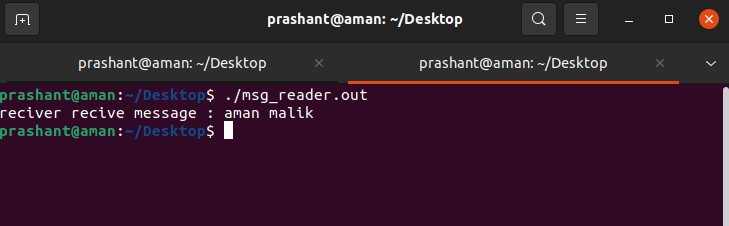
shmctl(shmid,IPC\_RMID,NULL);

return 0;

}

OUTPUT





PROBLEM 9:-IMPLEMENTATION OF FIRST COME FISRT SERVE.

PROGRAM:

#include<malloc.h>

#include<stdio.h>

#include<string.h>

typedef struct node{

char process[20];

int burst,arrival;

struct node \*next;

}node;

node \*front=NULL;

node \*rear=NULL;

void insert()

{

node \*p;

int a,b;

char str[20];

p=(node\*)malloc(sizeof(node));

printf("\nenter the process id: ");

scanf("%s",p->process);

printf("enter arrival time: ");

scanf("%d",&a);

printf("enter burst time: ");

scanf("%d",&b);

p->arrival=a;

p->burst=b;

if(front==NULL)

{

front=p;

rear=p;

}

else

{

rear->next=p;

rear=p;

}

}

void display(int n)

{

node \*temp=front;

int waite=0,exe=0;

float turn=0.0;

if(front!=NULL)

{

printf("\n-----------------------------------------------process--------------------\n");

while(temp!=NULL)

{

printf("%s\t",temp->process);

temp=temp->next;

}

temp=front;

printf("\n-----------------------------------------------burst time-----------------\n");

while(temp!=NULL)

{

printf("%d\t",temp->burst);

temp=temp->next;

}

printf("\n------------------------------------------------start time-----------------\n");

temp=front;

while(temp!=NULL)

{

waite+=exe;

turn+=exe+temp->burst;

exe+=temp->burst;

printf("%d\t",exe);

temp=temp->next;

}

printf("\naverage waiting time: %d\n",waite/n);

printf("average turn around time: %f\n",turn/n);

}

}

int main()

{

int n;

printf("enter number of process: ");

scanf("%d",&n);

int i;

for(i=0;i<n;i++)

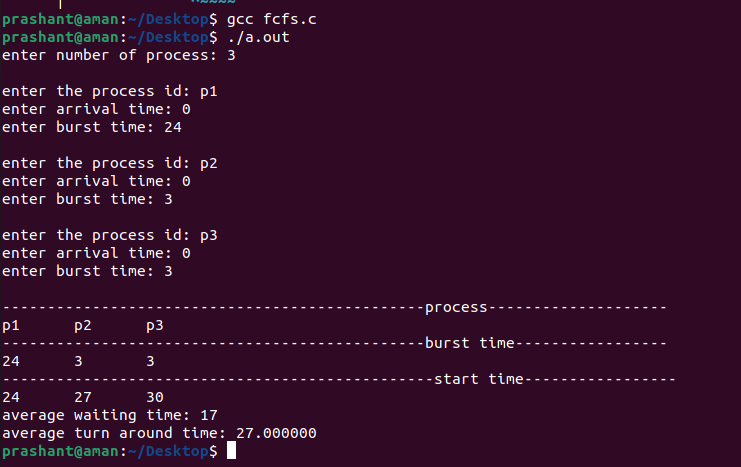
insert();

display(n);

return 0;

}

OUTPUT



PROBLEM 10:-IMPLEMENTATION OF SHORTEST JOS FIRST.

PROGRAM:

#include<malloc.h>

#include<stdio.h>

#include<string.h>

typedef struct node{

char process[20];

int burst,arrival;

struct node \*next;

}node;

node \*front=NULL;

node \*rear=NULL;

void insert()

{

node \*p,\*temp;

int a,b;

char str[20];

p=(node\*)malloc(sizeof(node));

printf("\nenter the process id: ");

scanf("%s",p->process);

printf("enter arrival time: ");

scanf("%d",&a);

printf("enter burst time: ");

scanf("%d",&b);

p->arrival=a;

p->burst=b;

if(front==NULL)

{

front=p;

rear=p;

}

else if(p->burst<front->burst)

{

p->next=front;

front=p;

}

else if(p->burst>rear->burst)

{

rear->next=p;

rear=p;

}

else

{

temp=front;

while(p->burst>(temp->next)->burst)

temp=temp->next;

p->next=temp->next;

temp->next=p;

}

}

void display(int n)

{

node \*temp=front;

int waite=0,exe=0;

float turn=0.0;

if(front!=NULL)

{

printf("\n-----------------------------------------------process--------------------\n");

while(temp!=NULL)

{

printf("%s\t",temp->process);

temp=temp->next;

}

temp=front;

printf("\n-----------------------------------------------burst time-----------------\n");

while(temp!=NULL)

{

printf("%d\t",temp->burst);

temp=temp->next;

}

printf("\n------------------------------------------------start time-----------------\n");

temp=front;

while(temp!=NULL)

{

waite+=exe;

turn+=exe+temp->burst;

exe+=temp->burst;

printf("%d\t",exe);

temp=temp->next;

}

printf("\naverage waiting time: %d\n",waite/n);

printf("average turn around time: %f\n",turn/n);

}

}

int main()

{

int n;

printf("enter number of process: ");

scanf("%d",&n);

int i;

for(i=0;i<n;i++)

insert();

display(n);

return 0;

}

OUTPUT

