

Alternancia

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>
#include <sys/shm.h>
#define key 1555
#define key2 1553

void proce(int i,int proceso[],int llegada[],int b[],
           int rci[],int rcd[],int *turno,int *suma);

int main(int argc, char * argv[])
{
    int pid1, pid2, pid3, pid4, estado;
    int p1_finalizado = 0, p2_finalizado = 0, p3_finalizado = 0, p4_finalizado = 0;
    int proceso[4];
    int llegada[4];
    int rci[4], rcd[4];
    int opc;
    int b[4];
    int t1=0;
    int id, id2;
    int *suma=NULL;
    int *turno=NULL;
    b[0]=0; b[1]=0; b[2]=0; b[3]=0;
    id=shmget(key, sizeof(int), IPC_CREAT|SHMR|SHMW); //se crea una memoria com
    if (id == -1)//Verifica si se creo la memoria
    {
        perror("shmget:");
        exit(-1);
    }
    turno= (int *) shmat (id, NULL, 0); //Ata segmento de memoria
    (*turno)=0;
    id2=shmget(key2, sizeof(int), IPC_CREAT|SHMR|SHMW); //se crea una memoria c
    if (id2 == -1)//Verifica si se creo la memoria
    {
        perror("shmget:");
        exit(-1);
    }
    suma= (int *) shmat (id2, NULL, 0); //Ata segmento d memoria
    (*suma)=0;
    int i;
    for(i=0; i<4; i++)
    {
        printf("Tiempo para proceso %d: ", i+1); //se llenan los procesos c
        scanf("%d",&proceso[i]);
        printf("Region critica\n1) Si\n2) No\n"); //Pregunta si tiene region
        scanf("%d",&opc);
    }
}
```

```

        if(opc==1)//Pide datos de la region
        {
            llegada[i]=t1;
            do//Pide datos de la region para los procesos que la tienen
            {
                printf("En que tiempo inicia\n");
                scanf("%d",&rci[i]);
            }while(rci[i]>proceso[i] && rci[i]<0);
            do//Pide dato de la duracion para los procesos que tienen
            {
                printf("Duracion\n");
                scanf("%d",&rcd[i]);
            }while(rcd[i]<0);
            t1++;
        }
        else//Para los procesos que no tienen region critica
        {
            rci[i]=-1;
            llegada[i]=-1;
        }
    }

//Empiezan a ejecutarse los procesos
pid1=fork();
/* Este es el proceso 4 */
if (pid1 == 0)
{
    while(proceso[3]!=0)//mientras el tiempo del proceso sea diferente
    {
        printf("Proceso 4\n");
        proce(3,proceso,llegada,b,rci,rcd,turno,suma);
    }
    puts("Proceso #4 finalizado.\n");
    exit(0);
}
pid2=fork();
/* Este es el proceso #3 */
if (pid2 == 0)
{
    while(proceso[2]!=0)//mientras el tiempo del proceso sea diferente de 0
    {
        printf("Proceso 3\n");
        proce(2,proceso,llegada,b,rci,rcd,turno,suma);
    }
    puts("Proceso #3 finalizado.\n");
    exit(0);
}
pid3=fork();
/* Este es el proceso #2 */
if (pid3 == 0)
{
    while(proceso[1]!=0)
    {

```

```

        printf("Proceso 2\n");
        proce(1,proceso ,llegada ,b,rci ,rcd ,turno ,suma);
    }
    puts("Proceso #2 finalizado.\n");
    exit (0);
}
pid4=fork();
/* Este es el proceso #1 */
if (pid4 == 0)
{
    while(proceso[0]!=0)
    {
        printf("Proceso 1\n");
        proce(0,proceso ,llegada ,b,rci ,rcd ,turno ,suma);
    }
    puts("Proceso #1 finalizado.\n");
    exit (0);
}
if ((pid1 < 0) || (pid2 < 0) || (pid3 < 0) || (pid4 < 0))
{ // se verifica que se hayan creado bien los procesos
    printf("No creados...\n");
    exit (1);
}
if ((pid1 > 0) && (pid2 > 0) && (pid3 > 0) && (pid4 > 0))
{ // si los procesos han sido creados bien
    while((!p1_finalizado) || (!p2_finalizado) || (!p3_finalizado) || (!
p4_finalizado))
    {
        int pid;
        //se espera informacion de los procesos
        pid = wait(&estado);
        //se verifica que proceso ha finalizado y se marca
        if (pid == pid1)
            p1_finalizado = 1;
        if (pid == pid2)
            p2_finalizado = 1;
        if (pid == pid3)
            p3_finalizado = 1;
        if (pid == pid4)
            p4_finalizado = 1;
    }
    //Se imprime que han terminado los procesos asi como la region critica
    puts("Procesos terminados.\n");
    printf("%d",*suma);
}
}

void proce(int i,int proceso[],int llegada[],int b[],int rci[],
int rcd[],int *turno,int *suma)
{
    if(rci[i]==proceso[i] && proceso[i]!=0 && b[i]!=1 && rci[i]!=-1)//se verifi
//el tiempo
    {

```

```

printf("Intentando entrar a region critica\n");
if(*turno==llegada[i])
{
    // se verifica el turno
    printf("En region critica\n");
    printf("Tiempo restante region critica: %d\n",rcd[i]);
    rcd[i]=rcd[i]-1;
    *suma=*suma+1; //se modifica el valor de la region critica
    sleep(1);
    if(rcd[i]==0)//cuando acabe el tiempo de la region se incre
    {
        printf("Proceso %d\n",i+1);
        printf("Saliendo de la region critica\n");
        proceso[i]--; // se resta tiempo al proceso
        b[i]=1;
        *turno=*turno+1; // se incrementa turno
        printf("Incrementando turno: %d\n",*turno);
        sleep(1);
    }
}
else //Si no desocupa region critica
{
    printf("Memoria ocupada\n");
    printf("Lugar: %d\n",llegada[i]);
    sleep(1);
}
else
{
    if(b[i]==1)// se verifica si el proceso ya ejecuto la region critica
    {
        printf("Tiempo: %d\n",proceso[i]);
        proceso[i]--; // se resta tiempo al proceso
        printf("Este proceso ya ejecuto su region critica\n");
        sleep(1);
    }
    else
    {
        printf("Tiempo: %d\n",proceso[i]);
        proceso[i]--; // se resta tiempo al proceso
        sleep(1);
    }
}
}
}

```

```

#include <sys/wait.h>
#include <stdlib.h>
#include <time.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <sys/shm.h>

#define FILEKEY "/bin/cat"
#define KEY 1300
// #define NUM 10
int NUM=0;
pid_t pidL, pidM;
// int num=10;
void manejador()
{
    printf("Recibi la senal");
    kill(getpid(), SIGKILL);
    kill(pidL, SIGKILL);
    kill(pidM, SIGKILL);
}

int main ()
{
    // estructura de la senal
    struct sigaction act;
    act.sa_handler = manejador;

```

Codigo de Carpeta 1

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <sys/shm.h> /* shm* */

#define FILEKEY "/bin/cat"
#define KEY 1300
// #define NUM 10
int NUM=0;
pid_t pidL, pidM;
// int num=10;
void manejador()
{
    printf("Recibi la senal");
    kill(getpid(), SIGKILL);
    kill(pidL, SIGKILL);
    kill(pidM, SIGKILL);
}

int main ()
{
    // estructura de la senal
    struct sigaction act;
    act.sa_handler = manejador;

```

```

sigemptyset (&act.sa_mask);
act.sa_flags=0;
sigaction(SIGALRM,&act,NULL);
alarm(3);
//Declaracion de variables
int fd[2];
int key, i;
int id_zone;
int *buffer;
char c;
char a;
pipe(fd);//Creacion de tuberia
/*El proceso padre crea la memoria compartida para la sincronizacion de los
//LLava e para la memoria compartida
key = ftok(FILEKEY, KEY);//crea la llave
if (key == -1)//Si no se pudo crear la llave
{
    fprintf(stderr, "Error al crear la llave \n");//Despliega letrero
    return -1;
}
//Crea la memoria compartida
id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
if (id_zone == -1)//Si no se pudo crear la memoria
{
    fprintf(stderr, "Error al crear memoria compartida\n");//Despliega
    return -1;
}
//printf ("ID zone shared memory: %d\n", id_zone);//Imprime el ID de la zona
//Ata memoria compartida
buffer = shmat (id_zone, (char *)0, 0);
if (buffer == NULL)//Si no se puede atar la memoria
{
    fprintf(stderr, "Error al reservar memoria compartida \n");//Imprime
    return -1;
}
//printf ("Puntero al buffer de la memoria compartida %p\n", buffer);//Imprime
//printf("Soy el proceso R \t Mi ID es: %d\n",getpid());
pipe(fd);//Creacion de tuberia
pidL = fork();//Creacion de proceso L
if (pidL == 0)//Si se creo el proceso L
{
    /*Crea memoria compartida para modificar variable compartida, incremento
    //printf("Proceso hijo %d intentando entrar a memoria",getpid());
    int key, i, id_zone, *buffer;
    /*LLave para memoria compartida */
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf(stderr, "Error al crear llave \n");
        return -1;
    }
    /* Se crea la memoria comartida*/
    id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);

```

```

    if (id_zone == -1)
    {
        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }
    //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
    /* Declaracion de la memoria compartida */
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }

    //printf ("Puntero del buffer de la memoria compartida: %p\n", buf
    while(1)
    {
        //printf(" %d\t", getpid());
        NUM++;
        write(fd[1], &NUM, sizeof(int));
    }
}
pidM = fork(); // Proceso de proceso M
if (pidM == 0)
{
    /*Crea memoria compartida para modificar variable compartida, decre
    //printf("Proceso hijo %d intentando entrar a memoria", getpid());
    int key, i, id_zone, *buffer;
    /*LLave para memoria compartida */
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf (stderr, "Error al crear llave \n");
        return -1;
    }
    /* Se crea la memoria comartida*/
    id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
    if (id_zone == -1)
    {
        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }
    //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
    /* Declaracion de la memoria compartida */
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }
    //printf ("Puntero del buffer de la memoria compartida: %p\n", buf
    while(1)
    {

```

```

        //printf(" %d\t",getpid());
        NUM--;
        write(fd[1],&NUM,sizeof(int));
    }
}
while (1)//Imprime los datos a pantalla
{
    //printf("Soy el prceso padre \t Mi ID es: %d\n",getpid());
    read(fd[0],&NUM,sizeof(int));
    printf("%d\n",NUM);
    read(fd[0],&NUM,sizeof(int));
    printf("%d\n",NUM);
}
c = getchar();
//libera la memoria compartida
shmdt ((char *)buffer);
shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
return 0;
}

```

```

Actividades Terminal dom 12:00
Comunica2.c
~/Documentos
Guardar
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>

#define FILE_NAME "Comunica2.dat"
#define KEY 1234567890
#define NUM 0

int NUM=0;
pid_t pidL;

//int num=1;
void maneja
{
    pri
    kil
    kil
    kil
}

int main ()
{
    //e
    str
    act
    sig
    act
    sig
    ala
    //D
    int fd[2];
    int key, i;
    int id_zone;
    int *buffer;
    char c;
    char a;
}

```

Codigo de carpeta 2

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <unistd.h>

```



```

int main()
{
    pid_t pid1, pid2;
    pid1=fork();
    if(pid1>0){
        pid2=fork();
        if(pid2>0){//Padre C
            int id,*ap,status,c[15],suma,i;
            suma=0;
            wait(&status);
            wait(&status);
            id=shmget(ftok(".", '&'), sizeof(c), IPC_CREAT);
            ap=shmat(id, 0, 0);
            for(i=0; i<15; i++){
                printf("ap %d: %d\n", i, *ap+i);
                suma+=(*ap+i);
            }
            shmdt(ap);
            shmctl(id, IPC_RMID, NULL);
            id=shmget(ftok(".", '%'), sizeof(int), IPC_CREAT);
            ap=shmat(id, 0, 0);
            *ap=suma;
        }
        if(pid2==0){//Hijo B
            int id,*ap,B[15],i;
            id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
            ap=shmat(id, 0, 0);
            for(i=1; i<=15; i+=2){
                //*(ap+i)=i;
                *(ap+i)=(100);
            }
        }
    }
    if(pid1==0){//Hijo A
        int id,*ap,B[15],i;
        id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
        ap=shmat(id, 0, 0);
        for(i=0; i<15; i+=2){
            //*(ap+i)=i;
            *(ap+i)=(100);
        }
    }
}

```

The screenshot shows a code editor window titled 'ejem1.c' with a file icon and a 'Guardar' button. The code is a C program with three main functions: `Padre C`, `Hijo B`, and `Hijo A`. The `Padre C` function creates a shared memory segment, prints its address, and then calls `Hijo B` and `Hijo A`. `Hijo B` and `Hijo A` both access the shared memory segment. The terminal window shows the compilation and execution of the program, resulting in a segmentation fault.

```

if(pid2>0){//Padre C
    int id,*ap,status,c[15],suma,i;
    suma=0;
    wait(&status);
    wait(&status);
    id=shmget(ftok(".", '&'), sizeof(c), IPC_CREAT);
    ap=shmat(id, 0, 0);
    for(i=0; i<15; i++){
        printf("ap %d: %d\n", i, *ap+i);
        suma+=(*ap+i);
    }
    shmdt(ap);
    shmctl(id, IPC_RMID, NULL);
    id=shmget(ftok(".", '%'), sizeof(int), IPC_CREAT);
    ap=shmat(id, 0, 0);
    *ap=suma;
}

if(pid2==0){//Hijo B
    int id,*ap,B[15],i;
    id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
    ap=shmat(id, 0, 0);
    for(i=1; i<15; i+=2){
        /*(ap+i)=i;
        *(ap+i)=(100);
    }
}

if(pid1==0){//Hijo A
    int id,*ap,B[15],i;
    id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
    ap=shmat(id, 0, 0);
    for(i=0; i<15; i+=2){
        /*(ap+i)=i;
        *(ap+i)=(100);
    }
}
}

```

```

victorkarmaroach@localhost:~/Documentos
Archivo Editar Ver Buscar Terminal Ayuda
[victorkarmaroach@localhost Documentos]$ gcc ejem1.c -o ejem1
[victorkarmaroach@localhost Documentos]$ ./ejem1
Segmentation fault (core generado)
[victorkarmaroach@localhost Documentos]$

```

```

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <string.h>
#include <errno.h>
#include <sys/shm.h> /* shm* */
#include <unistd.h>

```

```

#define FILEKEY "/bin/cat"
#define KEY 1300
#define MAXBUF 10

```

```

int main ()
{
    int key, i;
    int id_zone;
    int *buffer;
    char c;
    pid_t pid1;
    //LLava e para la memoria compartida
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf(stderr, "Error al crear la llave \n");
        return -1;
    }
    //Crea la memoria compartida

```

```

id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
if (id_zone == -1)
{
    fprintf (stderr, "Error al crear memoria compartida\n");
    return -1;
}
printf ("ID zone shared memory: %d\n", id_zone);
//Declarar memoria compartida
buffer = shmat (id_zone, (char *)0, 0);
if (buffer == NULL)
{
    fprintf (stderr, "Error al reservar memoria compartida \n");
    return -1;
}
printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
for (i = 0; i < MAXBUF; i++)
{
    buffer[i] = i;
}
pid1=fork();
//Crea un hijo para entrar en memria compartida.
if (pid1 == 0)
{
    printf("Proceso hijo %d intentando entrar a memoria",getpid());
    int key, i, id_zone, *buffer;
    /*LLave para memoria compartida */
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf (stderr, "Error al crear llave \n");
        return -1;
    }
    /* Se crea la memoria comartida*/
    id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
    if (id_zone == -1)
    {
        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }

    printf ("ID de la zona de memoria compartida: %d\n", id_zone);
    /* Declaracion de la memoria compartida */
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }

    printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
    /* Escribe los valores a la memoria */
    for (i = 0; i < MAXBUF; i++)

```

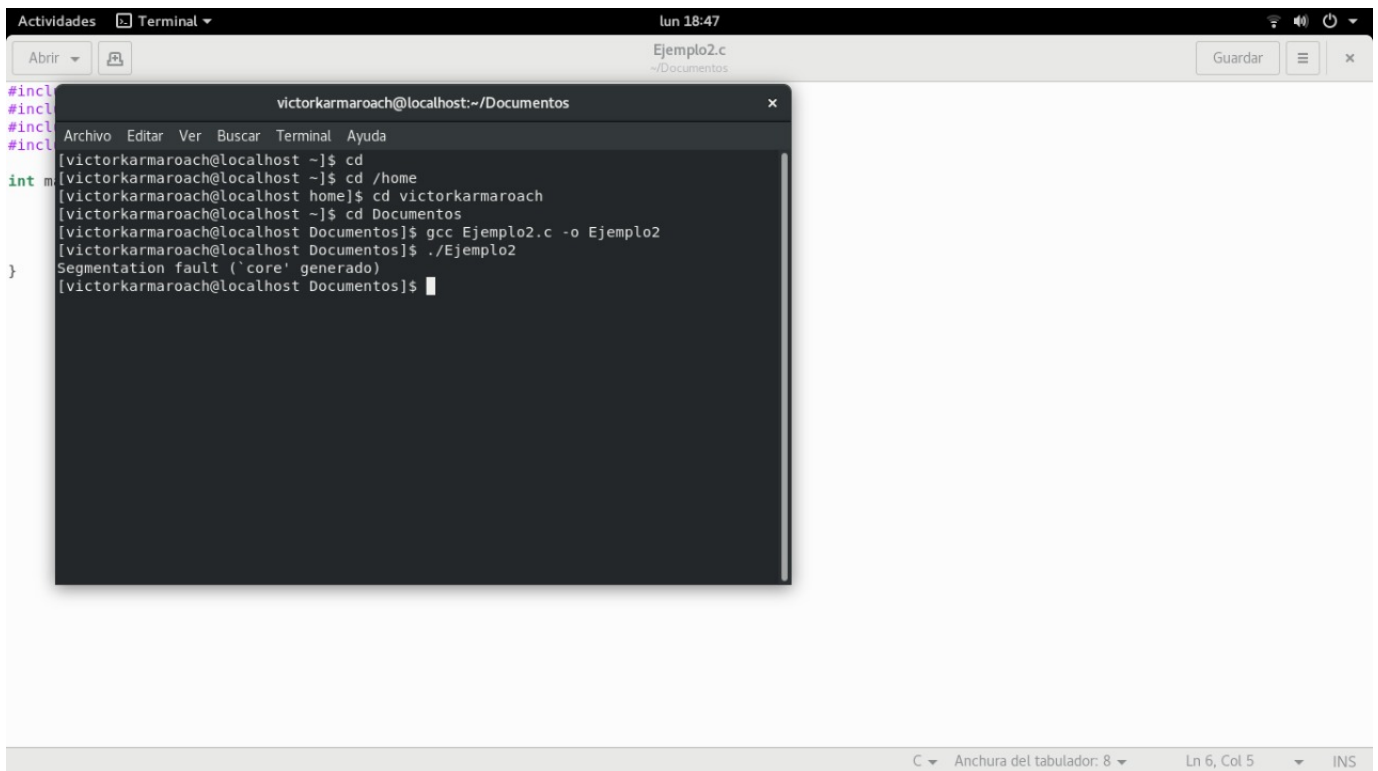
```

    {
        printf ("%i\n", buffer[i]);
    }
    //printf("Proceso hijo  %d intentando entrar a memoria",getpid());
}
c = getchar();
//libera la memoria compartida
shmdt ((char *)buffer);
shmctl (id_zone , IPC_RMID, (struct shmid_ds *)NULL);
return 0;
}

```

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(){
    int id, *ap;
    id=shmget (ftok (".", '%'), sizeof(int), IPC_CREAT);
    ap=shmat (id, 0, 0);
    printf("Suma:  %d\n", *ap);
}
```



```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <signal.h>
#include <sys/shm.h> /* shm* */

#define FILEKEY "/bin/cat"
#define KEY 1300
// #define NUM 10
int NUM=0;
pid_t pidL, pidM;
// int num=10;
void manejador()
{
    printf("Recibi la senal");
    kill(getpid(), SIGKILL);
    kill(pidL, SIGKILL);
    kill(pidM, SIGKILL);
}

int main ()
{
    // estructura de la senal
    struct sigaction act;
    act.sa_handler = manejador;
    sigemptyset (&act.sa_mask);
    act.sa_flags=0;
```

```

sigaction(SIGALRM,&act,NULL);
alarm(3);
//Declaracion de variables
int fd[2];
int key, i;
int id_zone;
int *buffer;
char c;
char a;
pipe(fd);//Creacion de tuberia
/*El proceso padre crea la memoria compartida para la sincronizacion de los
//LLava e para la memoria compartida
key = ftok(FILEKEY, KEY);//crea la llave
if (key == -1)//Si no se pudo crear la llave
{
    fprintf(stderr, "Error al crear la llave \n");//Despliega letrero
    return -1;
}
//Crea la memoria compartida
id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
if (id_zone == -1)//Si no se pudo crear la memoria
{
    fprintf (stderr, "Error al crear memoria compartida\n");//Despliega
    return -1;
}
//printf ("ID zone shared memory: %d\n", id_zone);//Imprime el ID de la zona
//Ata memoria compartida
buffer = shmat (id_zone, (char *)0, 0);
if (buffer == NULL)//Si no se puede atar la memoria
{
    fprintf (stderr, "Error al reservar memoria compartida \n");//Imprime
    return -1;
}
//printf ("Puntero al buffer de la memoria compartida %p\n", buffer);//Imprime
//printf("Soy el proceso R \t Mi ID es: %d\n",getpid());
pipe(fd);//Creacion de tuberia
pidL = fork();//Creacion de proceso L
if (pidL == 0)//Si se creo el proceso L
{
    /*Crea memoria compartida para modificar variable compartida, incrementa
    //printf("Proceso hijo %d intentando entrar a memoria",getpid());
    int key, i, id_zone, *buffer;
    /*LLave para memoria compartida */
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf (stderr, "Error al crear llave \n");
        return -1;
    }
    /* Se crea la memoria compartida*/
    id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
    if (id_zone == -1)
    {

```

```

        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }
    //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
    /* Declaracion de la memoria compartida */
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }

    //printf ("Puntero del buffer de la memoria compartida: %p\n", buf
    while(1)
    {
        //printf(" %d\t", getpid());
        NUM++;
        write(fd[1], &NUM, sizeof(int));
    }
}
pidM = fork();//Proceso de proceso M
if (pidM == 0)
{
    /*Crea memoria compartida para modificar variable compartida, decre
    //printf("Proceso hijo %d intentando entrar a memoria",getpid());
    int key, i, id_zone, *buffer;
    /*LLave para memoria compartida */
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf (stderr, "Error al crear llave \n");
        return -1;
    }
    /* Se crea la memoria comartida*/
    id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
    if (id_zone == -1)
    {
        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }
    //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
    /* Declaracion de la memoria compartida */
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }
    //printf ("Puntero del buffer de la memoria compartida: %p\n", buf
    while(1)
    {
        //printf(" %d\t", getpid());
        NUM--;

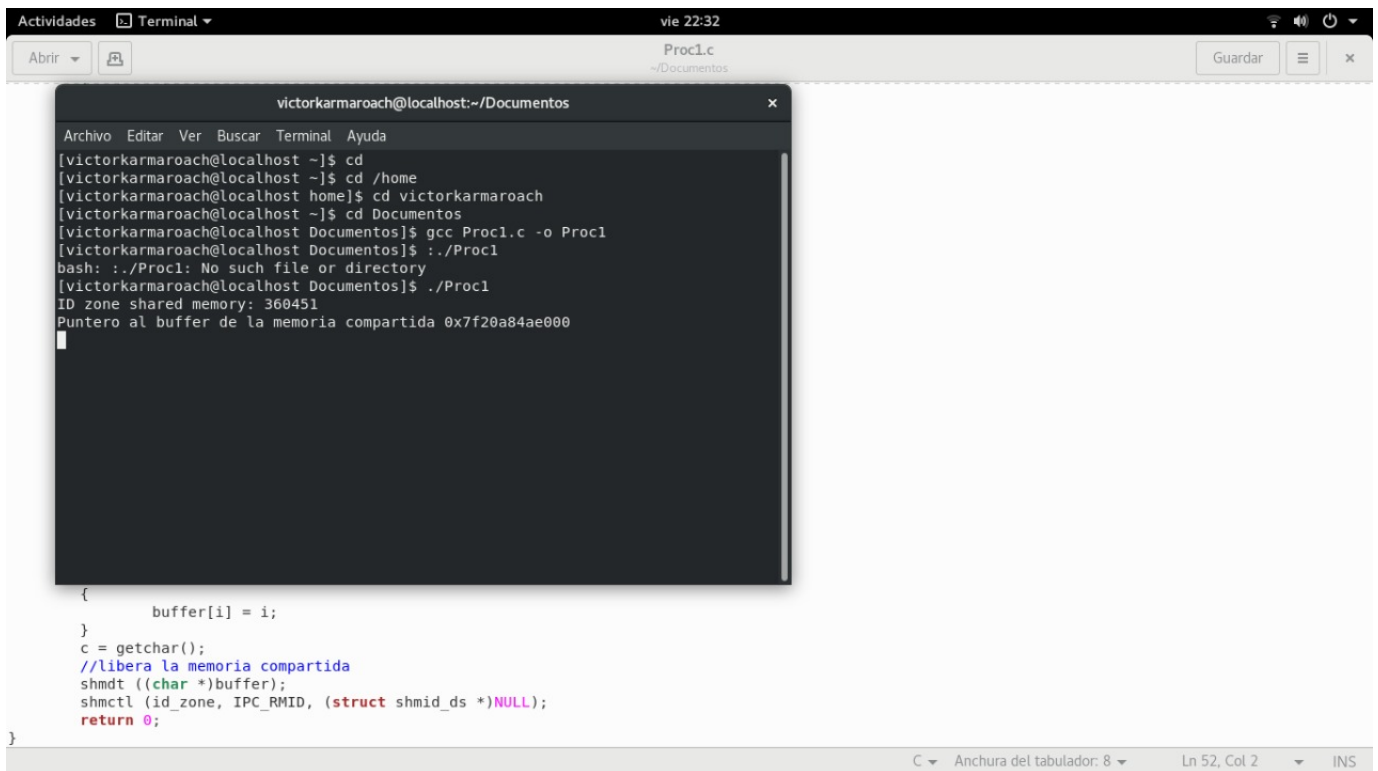
```



```

int main ()
{
    int key, i;
    int id_zone;
    int *buffer;
    char c;
    //LLava e para la memoria compartida
    key = ftok(FILEKEY, KEY);
    if (key == -1)
    {
        fprintf (stderr, "Error al crear la llave \n");
        return -1;
    }
    //Crea la memoria compartida
    id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
    if (id_zone == -1)
    {
        fprintf (stderr, "Error al crear memoria compartida\n");
        return -1;
    }
    printf ("ID zone shared memory: %d\n", id_zone);
    //Declarar memoria compartida
    buffer = shmat (id_zone, (char *)0, 0);
    if (buffer == NULL)
    {
        fprintf (stderr, "Error al reservar memoria compartida \n");
        return -1;
    }
    printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
    for (i = 0; i < MAXBUF; i++)
    {
        buffer[i] = i;
    }
    c = getchar();
    //libera la memoria compartida
    shmdt ((char *)buffer);
    shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
    return 0;
}

```



```
victorkarmaroach@localhost:~/Documentos
Archivo Editar Ver Buscar Terminal Ayuda
[victorkarmaroach@localhost ~]$ cd
[victorkarmaroach@localhost ~]$ cd /home
[victorkarmaroach@localhost home]$ cd victorkarmaroach
[victorkarmaroach@localhost ~]$ cd Documentos
[victorkarmaroach@localhost Documentos]$ gcc Proc1.c -o Proc1
[victorkarmaroach@localhost Documentos]$ ./Proc1
bash: ./Proc1: No such file or directory
[victorkarmaroach@localhost Documentos]$ ./Proc1
ID zone shared memory: 360451
Puntero al buffer de la memoria compartida 0x7f20a84ae000
{
    buffer[i] = i;
}
c = getchar();
//libera la memoria compartida
shmdt ((char *)buffer);
shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
return 0;
}
```

C Anchura del tabulador: 8 Ln 52, Col 2 INS

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

#define SIZE 512

int main ( char argc, char **argv )
{
    pid_t pid;
    int p[2], readbytes;
    char buffer[SIZE];

    pipe( p );

    if ( (pid=fork()) == 0 )
    { // hijo
        close( p[1] ); /* cerramos el lado de escritura del pipe */

        while( (readbytes=read( p[0], buffer, SIZE )) > 0)
            write( 1, buffer, readbytes );

        close( p[0] );
    }
    else
    { // padre
        close( p[0] ); /* cerramos el lado de lectura del pipe */

        strcpy( buffer, "Esto llega a traves de la tuberia\n" );
```

```

    write( p[1], buffer, strlen( buffer ) );

    close( p[1] );
}
write( pid, NULL, 0 );
exit( 0 );
}

```

The screenshot shows a Linux desktop environment. The top panel displays the time as 21:50 and the date as 'vie'. Below the panel, there is a window titled 'Terminal' showing the source code of a C program named 'Tubc2.c'. The code implements a pipe-based communication between a parent and a child process. The parent process writes the string 'Esto llega a traves de la tuberia' to the pipe, and the child process reads it. The code includes headers for `<fcntl.h>`, `<unistd.h>`, `<stdio.h>`, `<stdlib.h>`, and `<string.h>`. It defines a constant `SIZE` as 512. The `main` function uses `fork()` to create a child process. The child process closes its write end of the pipe and reads data from the read end. The parent process closes its read end of the pipe and writes data to the write end. Both processes then close their respective ends of the pipe and exit. The terminal window, titled 'victorkarmaroach@localhost:~/Documentos', shows the command `./Tubc2` being executed, and the output 'Esto llega a traves de la tuberia' being displayed.

```

#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 512

int main ( char argc, char **argv )
{
    pid_t pid;
    int p[2], readbytes;
    char buffer[SIZE];

    pipe( p );

    if ( (pid=fork()) == 0 )
    { // hijo
        close( p[1] ); /* cerramos el lado de escritura del pipe */

        while( (readbytes=read( p[0], buffer, SIZE )) > 0)
            write( 1, buffer, readbytes );

        close( p[0] );
    }
    else
    { // padre
        close( p[0] ); /* cerramos el lado de lectura del pipe */

        strcpy( buffer, "Esto llega a traves de la tuberia\n" );
        write( p[1], buffer, strlen( buffer ) );

        close( p[1] );
    }
    write( pid, NULL, 0 );
    exit( 0 );
}

```

victorkarmaroach@localhost:~/Documentos

```

Archivo Editar Ver Buscar Terminal Ayuda
[victorkarmaroach@localhost Documentos]$ ./Tubc2
Esto llega a traves de la tuberia
[victorkarmaroach@localhost Documentos]$

```

C Anchura del tabulador: 8 Ln 36, Col 8 INS