



## Programas

Jan Aldahir Velazquez Barrancoía  
Materia: Sistemas Operativos  
Profesora: Monserrat Ariana Huerta

29 de noviembre de 2019

## ÍNDICE

1. Alternancia.....	3
2. Señales.....	
3. Semaforos.....	
4. Comunicación de procesos.....	
5. Sincronización de procesos.....	

```

1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <stdlib.h>
6  #include <time.h>
7  #include <sys/shm.h>
8  #define key 1555
9  #define key2 1553
10
11 void proce(int i,int proceso[],int llegada[],int b[],
12 int rci[],int rcd[],int *turno,int *suma);
13
14 int main(int argc, char * argv[]) {
15
16     int pid1, pid2, pid3, pid4, estado;
17     int p1_finalizado = 0, p2_finalizado = 0, p3_finalizado = 0, p4_finalizado = 0;
18     int proceso[4];
19     int llegada[4];
20     int rci[4], rcd[4];
21     int opc;
22     int b[4];
23     int t1=0;
24     int id, id2;
25     int *suma=NULL;
26     int *turno=NULL;
27     b[0]=0; b[1]=0; b[2]=0; b[3]=0;
28     //se crea una memoria compartida para los turnos
29     id=shmget(key, sizeof(int), IPC_CREAT|SHM_R|SHM_W);
30     //Verifica si se creo la memoria
31     if (id == -1)
32     {
33         perror("shmget:");
34         exit(-1);
35     }
36     //Ata segmento de memoria
37     turno= (int *) shmat (id, NULL, 0);
38     (*turno)=0;
39     //se crea una memoria compartida para modificar region critica
40     id2=shmget(key2, sizeof(int), IPC_CREAT|SHM_R|SHM_W);
41     //Verifica si se creo la memoria
42     if (id2 == -1)
43     {
44         perror("shmget:");
45         exit(-1);
46     }
47     //Ata segmento d memoria
48     suma= (int *) shmat (id2, NULL, 0);
49     (*suma)=0;
50     int i;
51     for(i=0; i<4; i++)
52     {

```

```

53 //se llenan los procesos con tiempo, region y duracion
54 printf("Tiempo para proceso %d: ",i+1);
55 scanf("%d",&proceso[i]);
56 //Pregunta si tiene region critica
57 printf("Region critica\n1)Si\n2)No\n");
58 scanf("%d",&opc);
59 //Pide datos de la region
60 if(opc==1)
61 {
62     llegada[i]=t1;
63     //Pide datos de la region para los procesos que la tienen
64     do
65     {
66         printf("En que tiempo inicia\n");
67         scanf("%d",&rci[i]);
68     }while(rci[i]>proceso[i] && rci[i]<0);
69 //Pide dato de la duracion para los procesos que tienen region critica
70     do
71     {
72         printf("Duracion\n");
73         scanf("%d",&rcd[i]);
74     }while(rcd[i]<0);
75     t1++;
76 }
77 //Para los procesos que no tienen region critica
78 else
79 {
80     rci[i]=-1;
81     llegada[i]=-1;
82 }
83 }
84
85 //Empiezan a ejecutarse los procesos
86 pid1=fork();
87 /* Este es el proceso 4 */
88 if (pid1 == 0)
89 {
90     //mientras el tiempo del proceso sea diferente de 0
91     while(proceso[3]!=0)
92     {
93         printf("Proceso 4\n");
94         proce(3,proceso,llegada,b,rci,rcd,turno,suma);
95     }
96     puts("Proceso #4 finalizado.\n");
97     exit(0);
98 }
99 pid2=fork();
100 /* Este es el proceso #3 */
101 if (pid2 == 0)
102 {
103     // mientras el tiempo del proceso sea diferente de 0
104     while(proceso[2]!=0)
105     {

```

```

106         printf("Proceso 3\n");
107         proce(2, proceso, llegada, b, rci, rcd, turno, suma);
108     }
109     puts("Proceso #3 finalizado.\n");
110     exit(0);
111 }
112 pid3=fork();
113 /* Este es el proceso #2 */
114 if (pid3 == 0)
115 {
116     while(proceso[1]!=0)
117     {
118         printf("Proceso 2\n");
119         proce(1, proceso, llegada, b, rci, rcd, turno, suma);
120     }
121     puts("Proceso #2 finalizado.\n");
122     exit(0);
123 }
124 pid4=fork();
125 /* Este es el proceso #1 */
126 if (pid4 == 0)
127 {
128     while(proceso[0]!=0)
129     {
130         printf("Proceso 1\n");
131         proce(0, proceso, llegada, b, rci, rcd, turno, suma);
132     }
133     puts("Proceso #1 finalizado.\n");
134     exit(0);
135 }
136 if ((pid1 < 0) || (pid2 < 0) || (pid3 < 0) || (pid4 < 0))
137 { // se verifica que se hayan creado bien los procesos
138     printf("No creados...\n");
139     exit(1);
140 }
141 if ((pid1 > 0) && (pid2 > 0) && (pid3 > 0) && (pid4 > 0))
142 { // si los procesos han sido creados bien
143     while((!p1_finalizado) || (!p2_finalizado) || (!p3_finalizado) ||
144         (!p4_finalizado))
145     {
146         int pid;
147         //se espera informacion de los procesos
148         pid = wait(&estado);
149         //se verifica que proceso ha finalizado y se marca
150         if (pid == pid1)
151             p1_finalizado = 1;
152         if (pid == pid2)
153             p2_finalizado = 1;
154         if (pid == pid3)
155             p3_finalizado = 1;
156         if (pid == pid4)
157             p4_finalizado = 1;
158     }

```

```

159     }
160     //Se imprime que han terminado los procesos asi como la region critica
161     puts("Procesos terminados.\n");
162     printf(" %d",*suma);
163 }
164 }
165
166 void proce(int i,int proceso [],int llegada [],int b[],int rci[],int rcd[],
167 int *turno,int *suma)
168 {
169     //se verifica el tiempo de la region con el tiempo del proceso
170     if(rci[i]==proceso[i] && proceso[i]!=0 && b[i]!=1 && rci[i]!=-1)
171     {
172         printf("Intentando entrar a region critica\n");
173         if(*turno==llegada[i])
174         { // se verifica el turno
175             printf("En region critica\n");
176             printf("Tiempo restante region critica: %d\n",rcd[i]);
177             rcd[i]=rcd[i]-1;
178             //se modifica el valor de la region critica
179             *suma=*suma+1;
180             sleep(1);
181             // cuando acabe el tiempo de la region se incrementa turno
182             if(rcd[i]==0)
183             {
184                 printf("Proceso %d\n",i+1);
185                 printf("Saliendo de la region critica\n");
186                 proceso[i]--; // se resta tiempo al proceso
187                 b[i]=1;
188                 *turno=*turno+1; // se incrementa turno
189                 printf("Incrementando turno: %d\n",*turno);
190                 sleep(1);
191             }
192         }
193         else //Si no desocupa regionn critica
194         {
195             printf("Memoria ocupada\n");
196             printf("Lugar: %d\n",llegada[i]);
197             sleep(1);
198         }
199     }
200     else
201     {
202         // se verifica si el proceso ya ejecuto la region critica
203         if(b[i]==1)
204         {
205             printf("Tiempo: %d\n",proceso[i]);
206             proceso[i]--; // se resta tiempo al proceso
207             printf("Este proceso ya ejecuto su region critica\n");
208             sleep(1);
209         }
210         else
211         {

```

```

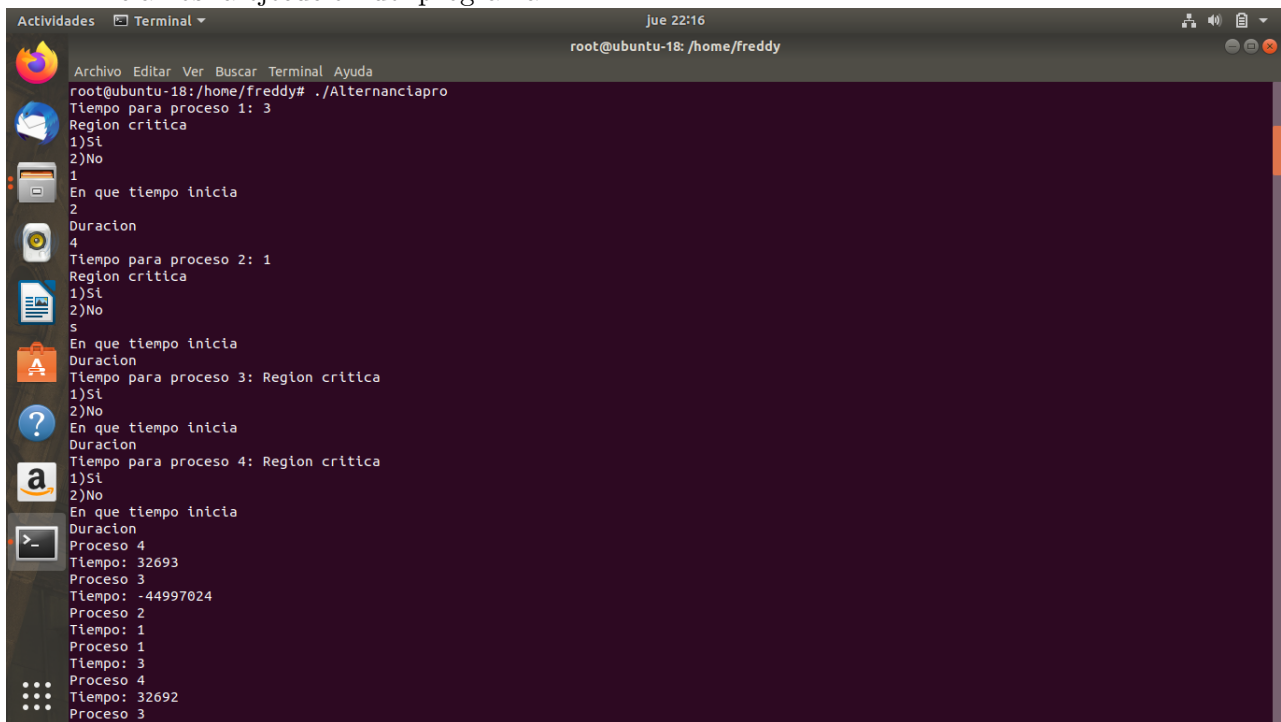
212         printf("Tiempo: %d\n", proceso[i]);
213         proceso[i]--; // se resta tiempo al proceso
214         sleep(1);
215     }
216 }
217 }

```

---

## Ejecución de Programa (capturas)

1.-Iniciamos la ejecución del programa.



```

root@ubuntu-18:/home/freddy# ./Alternanciapro
Tiempo para proceso 1: 3
Region critica
1)Si
2)No
1
En que tiempo inicia
2
Duracion
4
Tiempo para proceso 2: 1
Region critica
1)Si
2)No
s
En que tiempo inicia
Duracion
Tiempo para proceso 3: Region critica
1)Si
2)No
En que tiempo inicia
Duracion
Tiempo para proceso 4: Region critica
1)Si
2)No
En que tiempo inicia
Duracion
Proceso 4
Tiempo: 32693
Proceso 3
Tiempo: -44997024
Proceso 2
Tiempo: 1
Proceso 1
Tiempo: 3
Proceso 4
Tiempo: 32692
Proceso 3

```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/types.h>
5  #include <errno.h>
6  #include <signal.h>
7  #include <fcntl.h>
8  #include <sys/wait.h>
9  #include <ctype.h>
10 FILE *f;
11
12 int num_fork(int n)
13 {
14     int i;
15     for(i=1; i<n; i++)
16     {
17         if(fork()==0)
18         {
19             return (i);
20         }
21     }
22     return (0);
23 }
24 int main()
25 {
26     char nombre[15];
27     int cola_espera[4];
28     int timpo, reg, tim_ini, tim_fin, lugar, espera;
29     char r[10];
30     int n, op, m, hijo;
31     i=num_fork(5);
32     //char *nomarchivo;
33     //nomarchivo="Ids.txt";
34     //remove(nomarchivo);
35     printf("-----Menu-----\n");
36     printf("1. Datos proceso\n");
37     printf("2. Ejecuta\n");
38     printf("3. Mostrar Espera");
39     printf("4. Salir\n");
40     scanf("%d",&op);
41     switch (op)
42     {
43         case i:
44
45     }
46     switch (i)
47     {
48         case 1:
49             printf("Proceso uno Nombre");
50             popen();
51             i=getenv();
52             fprintf(f,"Id1:%d\n",i);
```



```

53                                     pclose ();
54                                     exit (3);
55
56                                     case 2:
57                                     popen ();
58                                     i=getenv ();
59                                     fprintf(f,"Id2:%d\n",i);
60                                     pclose ();
61                                     exit (3);
62                                     case 3:
63                                     popen ();
64                                     i=getenv ();
65                                     fprintf(f,"Id3:%d\n",i);
66                                     pclose ();
67                                     exit (3);
68                                     case 4:
69                                     popen ();
70                                     i=getenv ();
71                                     fprintf(f,"Id4:%d\n",i);
72                                     pclose ();
73                                     exit (3);
74     default:
75         m=main(hijo);
76         m=main(hijo);
77         m=main(hijo);
78         m=main(hijo);
79         f=popen("Ids.txt","r+");
80         //printf("Valor de wwait %d",m);
81         while (fscanf(f,"%s",r) != EOF)
82         {
83             printf("%s\n",r);
84         }
85     }
86 }
87 }

```

---

## Ejecución.



```
Actividades Terminal
jue 23:03
root@ubuntu-18: /home/freddy

freddy@ubuntu-18:~$ su
Contraseña:
root@ubuntu-18:/home/freddy# pico Comunica.c
root@ubuntu-18:/home/freddy# gcc Comunica.c -o Comunica
Comunica.c: In function 'main':
Comunica.c:31:9: error: 'i' undeclared (first use in this function)
    i=num_fork(5);
    ^
Comunica.c:31:9: note: each undeclared identifier is reported only once for each function it appears in
Comunica.c:43:8: error: label at end of compound statement
    case i:
    ^
Comunica.c:50:41: error: too few arguments to function 'popen'
    popen();
    ^~~~~~
In file included from Comunica.c:1:0:
/usr/include/stdio.h:800:14: note: declared here
extern FILE *popen (const char *__command, const char *__modes) __wur;
               ^~~~~~
Comunica.c:51:43: error: too few arguments to function 'getenv'
    i=getenv();
    ^~~~~~
In file included from Comunica.c:2:0:
/usr/include/stdlib.h:631:14: note: declared here
extern char *getenv (const char *__name) __THROW __nonnull ((1)) __wur;
               ^~~~~~
Comunica.c:53:41: error: too few arguments to function 'pclose'
    pclose();
    ^~~~~~
In file included from Comunica.c:1:0:
/usr/include/stdio.h:806:12: note: declared here
extern int pclose (FILE *__stream);
               ^~~~~~
Comunica.c:57:41: error: too few arguments to function 'popen'
    popen();
    ^~~~~~
In file included from Comunica.c:1:0:
/usr/include/stdio.h:800:14: note: declared here
```

El programa de comunica no se puede ejecutar bien aunque se agregen o se cambien librerías, métodos y funciones.

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <unistd.h>
7 #include <signal.h>
8 #include <sys/shm.h> /* shm* */
9
10 #define FILEKEY "/bin/cat"
11 #define KEY 1300
12 // #define NUM 10
13 int NUM=0;
14 pid_t pidL, pidM;
15 //int num=10;
16 void manejador()
17 {
18     printf("Recibi la senal");
19     kill(getpid(), SIGKILL);
20     kill(pidL, SIGKILL);
21     kill(pidM, SIGKILL);
22 }
23
24
25 int main ()
26 {
27     //estructura de la senal
28     struct sigaction act;
29     act.sa_handler = manejador;
30     sigemptyset (&act.sa_mask);
31     act.sa_flags=0;
32     sigaction(SIGALRM,&act,NULL);
33     alarm(3);
34     //Declaracion de variables
35     int fd[2];
36     int key, i;
37     int id_zone;
38     int *buffer;
39     char c;
40     char a;
41     pipe(fd); //Creacion de tuberia
42     /*El proceso padre crea la memoria compartida para la sincronizacion
43     de los procesos*/
44     //LLava e para la memoria compartida
45     key = ftok(FILEKEY, KEY); //crea la llave
46     if (key == -1) //Si no se pudo crear la llave
47     {
48         //Despliega letrero
49         fprintf (stderr, "Error al crear la llave \n");
50         return -1;
51     }
52     //Crea la memoria compartida
```

```

53     id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
54     if (id_zone == -1)//Si no se pudo crear la memoria
55     {
56         //Despliega letrero
57         fprintf (stderr, "Error al crear memoria compartida\n");
58         return -1;
59     }
60     //printf ("ID zone shared memory: %d\n", id_zone);
61     //Imprime el ID de la zona
62     //Ata memoria compartida
63     buffer = shmat (id_zone, (char *)0, 0);
64     if (buffer == NULL)//Si no se puede atar la memoria
65     {
66         //Imprime error
67         fprintf (stderr, "Error al reservar memoria compartida \n");
68         return -1;
69     }
70     //printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
71     //Imprim el puntero
72     //printf("Soy el prceso R \t Mi ID es: %d\n",getpid());
73     pipe(fd);//Creacion de tuberia
74     pidL = fork();//Cracion de proceso L
75     if (pidL == 0)//Si se creo el proceso L
76     {
77         /*Crea memoria compartida para modificar variable compartida, incrementa
78         variable*/
79         //printf("Proceso hijo %d intentando entrar a memoria",getpid());
80         int key, i, id_zone, *buffer;
81         /*LLave para memoria compartida */
82         key = ftok(FILEKEY, KEY);
83         if (key == -1)
84         {
85             fprintf (stderr, "Error al crear llave \n");
86             return -1;
87         }
88         /* Se crea la memoria comartida*/
89         id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
90         if (id_zone == -1)
91         {
92             fprintf (stderr, "Error al crear memoria compartida\n");
93             return -1;
94         }
95         //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
96         /* Declaracion de la memoria compartida */
97         buffer = shmat (id_zone, (char *)0, 0);
98         if (buffer == NULL)
99         {
100             fprintf (stderr, "Error al reservar memoria compartida \n");
101             return -1;
102         }
103
104         //printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
105         while(1)

```

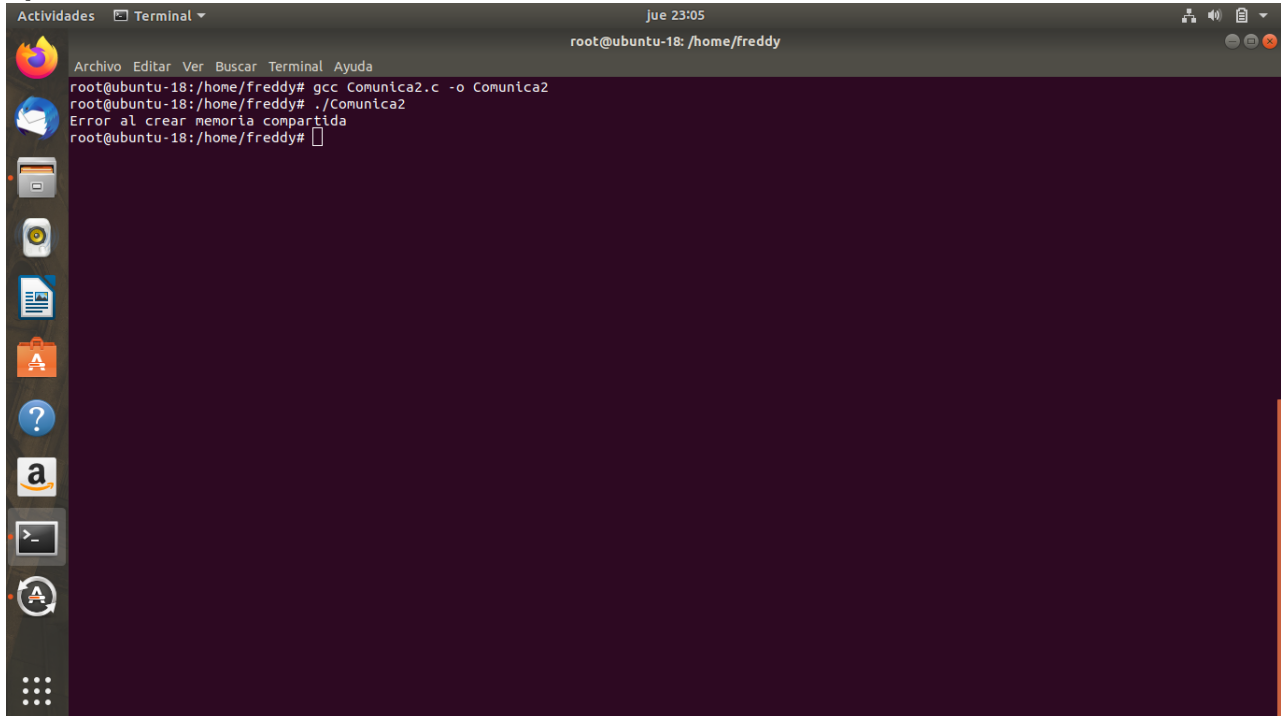
```

106     {
107         //printf(" %d\t",getpid());
108         NUM++;
109         write(fd[1],&NUM,sizeof(int));
110     }
111 }
112 pidM = fork();//Proceso de proceso M
113 if (pidM == 0)
114 {
115     /*Crea memoria compartida para modificar variable compartida,decrementa variable*/
116     //printf("Proceso hijo %d intentando entrar a memoria",getpid());
117     int key, i, id_zone, *buffer;
118     /*LLave para memoria compartida */
119     key = ftok(FILEKEY, KEY);
120     if (key == -1)
121     {
122         fprintf(stderr, "Error al crear llave \n");
123         return -1;
124     }
125     /* Se crea la memoria comartida*/
126     id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
127     if (id_zone == -1)
128     {
129         fprintf (stderr, "Error al crear memoria compartida\n");
130         return -1;
131     }
132     //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
133     /* Declaracion de la memoria compartida */
134     buffer = shmat (id_zone, (char *)0, 0);
135     if (buffer == NULL)
136     {
137         fprintf (stderr, "Error al reservar memoria compartida \n");
138         return -1;
139     }
140     //printf ("Puntero del buffer de la memoria compartida:
141     %p\n", buffer);
142     while(1)
143     {
144         //printf(" %d\t",getpid());
145         NUM--;
146         write(fd[1],&NUM,sizeof(int));
147     }
148 }
149 while (1)//Imprime los datos a pantalla
150 {
151     //printf("Soy el prceso padre \t Mi ID es: %d\n",getpid());
152     read(fd[0],&NUM,sizeof(int));
153     printf(" %d\n",NUM);
154     read(fd[0],&NUM,sizeof(int));
155     printf(" %d\n",NUM);
156 }
157 c = getchar();
158 //libera la memoria compartida

```

```
159     shmddt ((char *)buffer);
160     shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
161     return 0;
162
163 }
```

### Ejecución.



```
root@ubuntu-18:/home/freddy# gcc Comunica2.c -o Comunica2
root@ubuntu-18:/home/freddy# ./Comunica2
Error al crear memoria compartida
root@ubuntu-18:/home/freddy#
```

En el programa comunica2 se ejecuta y nos da como salida un error para crear memoria compartida.

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <sys/shm.h> /* shm* */
7 #include <unistd.h>
8
9
10 #define FILEKEY "/bin/cat"
11 #define KEY 1300
12 #define MAXBUF 10
13
14
15
16 int main ()
17 {
18     int key, i;
19     int id_zone;
20     int *buffer;
21     char c;
22     pid_t pid1;
23     //LLava e para la memoria compartida
24     key = ftok(FILEKEY, KEY);
25     if (key == -1)
26     {
27         fprintf (stderr, "Error al crear la llave \n");
28         return -1;
29     }
30     //Crea la memoria compartida
31     id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
32     if (id_zone == -1)
33     {
34         fprintf (stderr, "Error al crear memoria compartida\n");
35         return -1;
36         return -1;
37     }
38     printf ("ID zone shared memory: %d\n", id_zone);
39     //Declarar memoria compartida
40     buffer = shmat (id_zone, (char *)0, 0);
41     if (buffer == NULL)
42     {
43         fprintf (stderr, "Error al reservar memoria compartida \n");
44         return -1;
45     }
46     printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
47     for (i = 0; i < MAXBUF; i++)
48     {
49         buffer[i] = i;
50     }
51     pid1=fork();
52     //Crea un hijo para entrar en memria compartida.
```

```

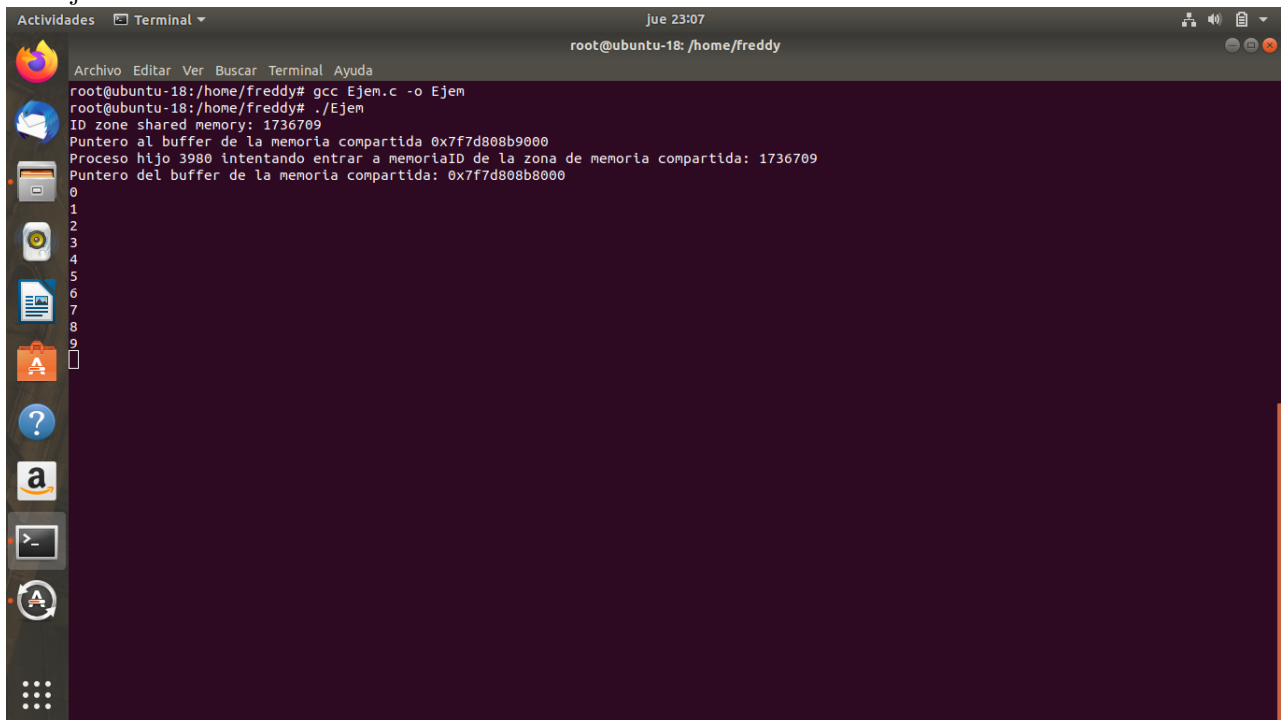
53  if (pid1 == 0)
54  {
55      printf("Proceso hijo %d intentando entrar a memoria", getpid());
56      int key, i, id_zone, *buffer;
57      /*LLave para memoria compartida */
58      key = ftok(FILEKEY, KEY);
59      if (key == -1)
60      {
61          fprintf (stderr, "Error al crear llave \n");
62          return -1;
63      }
64      /* Se crea la memoria comartida*/
65      id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
66      if (id_zone == -1)
67      {
68          fprintf (stderr, "Error al crear memoria compartida\n");
69          return -1;
70      }
71
72      printf ("ID de la zona de memoria compartida: %d\n", id_zone);
73      /* Declaracion de la memoria compartida */
74      buffer = shmat (id_zone, (char *)0, 0);
75      if (buffer == NULL)
76      {
77          fprintf (stderr, "Error al reservar memoria compartida \n");
78          return -1;
79      }
80
81      printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
82      /* Escribe los valores a la memoria */
83      for (i = 0; i < MAXBUF; i++)
84      {
85          printf ("%d\n", buffer[i]);
86      }
87      //printf("Proceso hijo %d intentando entrar a memoria", getpid());
88  }
89      c = getchar();
90      //libera la memoria compartida
91      shmdt ((char *)buffer);
92      shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
93      return 0;
94  }

```

---



## Ejecución



```
root@ubuntu-18:/home/freddy# gcc Ejem.c -o Ejem
root@ubuntu-18:/home/freddy# ./Ejem
ID zone shared memory: 1736709
Puntero al buffer de la memoria compartida 0x7f7d808b9000
Proceso hijo 3980 intentando entrar a memoriaID de la zona de memoria compartida: 1736709
Puntero del buffer de la memoria compartida: 0x7f7d808b8000
0
1
2
3
4
5
6
7
8
9
□
```

Nuestro programa ejecuta la funcion de crear un puntero en el buffer para la memoria compartida

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <sys/shm.h> /* shm* */
7
8 #define FILEKEY "/bin/cat"
9 #define KEY 1300
10 #define MAXBUF 10
11
12
13
14 int main ()
15 {
16     int key, i;
17     int id_zone;
18     int *buffer;
19     char c;
20     pid_t pid1;
21     //LLava e para la memoria compartida
22     key = ftok(FILEKEY, KEY);
23     if (key == -1)
24     {
25         fprintf (stderr, "Error al crear la llave \n");
26         return -1;
27     }
28     //Crea la memoria compartida
29     id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
30     if (id_zone == -1)
31     {
32         fprintf (stderr, "Error al crear memoria compartida\n");
33         return -1;
34         return -1;
35     }
36     printf ("ID zone shared memory: %d\n", id_zone);
37     //Declarar memoria compartida
38     buffer = shmat (id_zone, (char *)0, 0);
39     if (buffer == NULL)
40     {
41         fprintf (stderr, "Error al reservar memoria compartida \n");
42         return -1;
43     }
44     printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
45     for (i = 0; i < MAXBUF; i++)
46     {
47         buffer[i] = i;
48     }
49     pid1=fork();
50     //Crea un hijo para entrar en memria compartida.
51     if (pid1 == 0)
52     {
```

```

53     printf("Proceso hijo %d intentando entrar a memoria",getpid());
54     int key, i, id_zone, *buffer;
55     /*LLave para memoria compartida */
56     key = ftok(FILEKEY, KEY);
57     if (key == -1)
58     {
59         fprintf (stderr, "Error al crear llave \n");
60         return -1;
61     }
62     /* Se crea la memoria comartida*/
63     id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
64     if (id_zone == -1)
65     {
66         fprintf (stderr, "Error al crear memoria compartida\n");
67         return -1;
68     }
69
70     printf ("ID de la zona de memoria compartida: %d\n", id_zone);
71     /* Declaracion de la memoria compartida */
72     buffer = shmat (id_zone, (char *)0, 0);
73     if (buffer == NULL)
74     {
75         fprintf (stderr, "Error al reservar memoria compartida \n");
76         return -1;
77     }
78
79     printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
80     /* Escribe los valores a la memoria */
81     for (i = 0; i < MAXBUF; i++)
82     {
83         printf ("%d\n", buffer[i]);
84     }
85     //printf("Proceso hijo %d intentando entrar a memoria",getpid());
86 }
87     c = getchar();
88     //libera la memoria compartida
89     shmdt ((char *)buffer);
90     shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
91     return 0;
92 }

```

---

Ejecución

```
Actividades Terminal Jue 23:09 root@ubuntu-18: /home/freddy
Archivo Editar Ver Buscar Terminal Ayuda
root@ubuntu-18:/home/freddy# pico Ejen.men.c
root@ubuntu-18:/home/freddy# gcc Ejen.men.c -o Ejen.men
Ejen.men.c: In function 'main':
Ejen.men.c:49:7: warning: implicit declaration of function 'fork' [-Wimplicit-function-declaration]
  pid=fork();
  ~~~~~
Ejen.men.c:53:56: warning: implicit declaration of function 'getpid'; did you mean 'fgetpos'? [-Wimplicit-function-declaration]
  printf("Proceso hijo %d intentando entrar a memoria",getpid());
  ~~~~~
root@ubuntu-18:/home/freddy# ./Ejen.men
ID zone shared memory: 1769477
Puntero al buffer de la memoria compartida 0x7f4159dc8000
Proceso hijo 4024 intentando entrar a memoriaID de la zona de memoria compartida: 1769477
Puntero del buffer de la memoria compartida: 0x7f4159dc7000
0
1
2
3
4
5
6
7
8
9
[]
```

Puntero de memoria compartida, en la cual el proceso hijo intenta entrar a la memoria compartida

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <unistd.h>
7 #include <signal.h>
8 #include <sys/shm.h> /* shm* */
9
10 #define FILEKEY "/bin/cat"
11 #define KEY 1300
12 // #define NUM 10
13 int NUM=0;
14 pid_t pidL, pidM;
15 //int num=10;
16 void manejador()
17 {
18     printf("Recibi la senal");
19     kill(getpid(), SIGKILL);
20     kill(pidL, SIGKILL);
21     kill(pidM, SIGKILL);
22 }
23
24
25 int main ()
26 {
27     //estructura de la senal
28     struct sigaction act;
29     act.sa_handler = manejador;
30     sigemptyset (&act.sa_mask);
31     act.sa_flags=0;
32     sigaction(SIGALRM,&act, NULL);
33     alarm(3);
34     //Declaracion de variables
35     int fd[2];
36     int key, i;
37     int id_zone;
38     int *buffer;
39     char c;
40     char a;
41     pipe(fd); //Creacion de tuberia
42     /*El proceso padre crea la memoria compartida para la sincronizacion
43     de los procesos*/
44     //LLava e para la memoria compartida
45     key = ftok(FILEKEY, KEY); //crea la llave
46     if (key == -1) //Si no se pudo crear la llave
47     {
48         //Despliega letrero
49         fprintf (stderr, "Error al crear la llave \n");
50         return -1;
51     }
52     //Crea la memoria compartida
```

```

53     id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
54     if (id_zone == -1)//Si no se pudo crear la memoria
55     {
56         //Despliega letrero
57         fprintf (stderr, "Error al crear memoria compartida\n");
58         return -1;
59     }
60     //printf ("ID zone shared memory: %d\n", id_zone);
61     //Imprime el ID de la zona
62     //Ata memoria compartida
63     buffer = shmat (id_zone, (char *)0, 0);
64     if (buffer == NULL)//Si no se puede atar la memoria
65     {
66         //Imprime error
67         fprintf (stderr, "Error al reservar memoria compartida \n");
68         return -1;
69     }
70     //printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
71     //Imprim el puntero
72     //printf("Soy el prceso R \t Mi ID es: %d\n",getpid());
73     pipe(fd);//Creacion de tuberia
74     pidL = fork();//Cracion de proceso L
75     if (pidL == 0)//Si se creo el proceso L
76     {
77         /*Crea memoria compartida para modificar variable compartida,
78         incrementa variable*/
79         //printf("Proceso hijo %d intentando entrar a memoria",getpid());
80         int key, i, id_zone, *buffer;
81         /*LLave para memoria compartida */
82         key = ftok(FILEKEY, KEY);
83         if (key == -1)
84         {
85             fprintf (stderr, "Error al crear llave \n");
86             return -1;
87         }
88         /* Se crea la memoria comartida*/
89         id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
90         if (id_zone == -1)
91         {
92             fprintf (stderr, "Error al crear memoria compartida\n");
93             return -1;
94         }
95         //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
96         /* Declaracion de la memoria compartida */
97         buffer = shmat (id_zone, (char *)0, 0);
98         if (buffer == NULL)
99         {
100             fprintf (stderr, "Error al reservar memoria compartida \n");
101             return -1;
102         }
103
104         //printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
105         while(1)

```

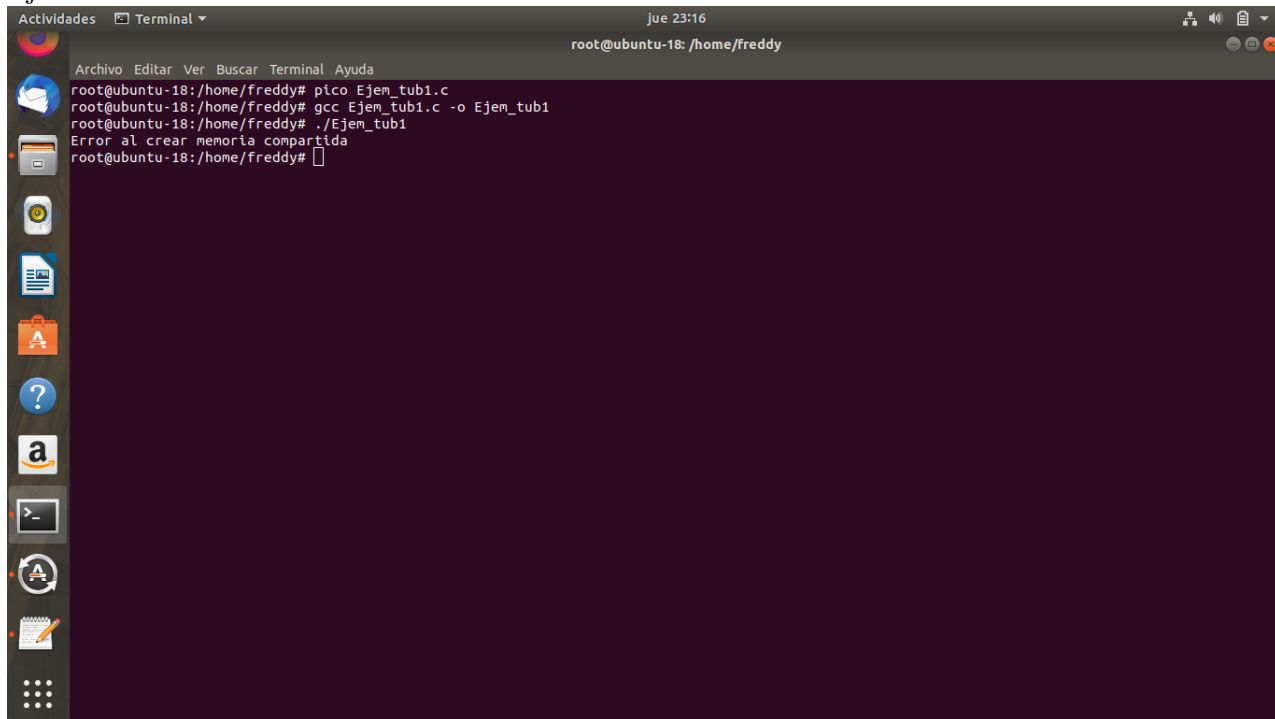
```

106     {
107         //printf(" %d\t",getpid());
108         NUM++;
109         write(fd[1],&NUM,sizeof(int));
110     }
111 }
112 pidM = fork();//Proceso de proceso M
113 if (pidM == 0)
114 {
115     /*Crea memoria compartida para modificar variable compartida,
116     decrementa variable*/
117     //printf("Proceso hijo %d intentando entrar a memoria",getpid());
118     int key, i, id_zone, *buffer;
119     /*LLave para memoria compartida */
120     key = ftok(FILEKEY, KEY);
121     if (key == -1)
122     {
123         fprintf (stderr, "Error al crear llave \n");
124         return -1;
125     }
126     /* Se crea la memoria comartida*/
127     id_zone = shmget (key, sizeof(int)*NUM, 0777 | IPC_CREAT);
128     if (id_zone == -1)
129     {
130         fprintf (stderr, "Error al crear memoria compartida\n");
131         return -1;
132     }
133     //printf ("ID de la zona de memoria compartida: %d\n", id_zone);
134     /* Declaracion de la memoria compartida */
135     buffer = shmat (id_zone, (char *)0, 0);
136     if (buffer == NULL)
137     {
138         fprintf (stderr, "Error al reservar memoria compartida \n");
139         return -1;
140     }
141     //printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
142     while(1)
143     {
144         //printf(" %d\t",getpid());
145         NUM--;
146         write(fd[1],&NUM,sizeof(int));
147     }
148 }
149 while (1)//Imprime los datos a pantalla
150 {
151     //printf("Soy el prceso padre \t Mi ID es: %d\n",getpid());
152     read(fd[0],&NUM,sizeof(int));
153     printf(" %d\n",NUM);
154     read(fd[0],&NUM,sizeof(int));
155     printf(" %d\n",NUM);
156 }
157 c = getchar();
158 //libera la memoria compartida

```

```
159     shmdt ((char *)buffer);
160     shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
161     return 0;
162
163 }
```

## Ejecución



```
root@ubuntu-18:/home/freddy# pico Ejem_tub1.c
root@ubuntu-18:/home/freddy# gcc Ejem_tub1.c -o Ejem_tub1
root@ubuntu-18:/home/freddy# ./Ejem_tub1
Error al crear memoria compartida
root@ubuntu-18:/home/freddy#
```

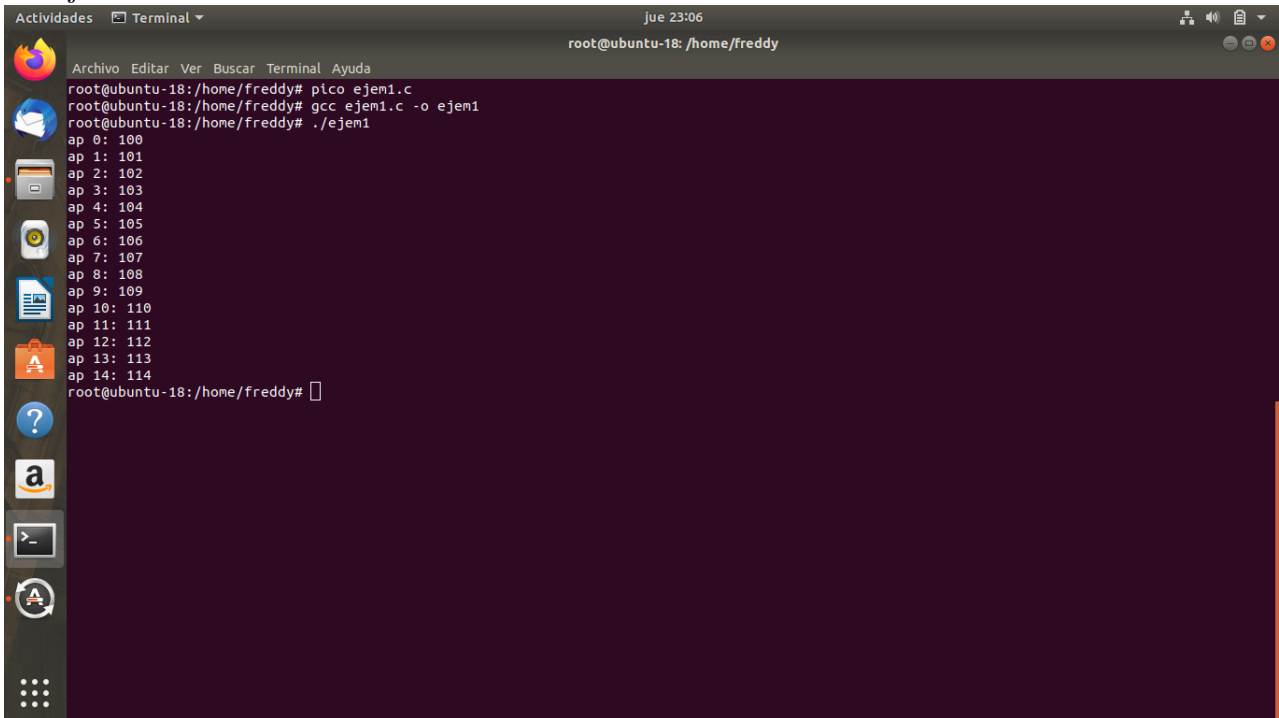
Error al crear la memoria compartida mediante un programa tipo tubería



```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <sys/shm.h>
5 #include <sys/wait.h>
6 #include <unistd.h>
7
8 int main()
9 {
10     pid_t pid1, pid2;
11     pid1=fork();
12     if(pid1>0){
13         pid2=fork();
14         if(pid2>0){ //Padre C
15             int id,*ap,status,c[15],suma,i;
16             suma=0;
17             wait(&status);
18             wait(&status);
19             id=shmget(ftok(".", '&'), sizeof(c), IPC_CREAT);
20             ap=shmat(id, 0, 0);
21             for(i=0; i<15; i++){
22                 printf("ap %d: %d\n", i, *ap+i);
23                 suma+=(*ap+i);
24             }
25             shmdt(ap);
26             shmctl(id, IPC_RMID, NULL);
27             id=shmget(ftok(".", '%'), sizeof(int), IPC_CREAT);
28             ap=shmat(id, 0, 0);
29             *ap=suma;
30         }
31         if(pid2==0){ //Hijo B
32             int id,*ap,B[15],i;
33             id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
34             ap=shmat(id, 0, 0);
35             for(i=1; i<=15; i+=2){
36                 //*(ap+i)=i;
37                 *(ap+i)=(100);
38             }
39         }
40     }
41     if(pid1==0){ //Hijo A
42         int id,*ap,B[15],i;
43         id=shmget(ftok(".", '&'), sizeof(B), IPC_CREAT);
44         ap=shmat(id, 0, 0);
45         for(i=0; i<15; i+=2){
46             //*(ap+i)=i;
47             *(ap+i)=(100);
48         }
49     }
50 }
```

---

## Ejecución



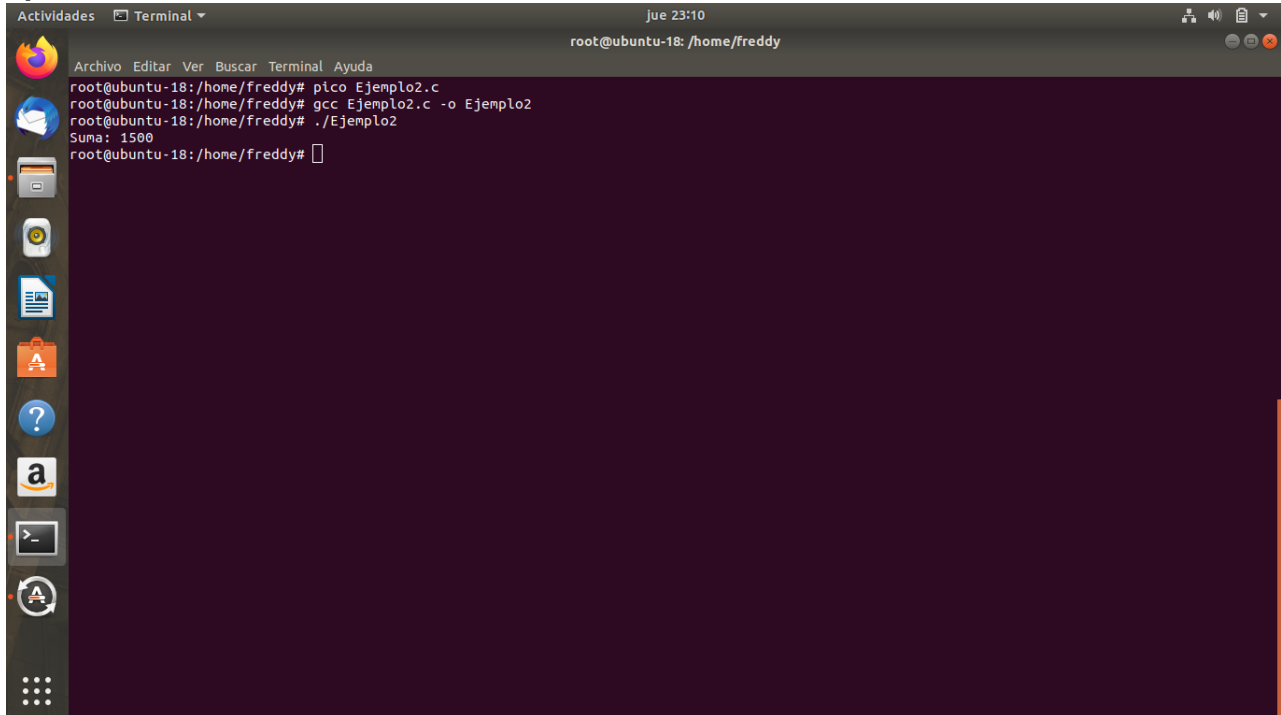
```
root@ubuntu-18:/home/freddy# pico ejem1.c
root@ubuntu-18:/home/freddy# gcc ejem1.c -o ejem1
root@ubuntu-18:/home/freddy# ./ejem1
ap 0: 100
ap 1: 101
ap 2: 102
ap 3: 103
ap 4: 104
ap 5: 105
ap 6: 106
ap 7: 107
ap 8: 108
ap 9: 109
ap 10: 110
ap 11: 111
ap 12: 112
ap 13: 113
ap 14: 114
root@ubuntu-18:/home/freddy#
```

Creando padre e hijos mediante tuberías

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <sys/shm.h>
5
6 int main(){
7     int id, *ap;
8     id=shmget(ftok(".", '%'), sizeof(int), IPC_CREAT);
9     ap=shmat(id, 0, 0);
10    printf("Suma: %d\n", *ap);
11 }
```

---

### Ejecución



```
root@ubuntu-18: /home/freddy
root@ubuntu-18:/home/freddy# pico Ejemplo2.c
root@ubuntu-18:/home/freddy# gcc Ejemplo2.c -o Ejemplo2
root@ubuntu-18:/home/freddy# ./Ejemplo2
Suma: 1500
root@ubuntu-18:/home/freddy#
```

Este programa calcula la suma mediante funcion shmget

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <signal.h>
5 #include <unistd.h>
6
7 void GeneraPares(int tuberia, int t1, int t2)
8 {
9     int i=0;
10    char testigo;
11    /*i es el numero par que se genera*/
12    /*Se genera el primer lugar el 0*/
13    write(tuberia, &i, sizeof(const void));
14    /*Sede el turno a p2*/
15    write(t1, testigo, sizeof(const void));
16    for(i=0; i < 2000; i=i+2)
17    {
18        /*Espera el turno*/
19        read(t2, testigo, sizeof(void));
20        /*Inserta el siguiente numero par*/
21        write(tuberia, &i, sizeof(const void));
22        /*Cede el turno a p2*/
23        write(t1, &testigo, sizeof(const void));
24    }
25    return;
26 }
27
28
29 void GeneraImpares(int tuberia, int t1, int t2)
30 {
31     int i=0;
32     char testigo;
33     /*i es el numero impar que se genera*/
34     for(i=1; i<2000; i=i+2)
35     {
36         /*Espera el turno */
37         read(t1, &testigo, sizeof(void));
38         /*Inserta el siguiente numero par*/
39         write(tuberia, &i, sizeof(const void));
40         /*Cede el turno al p1*/
41         write(t2, &testigo, sizeof(const void));
42     }
43     return;
44 }
45
46 void ConsumeNumeros(int tuberia)
47 {
48     int i;
49     while(read(tuberia, &i, sizeof(int)) > 0)
50     {
51         /*Escribe el caracter*/
52         printf("%d\n", i);
```

```

53     }
54     return;
55 }
56
57 int main ()
58 {
59     pid_t pid1, pid2;
60     /*Tuberia ocupada como sistema de comunicacion*/
61     /*Entre los tres procesos*/
62     int tuberia[2];
63
64     /*Tuberias utilizadas para sincronizar a los procesos p1 y p2*/
65     int t1[2], t2[2];
66     /*El proceso padre sera el que cree la tuberia*/
67     if (pipe(tuberia) < 0)
68     {
69         perror("No se puede crear la tuberia");
70         exit(0);
71     }
72     if (pipe(t1) < 0)
73     {
74         perror("No se puede crear la tuberia");
75
76         exit(0);
77     }
78     if (pipe(t2) < 0)
79     {
80         perror("No se puede crear la tuberia");
81     }
82     /*Se crea el proceso p1*/
83     switch(pid1=fork())
84     {
85         case -1:
86             perror("No se puede crear el proceso");
87             /*Se cierra la pipe*/
88             close(tuberia[0]);
89             close(tuberia[1]);
90             close(t1[0]);
91             close(t1[1]);
92             close(t2[0]);
93             close(t2[1]);
94             exit(0);
95         case 0: /*Proceso hijo proceso p1*/
96             /*Cierra el descriptor de lectura de la pipe*/
97             close(tuberia[0]);
98             /*Este proceso lee de t1 y escribe en t2*/
99             close(t1[1]);
100             close(t2[0]);
101             GeneralImpares(tuberia[1], t1[0], t2[1]);
102             /*El proceso acaba los descriptors*/
103             close(tuberia[1]);
104             close(t1[0]);
105             close(t2[1]);

```

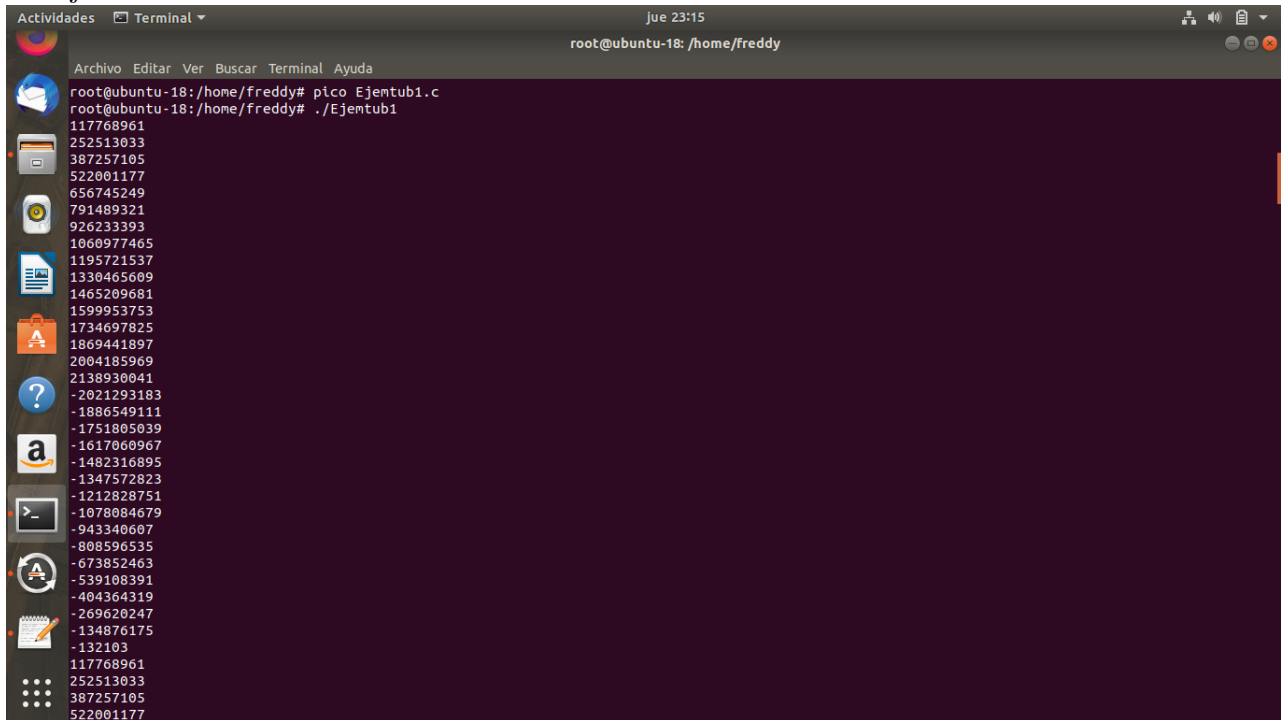
```

106         break;
107     default:
108         /*El proceso padre crea ahora el proceso p2*/
109         switch(pid2 = fork())
110         {
111             case -1:
112                 perror("Error al crear el proceso p2");
113                 /*Se cierra la pipe*/
114                 close(tuberia[0]);
115                 close(tuberia[1]);
116                 close(t1[0]);
117                 close(t1[1]);
118                 close(t2[0]);
119                 close(t2[1]);
120                 /*Se mata el proceso anterior*/
121                 kill(pid1, SIGKILL);
122                 exit(0);
123             case 0: /*Proceso hijo p2*/
124                 /*lee de la tuberia*/
125                 /*Cierra el descriptor de escritura*/
126                 close(tuberia[1]);
127                 /*no necesita t1 ni t2*/
128                 close(t1[0]);
129                 close(t1[1]);
130                 close(t2[0]);
131                 close(t2[1]);
132                 ConsumeNumeros(tuberia[0]);
133                 close(tuberia[0]);
134                 exit(0);
135                 break;
136             default: /*Procesoo padre*/
137                 /*Escribe en la tuberia*/
138                 /*Cierra el descriptor de lectura*/
139                 close(tuberia[0]);
140                 /*Este proceso lee de t2*/
141                 /*y escribe en t1.Cierra lo que no necesita*/
142                 close(t1[0]);
143                 close(t2[1]);
144                 GeneraImpares(tuberia[1], t1[1], t2[0]);
145                 /*El proceso cierra los descriptors*/
146                 close(tuberia[1]);
147                 close(t1[1]);
148                 close(t2[0]);
149         }
150     }
151 }

```

---

## Ejecución



```
root@ubuntu-18:/home/freddy# pico Ejentub1.c
root@ubuntu-18:/home/freddy# ./Ejentub1
117768961
252513033
387257105
522001177
656745249
791489321
926233393
1060977465
1195721537
1330465609
1465209681
1599953753
1734697825
1869441897
2004185969
2138930041
-2021293183
-1806549111
-1751805039
-1617060967
-1482316895
-1347572823
-1212828751
-1078084679
-943340607
-808596535
-673852463
-539108391
-404364319
-269620247
-134876175
-132103
117768961
252513033
387257105
522001177
```

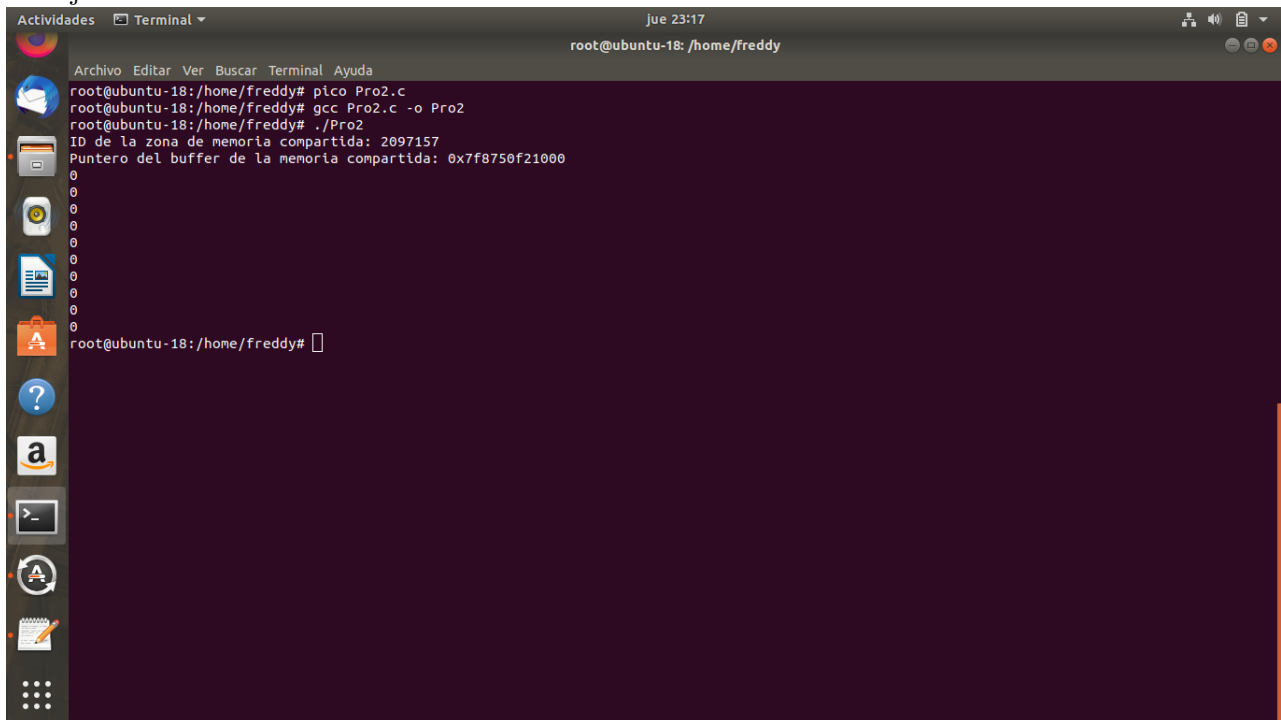
Este programa crea tuberías, los manda a llamar, pero si esta ocupada la tubería llama a otra, se utiliza para sincronizar procesos entre tuberías, al igual crea procesos padre e hijos y puede matar los procesos creados.

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <sys/shm.h> /* shm* */
7
8 #define FILEKEY "/bin/cat"
9 #define KEY 1300
10 #define MAXBUF 10
11
12 int main ()
13 {
14     int key, i, id_zone, *buffer;
15     /*LLave para memoria compartida */
16     key = ftok(FILEKEY, KEY);
17     if (key == -1)
18     {
19         fprintf (stderr, "Error al crear llave \n");
20         return -1;
21     }
22
23     /* Se crea la memoria comartida*/
24     id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
25     if (id_zone == -1)
26     {
27         fprintf (stderr, "Error al crear memoria compartida\n");
28         return -1;
29     }
30
31     printf ("ID de la zona de memoria compartida: %d\n", id_zone);
32     /* Declaracion de la memoria compartida */
33     buffer = shmat (id_zone, (char *)0, 0);
34     if (buffer == NULL)
35     {
36         fprintf (stderr, "Error al reservar memoria compartida \n");
37         return -1;
38     }
39
40
41     printf ("Puntero del buffer de la memoria compartida: %p\n", buffer);
42     /* Escribe los valores a la memoria */
43     for (i = 0; i < MAXBUF; i++)
44     {
45         printf ("%d\n", buffer[i]);
46     }
47     return 0;
48 }
```

---



## Ejecución



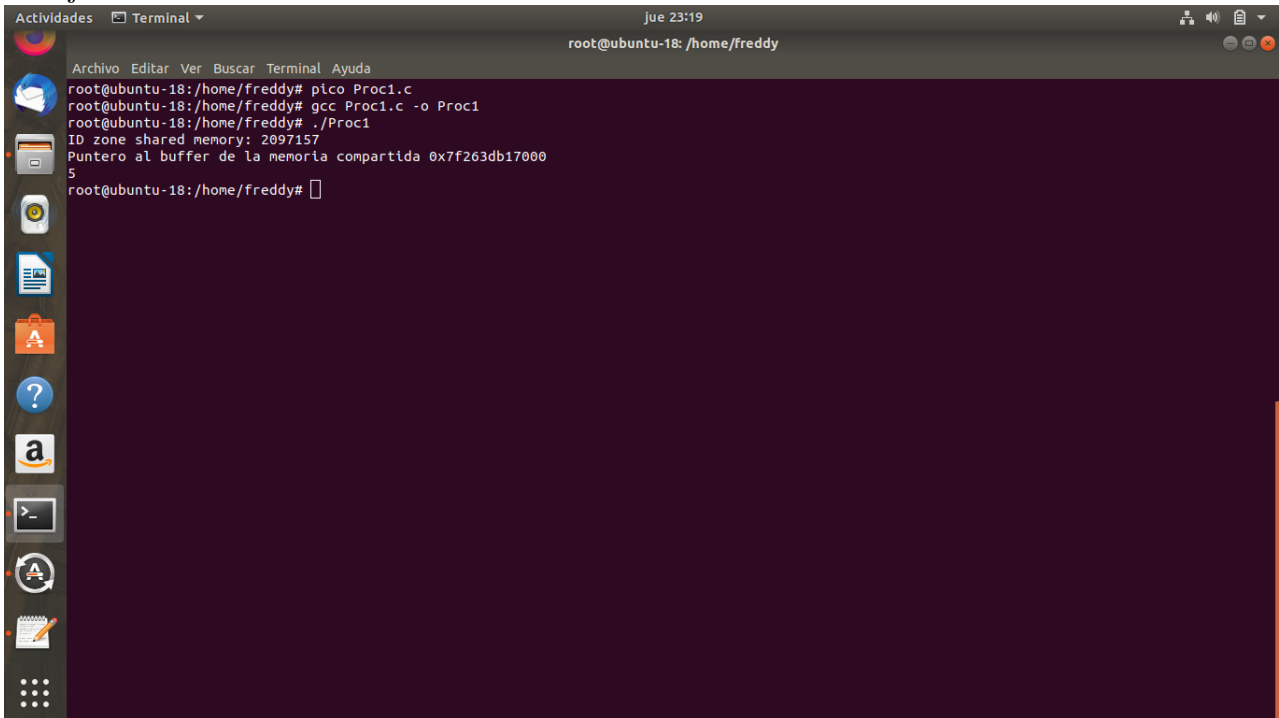
```
root@ubuntu-18:/home/freddy# pico Pro2.c
root@ubuntu-18:/home/freddy# gcc Pro2.c -o Pro2
root@ubuntu-18:/home/freddy# ./Pro2
ID de la zona de memoria compartida: 2097157
Puntero del buffer de la memoria compartida: 0x7f8750f21000
0
0
0
0
0
0
0
0
0
0
root@ubuntu-18:/home/freddy#
```

Creando llave para memoria compartida entrando a la zona de memoria

```
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <sys/ipc.h>
4 #include <string.h>
5 #include <errno.h>
6 #include <sys/shm.h> /* shm* */
7
8 #define FILEKEY "/bin/cat"
9 #define KEY 1300
10 #define MAXBUF 10
11
12
13
14 int main ()
15 {
16     int key, i;
17     int id_zone;
18     int *buffer;
19     char c;
20     //LLava e para la memoria compartida
21     key = ftok(FILEKEY, KEY);
22     if (key == -1)
23     {
24         fprintf (stderr, "Error al crear la llave \n");
25         return -1;
26     }
27     //Crea la memoria compartida
28     id_zone = shmget (key, sizeof(int)*MAXBUF, 0777 | IPC_CREAT);
29     if (id_zone == -1)
30     {
31         fprintf (stderr, "Error al crear memoria compartida\n");
32         return -1;
33     }
34     printf ("ID zone shared memory: %a\n", id_zone);
35     //Declarar memoria compartida
36     buffer = shmat (id_zone, (char *)0, 0);
37     if (buffer == NULL)
38     {
39         fprintf (stderr, "Error al reservar memoria compartida \n");
40         return -1;
41     }
42     printf ("Puntero al buffer de la memoria compartida %p\n", buffer);
43     for (i = 0; i < MAXBUF; i++)
44     {
45         buffer[i] = i;
46     }
47     c = getchar();
48     //libera la memoria compartida
49     shmdt ((char *)buffer);
50     shmctl (id_zone, IPC_RMID, (struct shmid_ds *)NULL);
51     return 0;
52 }
```

---

## Ejecución



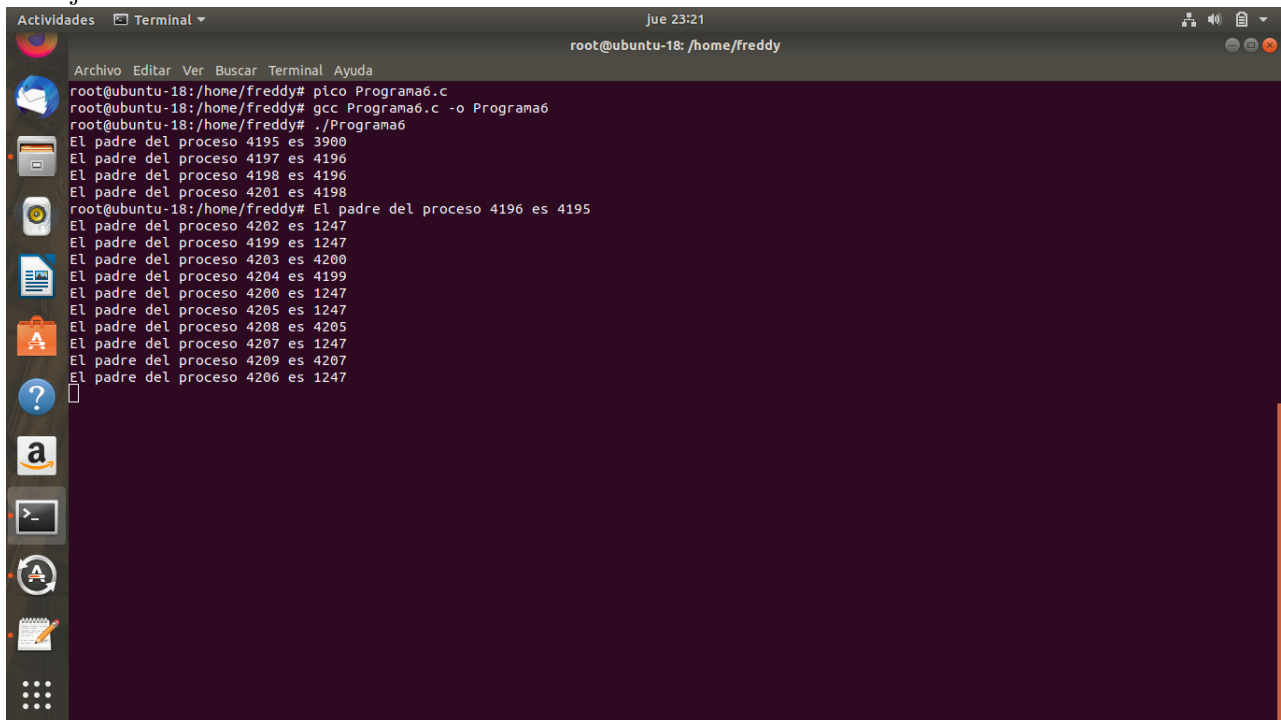
```
root@ubuntu-18:/home/freddy# pico Proc1.c
root@ubuntu-18:/home/freddy# gcc Proc1.c -o Proc1
root@ubuntu-18:/home/freddy# ./Proc1
ID zone shared memory: 2097157
Puntero al buffer de la memoria compartida 0x7f263db17000
5
root@ubuntu-18:/home/freddy#
```

Creando llave para memoria compartida entrando a la zona de memoria, pero crea y declara la memoria compartida, y libera la memoria

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/types.h>
4
5  int main(){
6      pid_t pid;
7      int n=3,i;
8      for (i=0;i<n;i++)
9          {
10             pid=fork ();
11             if (pid!=0)
12                 break;
13             else
14                 pid=fork ();
15         }
16     printf("El padre del proceso %d es %d\n",getpid(),getppid());
17 }
```

---

## Ejecución



```
root@ubuntu-18:/home/freddy# pico Programa6.c
root@ubuntu-18:/home/freddy# gcc Programa6.c -o Programa6
root@ubuntu-18:/home/freddy# ./Programa6
El padre del proceso 4195 es 3900
El padre del proceso 4197 es 4196
El padre del proceso 4198 es 4196
El padre del proceso 4201 es 4198
root@ubuntu-18:/home/freddy# El padre del proceso 4196 es 4195
El padre del proceso 4202 es 1247
El padre del proceso 4199 es 1247
El padre del proceso 4203 es 4200
El padre del proceso 4204 es 4199
El padre del proceso 4200 es 1247
El padre del proceso 4205 es 1247
El padre del proceso 4208 es 4205
El padre del proceso 4207 es 1247
El padre del proceso 4209 es 4207
El padre del proceso 4206 es 1247
```

Creando un proceso padre para los demas procesos

```
1  #include <time.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void delay(unsigned int mseconds)
6  {
7      clock_t goal = mseconds + clock();
8      while (goal > clock());
9  }
10
11
12 struct proceso
13 {
14     int prioridad;
15     char nombre[10];
16     int tiempo;
17     struct proceso *izq;
18     struct proceso *der;
19 };
20
21 int main()
22 {
23     struct proceso *nodo, *p, *q, *nuevo, *cabecera;
24     int n, i, prioridad, nombre, tiempo;
25     printf("Programa que crea procesos\n\n");
26
27     cabecera = (struct proceso *) malloc(sizeof(struct proceso));
28     cabecera->prioridad = 0;
29     cabecera->nombre[i] = ' ';
30     cabecera->tiempo = 0;
31     cabecera->izq = NULL;
32     cabecera->der = NULL;
33
34     do
35     {
36         p = (struct proceso *) malloc(sizeof(struct proceso));
37         printf("Deme la prioridad: ");
38         scanf("%d", &p->prioridad);
39         printf("Nombre del proceso ");
40         scanf("%s", &p->nombre);
41         printf("Tiempo: ");
42         scanf("%d", &p->tiempo);
43
44         if (cabecera->der == NULL)
45         {
46             p->der = NULL;
47             p->izq = cabecera;
48             cabecera->der = p;
49         }
50         else
51         {
52             q = cabecera;
```

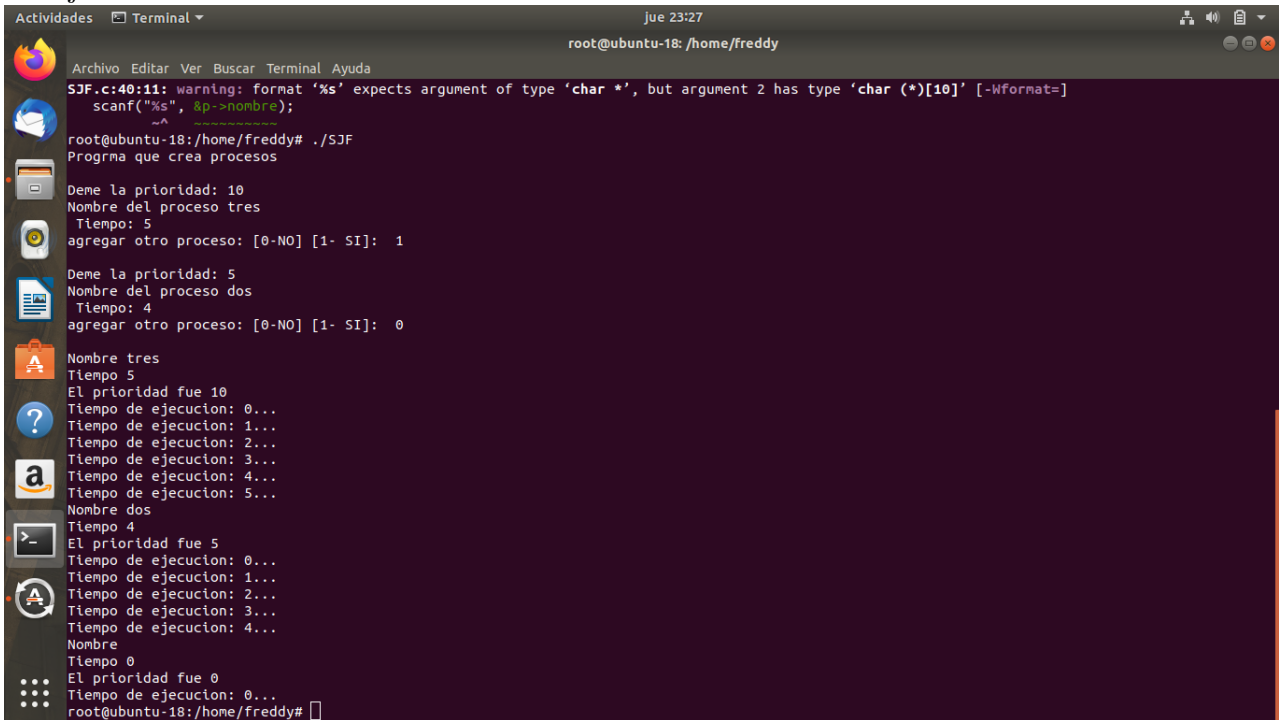
```

53         //p->prioridad=prioridad;
54         while((((q->der)!=NULL) && ((p->prioridad > (q->der)->prioridad))))
55             q=q->der;
56
57         if (q->der==NULL)
58         {
59             q->der=p;
60             p->der=NULL;
61             p->izq=q;
62         }
63         else
64         {
65             p->der=q->der;
66             q->der=p;
67             p->izq=q;
68             p->der->izq=p;
69         }
70     }
71     printf("agregar otro proceso: [0-NO] [1-SI]: ");
72     scanf("%d", &n);
73     printf("\n");
74 } while(n!=0);
75
76 while(p->der!=NULL)
77     p=p->der;
78
79 while(p)
80 {
81     printf("Nombre %s\n", p->nombre);
82     printf("Tiempo %d\n", p->tiempo);
83     printf("El prioridad fue %d\n", p->prioridad);
84     for (i=0; i<=p->tiempo; i++)
85     {
86         printf("Tiempo de ejecucion: %d...\n", i);
87         delay(1000);
88     }
89     p=p->izq;
90 }
91 }

```

---

## Ejecución



```
root@ubuntu-18: /home/freddy
Archivo Editar Ver Buscar Terminal Ayuda
SJT.c:40:11: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'char (*)[10]' [-Wformat=]
scanf("%s", &p->nombre);
          ^A
root@ubuntu-18:/home/freddy# ./SJF
Programa que crea procesos

Deme la prioridad: 10
Nombre del proceso tres
Tiempo: 5
agregar otro proceso: [0-N0] [1- SI]: 1

Deme la prioridad: 5
Nombre del proceso dos
Tiempo: 4
agregar otro proceso: [0-N0] [1- SI]: 0

Nombre tres
Tiempo 5
El prioridad fue 10
Tiempo de ejecucion: 0...
Tiempo de ejecucion: 1...
Tiempo de ejecucion: 2...
Tiempo de ejecucion: 3...
Tiempo de ejecucion: 4...
Tiempo de ejecucion: 5...
Nombre dos
Tiempo 4
El prioridad fue 5
Tiempo de ejecucion: 0...
Tiempo de ejecucion: 1...
Tiempo de ejecucion: 2...
Tiempo de ejecucion: 3...
Tiempo de ejecucion: 4...
Nombre
Tiempo 0
El prioridad fue 0
Tiempo de ejecucion: 0...
root@ubuntu-18:/home/freddy#
```

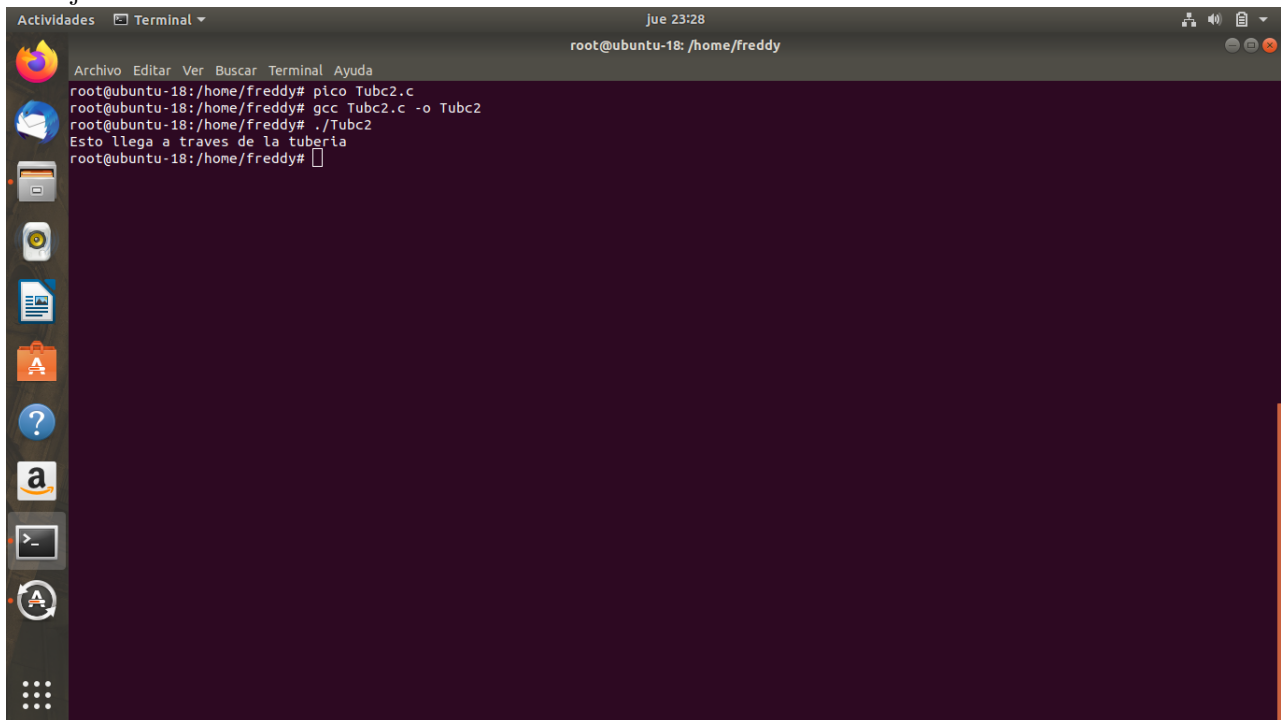
Creando un sincronizador de procesos mediante nodos, lo que realiza es mandar a llamar a los procesos para indicarles su prioridad y el tiempo de cada uno.



```
1 #include <fcntl.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <string.h>
6
7 #define SIZE 512
8
9 int main ( char argc , char **argv )
10 {
11     pid_t pid;
12     int p[2], readbytes;
13     char buffer[SIZE];
14
15     pipe( p );
16
17     if ( (pid=fork()) == 0 )
18     { // hijo
19         close( p[1] ); /* cerramos el lado de escritura del pipe */
20
21         while( (readbytes=read( p[0], buffer , SIZE )) > 0)
22             write( 1, buffer , readbytes );
23
24         close( p[0] );
25     }
26     else
27     { // padre
28         close( p[0] ); /* cerramos el lado de lectura del pipe */
29
30         strcpy( buffer , "Esto llega a traves de la tuberia\n" );
31         write( p[1], buffer , strlen( buffer ) );
32
33         close( p[1] );
34     }
35     write( pid , NULL, 0 );
36     exit( 0 );
37 }
```

---

## Ejecución



```
root@ubuntu-18: /home/freddy# pico Tubc2.c
root@ubuntu-18: /home/freddy# gcc Tubc2.c -o Tubc2
root@ubuntu-18: /home/freddy# ./Tubc2
Esto llega a través de la tubería
root@ubuntu-18: /home/freddy#
```

Creando pipes que llegan mediante tuberías.