TP VIII – IPC – Memoria Compartida y Semáforos – Shared Memory & Semaphores

Objetivo: Desarrollar una serie de programas que permitan comprobar distintas funcionalidades vinculadas con la gestión de memoria compartida y semáforos de tipo System V Release 4 (SVR4) como una forma de comunicación entre procesos (IPC: Inter Process Communication) provistas por el kernel del SO. Introducir al desarrollo de programas servidores y clientes, productores y consumidores. Aplicar la solución propuesta en la teoría en cuanto al problema del Productor - Consumidor.

Introducción

La memoria compartida es una de las técnicas de IPC más utilizadas y eficientes puesto que los datos están disponibles para todos los procesos y no requieren que éstos realicen "copias privadas" de los datos a intercambiar. La implementación de distintos tipos de servidores y soluciones empresariales basadas en memoria compartida plantea el problema de la exclusión mutua, requiriendo de semáforos para el control de concurrencia entre procesos.

En 1965 se publica el tratado de Dijkstra[1] cuyo principio fundamental es: “dos o más procesos pueden cooperar por medio de simples señales, de forma tal, que un proceso pueda ser obligado a parar en un lugar determinado hasta que haya recibido una señal específica. Cualquier requisito complejo de coordinación puede ser satisfecho con la estructura de señales apropiada”. Los semáforos son las señales a las cuales se refería Dijstra; enviar una señal al semáforo `s` es `semSignal(s)` y parar hasta recibir la señal del semáforo `s` es `semWait(s)`.

El Analista Y pretende diseñar un servidor que actualice los totales de ventas de las sucursales en un area de memoria compartida, de esta forma, otros procesos clientes podrán consultarla. Se simularán los totales de ventas de las sucursales y éstas se almacenarán como un arreglo de `10` posiciones de tipo `double`. Estos totales son información resumida en base a los datos de facturación de las sucursales.
Implementar un area de memoria compartida, implica obtener una clave IPC (tal como se comentó en el TP anterior), luego crear el area de memoria compartida usando dicha clave e indicando su tamaño en bytes y los permisos sobre la misma (tal como si se tratase de una cola de mensajes, archivo, proceso, etc.). Una vez creada la memoria compartida, ésta es accesible a través de un puntero local, es decir, nuestro programa obtendrá una referencia al area de memoria compartida creada (nos hemos conectado con el area de memoria compartida). A través de este puntero accedemos a la memoria compartida, nos desplazamos, leemos y cambiamos dicha memoria compartida como si fuese un area de memoria local de nuestro proceso. Estos cambios están disponibles para todos los procesos que acceden a dicha memoria. Por último, debemos desconectar la memoria compartida. En el caso del servidor, cuando éste termine su tarea, se supone no habrá clientes por atender, por lo tanto, borrará/destruirá la memoria compartida creada originalmente.
La descripción anterior coincide con los siguientes diseños:

[1] Dijstra, E., “Cooperating Sequential Processes”, Technological University, Eindhoven, The Netherlands, 1965.

Programa servidor

```
main()
      crear clave IPC según path y id proyecto
      crear Memoria usando clave,tamaño,permisos
      si pude crear Memoria Ok entonces
            conectar Memoria a través de pventas (puntero double *)
            si pude conectar Memoria Ok entonces
                  inicializar Memoria (pongo vector en cero)
                  generar totales de ventas de sucursales usando pventas
                  mostrar totales de ventas
                  mientras no salir
                        bloquear proceso hasta recibir señal (pause())
                  fin mientras
                  desconectar Memoria
            sino
                  error conectando Memoria
            fin si
            borrar Memoria
      sino
            error creando Memoria
      fin si
fin main()
```

Programa cliente

```
main()
      crear clave de IPC según path y id proyecto
      obtener Memoria usando clave,permisos
      si pude obtener Memoria Ok entonces
            conectar Memoria a través de pcliente (puntero double *)
            si pude conectar Memoria Ok entonces
                  mostrar totales de ventas usando pcliente
                  desconectar Memoria
            sino
                  error conectando Memoria
            fin si
      sino
            error creando Memoria
      fin si
fin main()
```

1. El Analista de Sistemas Y está muy entusiasmado con memoria compartida y semáforos; está convencido de que este no será el último desarrollo en esta línea, por lo tanto, se propone ampliar su librería para el manejo de IPC: `myipc.c` cuyo archivo de cabecera es `myipc.h`. Aquí nuevamente, aplicamos el criterio indicado en el punto 4 del TP VII Cola de Mensajes (puede copiar la librería o recompilarla). Este archivo de cabecera y librería será compartido tanto por cliente como por servidor, por lo tanto, aquí podemos acordar el path y id de proyecto, en este caso, usaremos otra clave para Memoria Compartida, así que dentro de `myipc.h` deberemos incluir:

```
#define IPC_MKEY 'h'
```

También se deberán incluir dentro de `myipc.h` los archivos de cabecera para utilizar semáforos y memoria compartida:

```
...
#include <sys/shm.h>
#include <sys/sem.h>
...
```

Los prototipos de las funciones a implementar dentro de `myipc.c` serán:

```
int borrar_memoria(int shmid);
void *conectar_memoria(int shmid,int modo);
int crear_memoria(key_t key,int size,int modo);
int desconectar_memoria(void *buffer);
/* Obtiene memoria compartida previamente creada */
int obtener_memoria(key_t key,int modo);
```

Tanto cliente como servidor ejecutarán:

```
key_t clave = obtener_clave(IPC_PATH, IPC_MKEY);
```

para generar una clave IPC con la cual identificar a la memoria compartida.
El servidor creará la memoria compartida en modo lectura-escritura:

```
int modo = SHM_R|SHM_W;  // lectura y escritura
int tamanio = sizeof(double)*10;  // son 10 totales de ventas
idmem = crear_memoria(clave,tamanio,modo);
```

El cliente se conectará a una memoria previamente creada en modo lectura:

```
int modo = SHM_R;
idmem = obtener_memoria(clave,modo)
```

Estas funciones devolverán -1 cuando exista alguna condición de error.
Tanto cliente como servidor deberán conectar a la memoria compartida:

```
pventas = (double *) conectar_memoria(idmem,modo);
```

El servidor inicializará el total de ventas de las sucursales y generará los totales:

```
memset(pventas,0,tamanio);
init(pventas);
```

También es posible agregar otras funciones compartidas y reutilizables a la librería `myipc.c`, por ejemplo:

```
void mostrar_ventas(double *ventas);
```

esta función permite mostrar el total de ventas de todas las sucursales por la consola, a partir de un puntero al primer total de ventas.

De forma muy similar y consistente con la cola de mensajes, para crear/acceder a un area de memoria compartida debemos utilizar la función `shmget()`, cuando se indica tamaño, implica una creación de la memoria y cuando el tamaño es `0`, implica una acceso a la misma. Para evitar confusión y errores de los programadores, el Analista Y ha decidido usar dos entradas distintas en la API de `myipc` (`crear_memoria()` y `obtener_memoria()`) para las dos situaciones:

```
int crear_memoria(key_t key,int size,int modo) {
   return shmget(key, size, IPC_CREAT|modo);
}
int obtener_memoria(key_t key,int modo) {
   return crear_memoria(key,0,modo);
}
```

Se consideraría incorrecto implementar `obtener_memoria()` en función de `shmget()`[2], esto atentaría contra el reuso y las ventajas que éste nos otorga: modificamos `crear_memoria()` y automáticamente también correjimos `obtener_memoria()`.
Para conectar a un area de memoria compartida usamos `shmat()` (share memory *at*tach) y `shmdt()` para desconectarnos (share memory *de*tach):

```
void *conectar_memoria(int shmid,int modo) {
   return shmat(shmid,0,modo);
}
int desconectar_memoria(void *buffer) {
   return shmdt(buffer);
}
```

De igual forma que con la cola de mensajes, una vez que la hemos creado, podemos cambiar algunas características de la misma, utilizando la función `shmctl()`, ésta también nos servirá para destruir el area de memoria compartida:

```
int borrar_memoria(int shmid) {
   return shmctl(shmid, IPC_RMID, 0);
}
```

La API requerida para manipular un area de memoria compartida es aún más simple que el de una cola de mensajes.
Puede completar el resto de la API, hacer un pequeño programa de prueba de la librería y compilar el código.

2. Utilice el programa de prueba de la librería del punto anterior y cópielo como `tp81.c`. Implemente en este programa el servidor de memoria compartida, tal como se indicó en la introducción.
Esta primera versión simplemente creará el area de memoria compartida, la inicializará, generará los totales de ventas, entrará en el loop principal y utilizará la función `pause()` para bloquear el proceso hasta que se llegue una señal. Implemente la señal `SIGUSR2` en combinación con la variable `salir` (`volatile`) para evitar que el servidor quede bloqueado indefinidamente.
Compile y ejecute `tp81.c`. Podrá obtener una salida como esta:

[2] Siendo posible el no hacerlo.

consola server:

```
grchere@debian:~/gccwork/src/sem$ ipcs
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere   600        196608     2          dest
0x00000000 65537      grchere   600        196608     2          dest
0x00000000 98306      grchere   600        393216     2          dest
0x00000000 131075     grchere   600        393216     2          dest
0x00000000 163844     grchere   600        12288      2          dest
0x00000000 262149     grchere   600        393216     2          dest
0x00000000 294918     grchere   600        196608     2          dest
0x00000000 327687     grchere   600        393216     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

grchere@debian:~/gccwork/src/sem$

grchere@debian:~/gccwork/src/sem$ ./tp81
main():inicio proceso productor!
main():para salir envie se?al SIGUSR2 a proceso 2894
main():solicito clave ipc
main():creo memoria compartida de 80 bytes
main():atachar memoria local a memoria compartida 393224
main():Inicializo memoria compartida a cero
main():genero total de ventas de sucursales en memoria compartida
mostrar_ventas(): sucursal[0]=105.933955
mostrar_ventas(): sucursal[1]=118.313756
mostrar_ventas(): sucursal[2]=172.562817
mostrar_ventas(): sucursal[3]=207.236120
mostrar_ventas(): sucursal[4]=19.042746
mostrar_ventas(): sucursal[5]=134.783351
mostrar_ventas(): sucursal[6]=206.104176
mostrar_ventas(): sucursal[7]=46.965428
mostrar_ventas(): sucursal[8]=117.049172
mostrar_ventas(): sucursal[9]=96.927325
main():Bloqueo proceso hasta recibir se?al de salida
```

consola cliente:

```
grchere@debian:~$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere   600        196608     2          dest
0x00000000 65537      grchere   600        196608     2          dest
0x00000000 98306      grchere   600        393216     2          dest
0x00000000 131075     grchere   600        393216     2          dest
0x00000000 163844     grchere   600        12288      2          dest
0x00000000 262149     grchere   600        393216     2          dest
0x00000000 294918     grchere   600        196608     2          dest
0x00000000 327687     grchere   600        393216     2          dest
0x68022a15 393224     grchere   600        80         1

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
```

```
key        msqid      owner      perms      used-bytes   messages

grchere@debian:~$
grchere@debian:~$ kill -SIGUSR2 2894
```

luego de enviada la señal SIGUSR2:

```
consola server:
main():recibi se?al de salida
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0
grchere@debian:~/gccwork/src/sem$
```

```
consola cliente:
grchere@debian:~$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere   600        196608     2          dest
0x00000000 65537      grchere   600        196608     2          dest
0x00000000 98306      grchere   600        393216     2          dest
0x00000000 131075     grchere   600        393216     2          dest
0x00000000 163844     grchere   600        12288      2          dest
0x00000000 262149     grchere   600        393216     2          dest
0x00000000 294918     grchere   600        196608     2          dest
0x00000000 327687     grchere   600        393216     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

grchere@debian:~$
```

3. Copiar tp81.c en tp82.c. . Este programa será el cliente de tp81.c. Hágalo según lo indicado en la introducción. Tp82.c es un cliente que accede a la memoria compartida y muestra los valores allí contenidos, no tiene datos globales, no especifica tamaño de memoria compartida, el modo de acceso es SHM_R, no olvide utilizar obtener_memoria() en vez de crear_memoria(). Conecta memoria con puntero local, usa puntero local para mostrar ventas (mostrar_ventas()), desconecta memoria, no implementa señales.
Podrá obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/sem$ ./tp81
main():inicio proceso productor!
main():para salir envie se?al SIGUSR2 a proceso 2986
main():solicito clave ipc
main():creo memoria compartida de 80 bytes
main():atachar memoria local a memoria compartida 458760
main():Inicializo memoria compartida a cero
main():genero total de ventas de sucursales en memoria compartida
mostrar_ventas(): sucursal[0]=105.933955
mostrar_ventas(): sucursal[1]=118.313756
mostrar_ventas(): sucursal[2]=172.562817
```

```
mostrar_ventas(): sucursal[3]=207.236120
mostrar_ventas(): sucursal[4]=19.042746
mostrar_ventas(): sucursal[5]=134.783351
mostrar_ventas(): sucursal[6]=206.104176
mostrar_ventas(): sucursal[7]=46.965428
mostrar_ventas(): sucursal[8]=117.049172
mostrar_ventas(): sucursal[9]=96.927325
main():Bloqueo proceso hasta recibir se?al de salida
main():recibi se?al de salida
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0
grchere@debian:~/gccwork/src/sem$
```

consola cliente:

```
grchere@debian:~/gccwork/src/sem$ ./tp82
main():inicio proceso consumidor!
main():para salir envie se?al SIGUSR2 a proceso 2987
main():solicito clave ipc
main():obtengo memoria compartida
main():atachar memoria local a memoria compartida 458760
main():leo memoria compartida
mostrar_ventas(): sucursal[0]=105.933955
mostrar_ventas(): sucursal[1]=118.313756
mostrar_ventas(): sucursal[2]=172.562817
mostrar_ventas(): sucursal[3]=207.236120
mostrar_ventas(): sucursal[4]=19.042746
mostrar_ventas(): sucursal[5]=134.783351
mostrar_ventas(): sucursal[6]=206.104176
mostrar_ventas(): sucursal[7]=46.965428
mostrar_ventas(): sucursal[8]=117.049172
mostrar_ventas(): sucursal[9]=96.927325
main():Desconecto memoria compartida
main():fin proceso consumidor! retorno=0
grchere@debian:~/gccwork/src/sem$
grchere@debian:~/gccwork/src/sem$ kill -SIGUSR2 2986
```

4. Copiar `tp81.c` en `tp83.c`. El servidor `tp81.c` mostraba siempre los mismos totales de ventas. El Analista Y sabe que estos datos son muy dinámicos. Implemente en `tp83.c` una nueva versión del servidor, ahora el servidor permite cambiar los datos de los totales de ventas de las sucursales cada vez que recibe la señal `SIGUSR1`.

Esta nueva versión de servidor (más real que la anterior) plantea un problema: `tp82` (el cliente) podría ver -en una misma consulta- un "mix" de valores actualizados y no actualizados de totales de ventas de sucursales. Esto no compromete al sistema, ni a los procesos, puesto que el acceso de todos los clientes es de lectura. Si no se desea este efecto, es decir, se desea que, ante una consulta de un cliente, ésta muestre todos los datos actualizados de todas las sucursales. De ser así, se requiere de un mecanismo de exclusión mutua: un cliente no puede consultar datos de sucursales cuando estos datos están siendo actualizados por el servidor. Sólo por ahora, no vamos a preocuparnos de esto.

Sugerencia: utilizar un puntero global que apunte al vector de totales de ventas “atachado” a memoria compartida, cuando se reciba la señal `SIGUSR1`, recalcular los valores sumando un 10% al valor actual (suponiendo que `p` es un dato local y `pventas` es el puntero global):

```
...
   double *p = pventas;
   int i;for(i=0;i<10;i++,p++) *p *= 1.1;
...
```

Podrá obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/sem$ ./tp83
main():inicio proceso productor!
main():para recalcular envie se?al SIGUSR1 a proceso 2904
main():para salir envie se?al SIGUSR2 a proceso 2904
main():solicito clave ipc
main():creo memoria compartida de 80 bytes
main():atachar memoria local a memoria compartida 360455
main():Inicializo memoria compartida a cero
main():genero total de ventas de sucursales en memoria compartida
mostrar_ventas(): sucursal[0]=116.592117
mostrar_ventas(): sucursal[1]=10.888748
mostrar_ventas(): sucursal[2]=151.835415
mostrar_ventas(): sucursal[3]=59.417490
mostrar_ventas(): sucursal[4]=110.724445
mostrar_ventas(): sucursal[5]=188.109783
mostrar_ventas(): sucursal[6]=193.502843
mostrar_ventas(): sucursal[7]=87.824307
mostrar_ventas(): sucursal[8]=210.795342
mostrar_ventas(): sucursal[9]=57.125540
main():Bloqueo proceso hasta recibir se?al de salida

(luego de recibir SIGUSR1:)

mostrar_ventas(): sucursal[0]=128.251329
mostrar_ventas(): sucursal[1]=11.977622
mostrar_ventas(): sucursal[2]=167.018956
mostrar_ventas(): sucursal[3]=65.359239
mostrar_ventas(): sucursal[4]=121.796890
mostrar_ventas(): sucursal[5]=206.920762
mostrar_ventas(): sucursal[6]=212.853127
mostrar_ventas(): sucursal[7]=96.606737
mostrar_ventas(): sucursal[8]=231.874876
mostrar_ventas(): sucursal[9]=62.838094
...
main():recibi se?al de salida
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0

consola cliente:
grchere@debian:~/gccwork/src/sem$ kill -SIGUSR1 2904
grchere@debian:~/gccwork/src/sem$
```

5. Hasta aquí hemos realizado lectura y escritura de memoria compartida, con la posible necesidad de exclusión mutua.
Copiar `tp83.c` como `tp84p.c`, para implementar una nueva versión de este servidor.
La Empresa X tiene un sistema que va registrando las ventas que ocurren en las distintas sucursales.

Modifique `tp84p.c` para que éste guarde (junto con el vector de totales de ventas por sucursal) en la misma area de memoria compartida (a medida que se van reportando las ventas) los siguientes datos que representan una venta en una sucursal:

sucursal (int, número de sucursal)
fecha (almacenada como un conjunto de caracteres con el formato: dd/mm/yyyy)
factura (int, número de factura)
monto (double, importe total de la factura)

estos datos pueden representarse a través de la estructura `sucvta`:

```
struct sucvta {
   int sucursal;
   char fecha[11]; // almacena tambien el \0 final de la cadena de caracteres
   int factura;
   double monto;
};
```

luego podemos, por ejemplo, declarar un puntero a una estructura de dicho tipo:

```
...
   struct sucvta *psucvta;
...
```

a medida que llegan las ventas, este proceso las ubica en un buffer de `10` elementos, es decir, que este programa tiene capacidad para "producir"[3] y almacenar hasta un máximo de `10` ventas[4] hechas en algunas de las sucursales de la Empresa X, paralelamente se continúa manteniendo otro buffer de `10` elementos con los totales de ventas de las sucursales (al igual que en `tp81`). Ambos se encuentran en la misma area de memoria compartida de la siguiente forma:

```
<arreglo de 10 elementos para totales de sucursal><arreglo de 10 elementos para
las ventas de las sucursales>
```

ambos arreglos se inicializan a `0` cuando comienza el proceso. Por lo tanto, el nuevo tamaño de memoria compartida sería:

```
...
   int tamanio = (sizeof(double)*10)+(sizeof(struct sucvta)*10);
...
```

si `pventas` continúa siendo un puntero al primer arreglo de `10` elementos:

```
...
   pventas = (double *) conectar_memoria(idmem,modo);
...
```

Entonces, `psucvta` (puntero al segundo arreglo en memoria compartida) sería:

[3] Efectivamente, según lo visto en la teoría, tp84p se trata de un proceso productor. De allí el cambio de nomenclatura en su nombre.
[4] Para ser más precisos, tp84p es un proceso productor con buffer acotado.

```
...
   psucvta = (struct sucvta *) (pventas + sizeof(double)*10);
...
```

debemos sumarle el tamaño del primer arreglo[5] para que este puntero quede apuntando al comienzo del segundo puntero. Es nuestra responsabilidad como programadores hacer un manejo correcto de estos punteros, sin superar ninguno de sus límites[6].
Se puede implementar la funcion `void generar_venta(struct sucvta *)` para simular una venta de una sucursal de la siguiente forma:

```
void generar_venta(struct sucvta *sv) {
   sv->sucursal = (int) ((long) random() % 10L);
   int dia = (int) ((long) random() % 31L);
   int mes = (int) ((long) random() % 12L)+1;
   int anio = 2009;
   snprintf(sv->fecha,11,"%2d/%2d/%4d",dia,mes,anio);
   sv->factura = (int) ((long) random())/1000000L;
   sv->monto = ((double) random())/10000000.0;
}
```

A medida que el servidor recibe ventas de sucursales (genera ventas), se deberá actualizar el correspondiente total de ventas de la sucursal:

```
...
            p = psucvta;
            for(i=0;i<10;i++,p++) {
               generar_venta(p);
               *(pventas+p->sucursal) += p->monto; // acumulo en totales
            }
...
```

obviamente, `p` es de tipo `struct sucvta *` y `pventas` continua siendo un puntero al primer elemento del arreglo de totales de ventas, obsérvese que `pventas` + <número de sucursal> es el desplazamiento necesario en memoria para actualizar en el lugar correcto el total de ventas de la sucursal.
Generar ventas cada 10 segundos aproximadamente, para ello puede hacer una llama a la función sleep (`sleep(10);`) dentro del loop principal del servidor, luego de generar las ventas y actualizar los totales de ventas.

Podrá obtener una salida como esta:

```
consola server:
grchere@debian:~/gccwork/src/sem$ ./tp84p
main():inicio proceso productor!
main():para salir envie se?al SIGUSR2 a proceso 3037
main():solicito clave ipc
main():creo memoria compartida de 360 bytes
main():atachar memoria local a memoria compartida 458759
main():Inicializo memoria compartida a cero
```

[5] A esto se lo denomina “desplazamiento” (offset).
[6] Suele ser la mayor fuente de errores en la programación C/C++, al mismo tiempo, es una de las características que hacen a la performance, la elegancia y lo compacto del código C.

```
main():genero total de ventas de sucursales en memoria compartida
main():Bloqueo proceso hasta recibir se?al de salida
mostrar_ventas(): sucursal[0]=352.309974
mostrar_ventas(): sucursal[1]=555.469024
mostrar_ventas(): sucursal[2]=129.552290
mostrar_ventas(): sucursal[3]=659.881827
mostrar_ventas(): sucursal[4]=1056.077428
mostrar_ventas(): sucursal[5]=181.898165
mostrar_ventas(): sucursal[6]=1130.669420
mostrar_ventas(): sucursal[7]=522.252935
mostrar_ventas(): sucursal[8]=741.985095
mostrar_ventas(): sucursal[9]=590.463744
mostrar_sucventas(): suc=[4] fecha=[27/ 4/2009] fac=[1595] monto=[197.529392]
mostrar_sucventas(): suc=[0] fecha=[27/ 8/2009] fac=[778] monto=[145.057775]
mostrar_sucventas(): suc=[8] fecha=[29/ 4/2009] fac=[607] monto=[38.361159]
mostrar_sucventas(): suc=[1] fecha=[ 4/ 5/2009] fac=[1546] monto=[173.807989]
mostrar_sucventas(): suc=[6] fecha=[ 2/ 6/2009] fac=[1294] monto=[160.439974]
mostrar_sucventas(): suc=[9] fecha=[10/ 6/2009] fac=[205] monto=[89.141984]
mostrar_sucventas(): suc=[6] fecha=[16/ 7/2009] fac=[806] monto=[42.004613]
mostrar_sucventas(): suc=[7] fecha=[27/ 2/2009] fac=[2034] monto=[61.111264]
mostrar_sucventas(): suc=[3] fecha=[27/ 3/2009] fac=[1620] monto=[24.583503]
mostrar_sucventas(): suc=[6] fecha=[16/ 7/2009] fac=[2055] monto=[206.386668]
mostrar_ventas(): sucursal[0]=639.126360
mostrar_ventas(): sucursal[1]=571.337362
mostrar_ventas(): sucursal[2]=202.827572
mostrar_ventas(): sucursal[3]=659.881827
mostrar_ventas(): sucursal[4]=1056.077428
mostrar_ventas(): sucursal[5]=213.922284
mostrar_ventas(): sucursal[6]=1310.140690
mostrar_ventas(): sucursal[7]=580.505195
mostrar_ventas(): sucursal[8]=848.977764
mostrar_ventas(): sucursal[9]=758.659658
mostrar_sucventas(): suc=[0] fecha=[15/ 9/2009] fac=[691] monto=[188.332799]
mostrar_sucventas(): suc=[9] fecha=[ 9/11/2009] fac=[1604] monto=[168.195914]
mostrar_sucventas(): suc=[7] fecha=[28/ 4/2009] fac=[2007] monto=[58.252260]
mostrar_sucventas(): suc=[6] fecha=[ 8/10/2009] fac=[367] monto=[173.445431]
mostrar_sucventas(): suc=[2] fecha=[ 8/ 6/2009] fac=[902] monto=[73.275281]
mostrar_sucventas(): suc=[8] fecha=[13/ 5/2009] fac=[1902] monto=[106.992669]
mostrar_sucventas(): suc=[5] fecha=[12/10/2009] fac=[1205] monto=[32.024119]
mostrar_sucventas(): suc=[1] fecha=[13/ 4/2009] fac=[351] monto=[15.868338]
mostrar_sucventas(): suc=[6] fecha=[27/ 8/2009] fac=[1837] monto=[6.025839]
mostrar_sucventas(): suc=[0] fecha=[22/ 9/2009] fac=[931] monto=[98.483587]
...
main():recibi se?al de salida
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0
grchere@debian:~/gccwork/src/sem$
```

6. Copiar `tp82.c` en `tp84c.c`. Modificar `tp84c.c` para implementar una nueva versión del programa cliente: la Empresa X tiene uno o más procesos cliente que acceden al area de memoria compartida (en donde el programa servidor genera y actualiza los datos) y retiran desde allí las ventas de las sucursales (cada vez que se retira una venta, esa porción de memoria compartida se inicializa a cero)[7]; los datos de ventas que se extraen de la memoria se guardan en un archivo secuencial para su posterior procesamiento. El nombre del archivo se forma con el id del proceso y la extensión `.ventas`, con un formato de texto, tal como:

[7] Según lo visto en la teoría, tp84c es un proceso consumidor.

```
...
        char archivo[30];
        snprintf(archivo,30,"%d.ventas",(int) getpid());
        FILE *fpo=fopen(archivo,"a");
...
        fclose(fpo);
```

la estructura interna del archivo es la siguiente:

```
      número de sucursal (2 posiciones)
      fecha (8 posiciones)
      factura (10 posiciones numéricas)
      monto (12 posiciones totales con punto y dos decimales)
```

cada registro se separa con `\n` (new line, Intro), de forma tal, que es posible ver/editar su contenido con cualquier archivo de texto.
Puede lograr esto valiéndose de las funciones `fopen()`, `fclose()` (tal como se indicó anteriormente) y puede usar `fprintf()` para grabar los registros con el formato indicado previamente, de la siguiente forma (suponiendo que `sv` es un puntero de tipo `struct sucvta *` y apunta a la posición iésima del vector de ventas de sucursales):

```
...
   fprintf(salida,"%2d%s%10d%12.2lf\n",
                          sv->sucursal,sv->fecha,sv->factura,sv->monto);
...
```

`salida` debe ser un stream, algo de tipo `FILE *`. Todo programa C tiene acceso a tres streams (que ni siquiera necesita abrir): `stdio` (generalmente el teclado), `stout` (generalmente el monitor), `stderr` (generalmente el monitor). Por lo tanto se podría diseñar una función tal como `void mostrar_sucventas(FILE *,struct sucvta *)`, que nos sirva tanto para mostrar por pantalla como para enviar a un archivo:

```
...
   FILE *fout = fopen("miarchivo.txt","a");
   struct sucvta *sv = ...;
...

   // para mostrar por pantalla
   mostrar_sucventas(stdout,sv);
   // para grabar en archivo
   mostrar_sucventas(fout,sv)
...
   fclose(fout);
...
```

Implemente el programa cliente consumidor `tp84c.c`, podrá obtener una salida como esta:

```
consola cliente:
grchere@debian:~/gccwork/src/sem$ ./tp84c
main():inicio proceso consumidor!
main():solicito clave ipc
main():obtengo memoria compartida
```

```
main():atachar memoria local a memoria compartida 589831
main():leo memoria compartida
mostrar_ventas(): sucursal[0]=397.650330
mostrar_ventas(): sucursal[1]=326.846774
mostrar_ventas(): sucursal[2]=5.049239
mostrar_ventas(): sucursal[3]=215.701141
mostrar_ventas(): sucursal[4]=282.191748
mostrar_ventas(): sucursal[5]=299.724144
mostrar_ventas(): sucursal[6]=480.924430
mostrar_ventas(): sucursal[7]=413.780045
mostrar_ventas(): sucursal[8]=191.408803
mostrar_ventas(): sucursal[9]=156.867319
mostrar_sucventas(): suc=[5] fecha=[22/ 4/2009] fac=[1175] monto=[142.859295]
mostrar_sucventas(): suc=[6] fecha=[11/10/2009] fac=[350] monto=[26.475312]
mostrar_sucventas(): suc=[0] fecha=[18/ 5/2009] fac=[1268] monto=[30.115256]
mostrar_sucventas(): suc=[9] fecha=[25/ 7/2009] fac=[1681] monto=[1.871950]
mostrar_sucventas(): suc=[7] fecha=[17/ 4/2009] fac=[1927] monto=[191.437293]
mostrar_sucventas(): suc=[6] fecha=[ 8/ 7/2009] fac=[573] monto=[208.439285]
mostrar_sucventas(): suc=[1] fecha=[ 0/ 9/2009] fac=[1449] monto=[10.224205]
mostrar_sucventas(): suc=[7] fecha=[16/11/2009] fac=[799] monto=[208.244878]
mostrar_sucventas(): suc=[1] fecha=[26/ 6/2009] fac=[73] monto=[207.803250]
mostrar_sucventas(): suc=[4] fecha=[11/ 5/2009] fac=[1770] monto=[61.441396]
main():Desconecto memoria compartida
main():fin proceso consumidor! retorno=0
grchere@debian:~/gccwork/src/sem$
```

7. Según lo visto en la teoría, esto se trata de una situación productor - consumidor con un buffer acotado. El primer proceso es el productor (`tp84p.c`), el segundo proceso (`tp84c.c`) es el consumidor. Ahora se requiere de la implementación de exclusión mutua, teniendo en cuenta de no producir cuando ya no hay lugar dónde guardar la producción y de no consumir cuando ya no hay nada que consumir.
Implementar ambos procesos utilizando memoria compartida y 3 semáforos según lo recomendado por William Stallings en su libro "Sistemas Operativos" 5ta. edición (español) en la página 227, Fig. 5.13, "Una solución al problema productor/consumidor con buffer acotado usando semáforos".
Probar la ejecución concurrente del productor y uno o más consumidores.
7.1) Los semáforos a utilizar serán de tipo System V Release 4 (SVR4)[8], nuevamente, se debe ampliar la librería `myipc.c` para dar soporte a semáforos, se propone la siguiente API básica:

```
int crear_semaforo(key_t key,int modo);
int borrar_semaforo(int semid);
int borrar_semaforos(int semid,int nsem);
int semSignalTo(int semid,int nro); // incrementar/decrementar en n un semaforo
int semSignal(int semid); // incrementar un semaforo
int semWait(int semid);   // decrementar un semaforo
int obtener_semaforo(key_t key,int modo);
```

debido al hecho de que, en SVR4 no se crea un semáforo sino un conjunto de semáforos, entonces se podría pensar en una ampliación de esta API básica para agregar otras funciones que trabajen sobre grupos de semáforos y algunas otras operaciones adicionales[9]:

[8] Vale la aclaración puesto que también existen los semáforos de tipo POSIX (prototipos en semaphore.h). Existen varias diferencias entre éstos y los semáforos SVR4 que aquí usaremos, en cuanto a su implementación, su performance, su portabilidad.
[9] Dejamos la implementación de estas funciones a cargo de aquel alumno que quiera profundizar aún más sus conocimientos de la API de semáforos SVR4. Esta tarea esta fuera del alcance de este trabajo práctico.

```
int crear_semaforos(key_t key,int nsem,int modo);
int semSignalWaitTo(int semid,int nro);
int semSignalTo(int semid,int nro);
int semSignalWaitsTo(int semid,struct sembuf sb[],int nsem);
int semSignalsTo(int semid,struct sembuf sb[],int nsem) ;
int semSignalWaitsToNum(int semid,int nro,int nsem);
int semSignalsToNum(int semid,int nro,int nsem);
void obtener_claves(char *path[],char car[],key_t arrkey[],int nkeys);
```

Para especificar el tipo de uso que se pretende hacer del semáforo se pueden usar las macros SEM_R (para lectura) y SEM_A (para escritura, estas macros no se han encontrado en Linux (por lo menos, en la distribución Debian 5.01), en tal caso, podremos definirlas como:

```
// definicion de SEM_R & SEM_A
#ifndef SEM_R
   #define SEM_R 0400
#endif
#ifndef SEM_A
   #define SEM_A 0200
#endif
```

Si se va a crear un conjunto de semáforos (de un 1 semáforo cada uno) por cada semáforo propuesto por Stallings, se necesitará usar una clave única a combinar con el path para obtener una clave IPC única para cada semáforo (IPC_SKEYs, IPC_SKEYn, IPC_SKEYe). Las macros SEM_s, SEM_n, SEM_e, indican la posición dentro del arreglo usado para facilitar la creación de semáforos como se verá más adelante:

```
#define IPC_SKEYs 's'    // letra para semaforo s
#define IPC_SKEYn 'n'    // letra para semaforo n
#define IPC_SKEYe 'e'    // letra para semaforo e
#define SEM_s     0    // id (dentro del vector de semaforos) para el semaforo s
#define SEM_n     1    // id (dentro del vector de semaforos) para el semaforo n
#define SEM_e     2    // id (dentro del vector de semaforos) para el semaforo e
```

Nuevamente, debemos obtener una clave IPC y con ella intentar crear un conjunto de semáforos que se identificarán a través de un mismo id, para ello usaremos semget(), siguiendo la misma lógica de las restantes funciones de IPC:

```
// creamos semaforos "de a uno", a pesar de que la api de svr4
// permite crear n semaforos en una misma llamada
int crear_semaforo(key_t key,int modo) {
   return crear_semaforos(key,1,modo);
}
int crear_semaforos(key_t key,int nsem,int modo) {
   return semget(key, nsem, 0666|IPC_CREAT);
}
```

el o los procesos consumidores deberán obtener un semáforo previamente creado:

```
// obtenemos semaforos "de a uno", a pesar de que la api de svr4
// permite crear n semaforos en una misma llamada
int obtener_semaforo(key_t key,int modo) {
   return semget(key, 0, 0666);
```

```
}
```

es decir, que esta API crea un conjunto de `1` solo semáforo por vez y trabaja siempre sobre el semáforo número `0`.
Para borrar un semáforo, análogamente utilizamos la función `semctl()`:

```
int borrar_semaforo(int semid) {
   return borrar_semaforos(semid,0);
}
int borrar_semaforos(int semid,int nsem) {
   return semctl(semid, nsem, IPC_RMID, 0);
}
```

Para incrementar/decrementar un semáforo:

```
// incrementar en 1 a un semaforo
int semSignal(int semid) {
   return semSignalTo(semid,1);
}

// decrementar en 1 a un semaforo
int semWait(int semid) {
   return semSignalTo(semid,-1);
}

// incrementar/decrementar en n a un semaforo
int semSignalTo(int semid,int nro) {
   struct sembuf sb[1];
   sb[0].sem_num = 0;
   sb[0].sem_op = nro;
   sb[0].sem_flg = SEM_UNDO | 0; //vuelve atras el cambio si se cae este proceso
   return semop(semid,sb,1);
}
```

en este caso hay alguna diferencia con lo visto anteriormente, puesto que interviene la función `semop()` (realizar operaciones sobre semáforos). Cabe aclarar que cada semáforo del conjunto se representa por una estructura de tipo `sembuf`. Se podrá utilizar un arreglo de tipo `sembuf` para indicar `n` semáforos (o al menos uno). El elemento `sem_num` coincide con el número de semáforo dentro del conjunto (`0, 1, 2 ...`), `sem_op` el número a incrementar/decrementar (si es negativo) del semáforo y `sem_flg` indica el modo y otros parámetros, tales como `SEM_UNDO` que permite indicar que, en caso de que este proceso termine abruptamente, las operaciones realizadas por el mismo se restablecerán.

Modifique `myipc.h` y `myipc.c` en este sentido. Recompile.

7.2) Identificar sección critica de código en `tp84p.c` y `tp84c.c`. Copiar `tp84p.c` en `tp85p.c`. Modificar `tp85p.c` para crear una nueva versión del servidor, incorporando semáforos según Stallings y acorde con librería `myipc`:

El modo de uso de estos semáforos será de lectura y escritura:

```
...
   int sem_modo = SEM_R | SEM_A;
```

```
...
```

para las distintas claves asociadas con estos semáforos (según Stallings, los semáforos s,n,e):

```
...
   int sem_ids[3];     // id de c/u de los semaforos a crear: s,n,e
   key_t sem_keys[3];  // claves IPC para luego generar los ids de cada semaforo
   char *sem_paths[] = {IPC_PATH,IPC_PATH,IPC_PATH};
   char sem_chars[] = {IPC_SKEYs ,IPC_SKEYn ,IPC_SKEYe };
...

         printf("main():solicito claves ipc para semaforos\n");
         obtener_claves(sem_paths,sem_chars,sem_keys,3);
...
```

Una vez obtenidas las claves, se pueden crear e inicializar los semáforos y por último borrar:

```
...
   char *sem_error1[] = {"main():Error creando semaforo s",
                         "main():Error creando semaforo n",
                         "main():Error creando semaforo e",};
   char *sem_error2[] = {"main():Error inicializando semaforo s",
                         "main():Error inicializando semaforo n",
                         "main():Error inicializando semaforo e",};
   char *sem_error3[] = {"main():Error borrando semaforo s",
                         "main():Error borrando semaforo n",
                         "main():Error borrando semaforo e",};
   // valores iniciales de los semaforos s,n,e  segun Stallings Pag.227
   int sem_init[] = {1,0,1};
...

         printf("main():creo semaforos\n");
         for(i=0;i<3;i++)
            if ( (sem_ids[i] = crear_semaforo(sem_keys[i],sem_modo)) == -1)
               perror(sem_error1[i]);

         printf("main():inicializo semaforos\n");
         for(i=0;i<3;i++)
            if ( semSignalTo(sem_ids[i],sem_init[i]) == -1)
               perror(sem_error2[i]);
...
         // elimino semaforos
         printf("main():borro semaforos\n");
         for(i=0;i<3;i++)
            if ( borrar_semaforo(sem_ids[i]) == -1)
               perror(sem_error3[i]);
...
```

Habiendo creado los semáforos de esta forma, se pueden incrementar/decrementar sus valores de la siguiente forma:

```
...
           semWait(sem_ids[SEM_e]);  // semWait(e)
...
           semSignal(sem_ids[SEM_n]);  // semSignal(n)
...
```

También se puede usar un único set de semáforos o bien tres semáforos independientes (como se optó en este caso).
Eliminar la llamada: `sleep(10);` dentro del loop principal del servidor para generar un mayor nivel de concurrencia con `tp85c`.

Podrá obtener una salida como esta:

consola server:

```
debian:/home/grchere/gccwork/src/sem# ./tp85p
main():inicio proceso productor!
main():para salir envie se?al SIGUSR2 a proceso 2708
main():solicito clave ipc
main():creo memoria compartida de 360 bytes
main():atachar memoria local a memoria compartida 262151
main():Inicializo memoria compartida a cero
main():genero total de ventas de sucursales en memoria compartida
main():solicito claves ipc para semaforos
main():creo semaforos
main():inicializo semaforos
main():Bloqueo proceso hasta recibir se?al de salida
mostrar_ventas(): sucursal[0]=136.105426
mostrar_ventas(): sucursal[1]=145.062103
mostrar_ventas(): sucursal[2]=298.346262
mostrar_ventas(): sucursal[3]=58.790672
mostrar_ventas(): sucursal[4]=13.892183
mostrar_ventas(): sucursal[5]=210.941994
mostrar_ventas(): sucursal[6]=184.822537
mostrar_ventas(): sucursal[7]=288.855166
mostrar_ventas(): sucursal[8]=80.767787
mostrar_ventas(): sucursal[9]=330.083290
 913/ 5/2009      1983       85.81
 213/ 5/2009       323      186.31
 829/10/2009       105       38.55
 713/ 3/2009       973      140.58
 915/ 7/2009       202      208.76
 3 1/ 2/2009      1432       58.66
 0 4/ 2/2009      2039       24.81
 629/11/2009       102      115.21
 128/ 2/2009       167       89.86
 223/ 6/2009      2123       10.84
...
main():borro semaforos
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0
debian:/home/grchere/gccwork/src/sem#
```

7.3) Idem anterior. Copiar `tp84c.c` en `tp85c.c`. Modificar tp85c.c para incorporar semáforos según Stallings y acorde con libreria `myipc`. Modificar `tp85c`.c para consumir (en este caso, significa mostrar los totales de ventas por sucursal, mostrar las ventas de sucursales, agregar las ventas de las sucursales en el archivo <`id proceso`>.`ventas`, poner en cero las ventas de sucursales) los totales generados por `tp85p.c` hasta que reciba la señal `SIGUSR2` para salir.
Este proceso no destruye ni crea semáforos, sino que obtiene el id de los semáforos creados por el servidor (usando los mismos arreglos que en el caso anterior):

```
...
        printf("main():solicito claves ipc para semaforos\n");
        obtener_claves(sem_paths,sem_chars,sem_keys,3);
        printf("main():obtengo semaforos\n");
        for(i=0;i<3;i++)
           if ( (sem_ids[i] = obtener_semaforo(sem_keys[i],sem_modo)) == -1)
              perror(sem_error1[i]);
...
```

El proceso de "consumir", asumiendo que:

- `fpo` es un puntero de tipo `FILE *` que apunta al stream abierto de modo append conectado con el archivo <`id proceso`>`.ventas`
- `pventas` es un puntero de tipo `double *` que apunta al primer elemento del arreglo de totales de ventas por sucursal
- `psucvta` es un puntero de tipo `struct sucvta *` que apunta al primer elemento del arreglo de ventas realizadas en sucursales podría ser el siguiente:

```
...
              mostrar_sucventas(fpo,psucvta);
              mostrar_ventas(pventas);
              mostrar_sucventas(stdout,psucvta);
              printf("main():pongo a cero memoria consumida\n");
              memset(psucvta,0,sizeof(struct sucvta)*10);
...
```

Usar vectores para facilitar la implementación de los 3 semáforos requeridos.
También se puede usar un único set de semáforos o bien tres semáforos independientes.
Una vez implementados los puntos 7.1), 7.2) y 7.3) podrá obtener una salida como esta:

```
consola server:
debian:/home/grchere/gccwork/src/sem# ./tp85p
main():inicio proceso productor!
main():para salir envie se?al SIGUSR2 a proceso 2922
main():solicito clave ipc
main():creo memoria compartida de 360 bytes
main():atachar memoria local a memoria compartida 393223
main():Inicializo memoria compartida a cero
main():genero total de ventas de sucursales en memoria compartida
main():solicito claves ipc para semaforos
main():creo semaforos
main():inicializo semaforos
main():Bloqueo proceso hasta recibir se?al de salida
...
mostrar_ventas(): sucursal[0]=160157.865768
mostrar_ventas(): sucursal[1]=151184.964007
mostrar_ventas(): sucursal[2]=155957.583394
mostrar_ventas(): sucursal[3]=150512.363443
mostrar_ventas(): sucursal[4]=149154.134820
mostrar_ventas(): sucursal[5]=148647.715444
mostrar_ventas(): sucursal[6]=148937.294108
mostrar_ventas(): sucursal[7]=134875.590290
mostrar_ventas(): sucursal[8]=149356.049756
mostrar_ventas(): sucursal[9]=151668.660699
 4 7/10/2009      1865      119.61
 727/ 9/2009      1359      149.10
```

```
 917/ 9/2009      1053      160.33
 918/ 4/2009       808      178.43
 2 7/11/2009       970      108.02
 3 7/11/2009      2000      211.54
 012/11/2009       806      130.30
 320/ 4/2009      2087       93.76
 026/ 4/2009       845       53.79
 321/ 3/2009      1445       94.11
...
main():borro semaforos
main():Desconecto memoria compartida
main():Destruyo memoria compartida
main():fin proceso productor! retorno=0
debian:/home/grchere/gccwork/src/sem#
```

----------x----------

consola cliente:

```
....
main():pongo a cero memoria consumida
main():leo memoria compartida
mostrar_ventas(): sucursal[0]=160157.865768
mostrar_ventas(): sucursal[1]=151184.964007
mostrar_ventas(): sucursal[2]=155957.583394
mostrar_ventas(): sucursal[3]=150512.363443
mostrar_ventas(): sucursal[4]=149154.134820
mostrar_ventas(): sucursal[5]=148647.715444
mostrar_ventas(): sucursal[6]=148937.294108
mostrar_ventas(): sucursal[7]=134875.590290
mostrar_ventas(): sucursal[8]=149356.049756
mostrar_ventas(): sucursal[9]=151668.660699
 4 7/10/2009      1865      119.61
 727/ 9/2009      1359      149.10
 917/ 9/2009      1053      160.33
 918/ 4/2009       808      178.43
 2 7/11/2009       970      108.02
 3 7/11/2009      2000      211.54
 012/11/2009       806      130.30
 320/ 4/2009      2087       93.76
 026/ 4/2009       845       53.79
 321/ 3/2009      1445       94.11
main():pongo a cero memoria consumida
...
main():recibi se?al de salida
main():pongo a cero memoria consumida
main():Desconecto memoria compartida
main():fin proceso consumidor! retorno=0
```

----------x----------

consola otra consola, durante el proceso:

```
debian:/home/grchere/gccwork/src/sem# ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere   600        196608     2          dest
0x00000000 65537      grchere   600        196608     2          dest
0x00000000 98306      grchere   600        393216     2          dest
```

```
0x00000000 131075     grchere   600        393216     2          dest
0x00000000 163844     grchere   600        12288      2          dest
0x00000000 196613     grchere   600        393216     2          dest
0x00000000 229382     grchere   600        196608     2          dest
0x68022a15 327687     root      600        360        1

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x73022a15 196608     root      666        1
0x6e022a15 229377     root      666        1
0x65022a15 262146     root      666        1

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

debian:/home/grchere/gccwork/src/sem#
```

al terminar proceso no quedan semaforos asignados:

```
debian:/home/grchere/gccwork/src/sem# ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 32768      grchere   600        196608     2          dest
0x00000000 65537      grchere   600        196608     2          dest
0x00000000 98306      grchere   600        393216     2          dest
0x00000000 131075     grchere   600        393216     2          dest
0x00000000 163844     grchere   600        12288      2          dest
0x00000000 196613     grchere   600        393216     2          dest
0x00000000 229382     grchere   600        196608     2          dest

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages

debian:/home/grchere/gccwork/src/sem#
```

Obsérvese que el productor queda bloqueado mientras está imposibilitado de producir, cuando ya no hay ningún consumidor en ejecución.

Bibliografía

- Stallings, William, "Sistemas Operativos", 5ta. Edición. Prentice Hall. 2001. Madrid.
- Stevens, Richard, "Advanced Programming in the UNIX Environment", Addison-Wesley Professional Computing Series, 1993, ISBN 0-201-56317-7
- LooseSandra Loosemore, Richard M. Stallman, Roland McGrath, Andrew Oram, Ulrich Drepper, "The GNU C Library Reference Manual", Free Software Foundation, 2007, Boston, USA.

Espero que esta práctica haya sido de vuestro agrado y le permita comprender mejor los mecanismos básicos de manejo memoria compartida combinado con la necesidad de implementar mecanismos de exclusión mutua a través del uso -en este caso- de semáforos de tipo SVR4 acorde con lo requerido por procesos de tipo productor - consumidor, según lo visto en la teoría.