

Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Principios de Sistemas Operativos

II Tarea Programada

Daniel Cortés Sáenz, Isaac Ramírez Solano

II Semestre, 2013

## Semáforos utilizados

En el proyecto se uso un semáforo en la memoria compartida principal. El semáforo corresponde a una bandera binaria. La utilización de este semáforo es para que los procesos que acceden a memoria y se apoderan de esta, sin permitirle a otros procesos que ingresen a realizar acciones como leer o escribir. El uso de este semáforo fue debido a que los procesos egoístas y los procesos escritores cuando se apoderan de la memoria no permiten a ningún otro proceso que entre a esta a hacer otra acción.

## Sincronización

La sincronización de procesos se logro gracias al semáforo binario que se implemento en la posición 1 de la memoria. Cada vez que un proceso intenta ingresar a la memoria pregunta si este semáforo esta disponible para acceder, si el semáforo esta en 0 cualquier proceso puede entrar, de lo contrario si el semáforo esta en 1 es porque otro proceso esta en la zona critica y no le permite el acceso a nadie mas.

Si el semáforo es 1 entonces el proceso que solicita el acceso se pone en estado de bloqueado ya que no se le puede dar los recursos que necesita.

Una explicación de cómo logró la sincronización?

## MMAP

La diferencia entre “mmap” y “shmget” usualmente no es muy clara, la diferencia hubiera sido la forma de solicitar la memoria compartida y como accederla. Por ejemplo para acceder a memoria con shmget se hace de la siguiente manera:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
size_t len; // size of data we want
void *buf; // to point to shared data
int shmid;
key_t key = 9876; // or IPC_PRIVATE
shmid = shmget(key, len, IPC_CREAT|0666);
if (shmid == -1) ...
    // error
buf = shmat(shmid, NULL, 0);
```

Contrario al mmap que solicita memoria así:

```
#include <sys/types.h>
#include <sys/mman.h>
size_t len; // size of data we want
void *buf; // to point to shared data
buf = mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_SHARED|MAP_ANON,
-1, // fd=-1 is unnamed 0);
```

## Uso del programa

Ejecutar el archivo MakeFile para generar los ejecutables de los programas:

```
1> make
```

Si la compilación es exitosa, se deben crear los siguientes archivos ejecutables:

```
2> espia finalizador inicializador reader readerEgoista writer
```

El primer programa a ejecutar es el inicializador, recibe por parámetros la cantidad de líneas que va a tener la memoria compartida y el número de procesos que utilizarán la memoria. Este último parámetro es importante porque por debajo se crea otra memoria compartida. Esta memoria será actualizada con la acción que realizan los procesos para que el programa espía pueda saber qué es lo que están haciendo:

```
3> ./inicializador 10 5
```

En el caso anterior, se definió una memoria compartida de 10 líneas y habrán 5 procesos corriendo durante toda la ejecución. Los procesos se distribuirán de acuerdo a cómo se creen los readers, writers y readers egoístas.

El siguiente paso consiste en crear los readers, writers y readers egoístas, cada uno recibe 3 parámetros. El primero consiste en cuántos procesos de ese tipo se crearán. El segundo parámetro especifica el tiempo en que el proceso estará dormido cuando así lo requiera. El último, especifica el tiempo que tardará ya sea escribiendo o leyendo (dependiendo del tipo de proceso):

```
4> ./writer 2 2 2
5> ./reader 2 1 1
6> ./readerEgoista 1 2 2
```

En este caso, se definieron 5 procesos (coincide con la cantidad de procesos especificados al inicializador). Se recomienda utilizar terminales diferentes para todos los programas ya que al iniciar alguna instancia de ellos, actualizan constantemente la consola imprimiendo mensajes.

Para iniciar el espía se debe ejecutar el siguiente comando:

```
7> ./espia
```

Al iniciar el programa se mostrará un menú con tres opciones.

- 0. Estado de la memoria
- 1. Estado de los procesos
- 2. Salir

La opción 0 imprime toda la memoria compartida con su contenido. La opción 1 muestra el estado de todos los procesos en ese instante. Se muestra la información de cada proceso de la siguiente manera PID Tipo Estado Usando memoria. La opción 2 termina el programa espía.

Finalmente, para terminar la ejecución del proyecto, terminar los procesos y liberar los recursos compartidos se debe llamar al finalizador:

```
8> ./finalizador
```

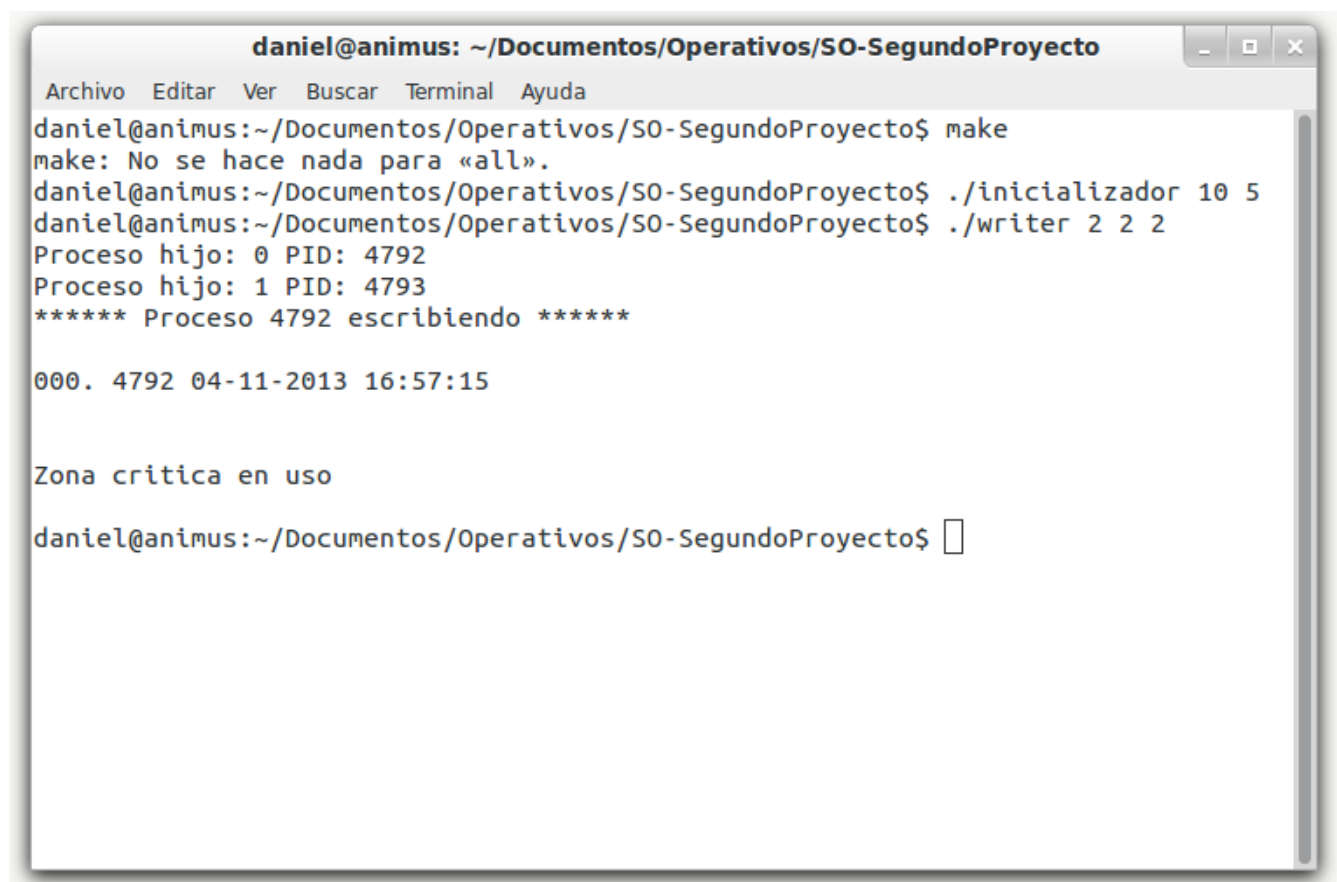
## Análisis de Resultados

Se lograron implementar todos los programas. Con el que más se tuvo problemas fue con el espía pues se definió como otro segmento de memoria compartida al cual los demás procesos escribían. Sin embargo, se logró incorporar al resto de programas.

### Caso de Prueba

Para el caso de prueba seguir el ejemplo del uso del programa.

- 1> make
- 2> ./inicializador 10 5
- 3> ./writer 2 2 2



```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ make
make: No se hace nada para «all».
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./inicializador 10 5
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./writer 2 2 2
Proceso hijo: 0 PID: 4792
Proceso hijo: 1 PID: 4793
***** Proceso 4792 escribiendo *****

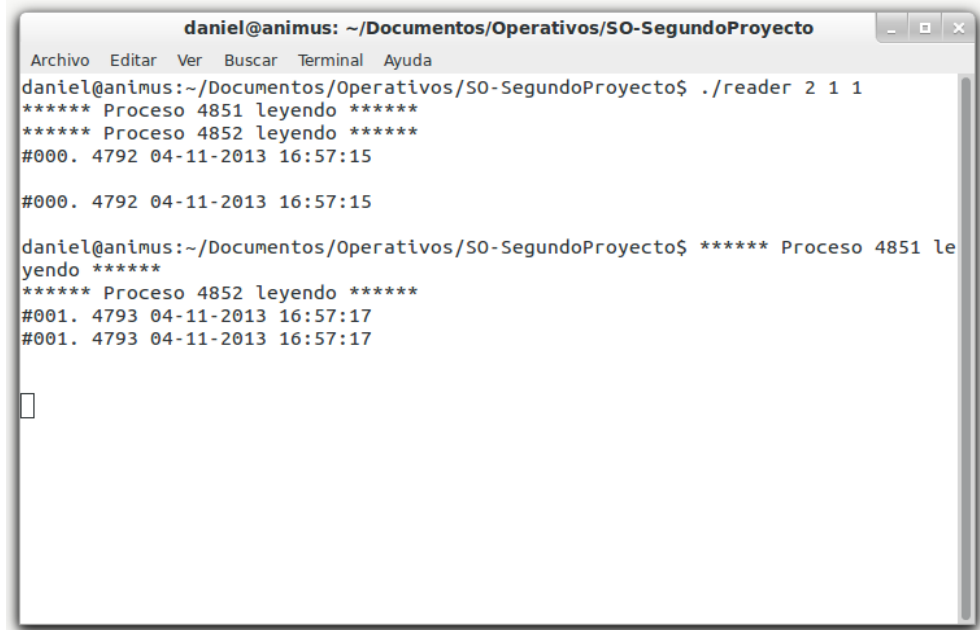
000. 4792 04-11-2013 16:57:15

Zona critica en uso

daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$
```

En este caso, la terminal será actualizada con los mensajes de los writers. Como se puede observar, ya hay un proceso escribiendo y otro está bloqueado debido a que el proceso 4792 está escribiendo.

4> ./reader 2 1 1



```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./reader 2 1 1
***** Proceso 4851 leyendo *****
***** Proceso 4852 leyendo *****
#000. 4792 04-11-2013 16:57:15


#000. 4792 04-11-2013 16:57:15

daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ***** Proceso 4851 le
yendo *****
***** Proceso 4852 leyendo *****
#001. 4793 04-11-2013 16:57:17
#001. 4793 04-11-2013 16:57:17

□
```

En esta otra terminal, se crearon los readers. La consola se actualizará con los mensajes de los readers. En el ejemplo, se puede observar como los dos readers creados ya están leyendo de la memoria compartida.

5> ./readerEgoista 1 2 2

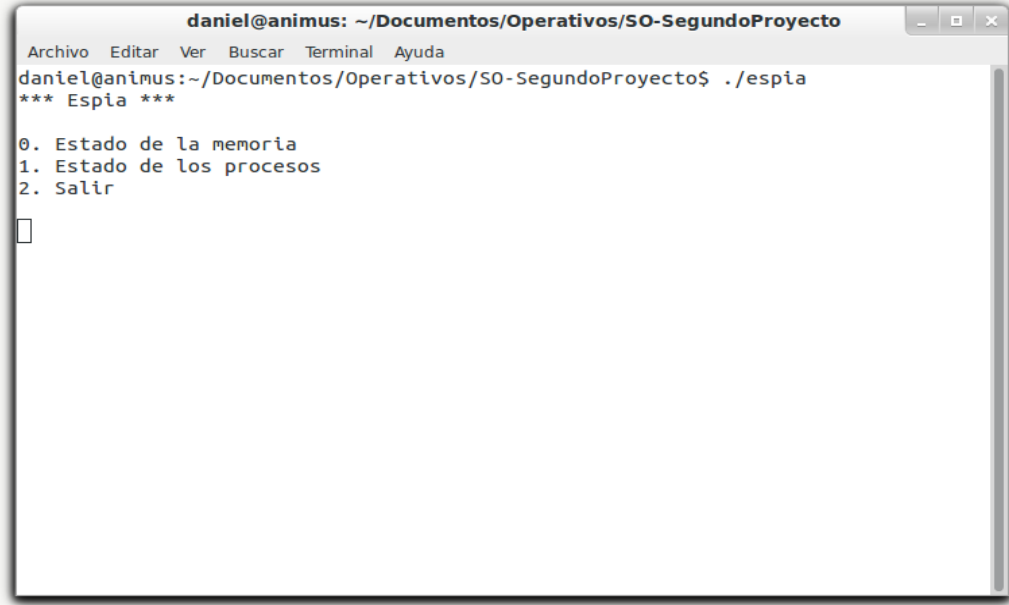


```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./readerEgoista 1 2 2
***** Proceso egoista 4909 leyendo *****
#000. 4792 04-11-2013 16:57:15

daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ Zona critica en uso

□
```

6> ./espia

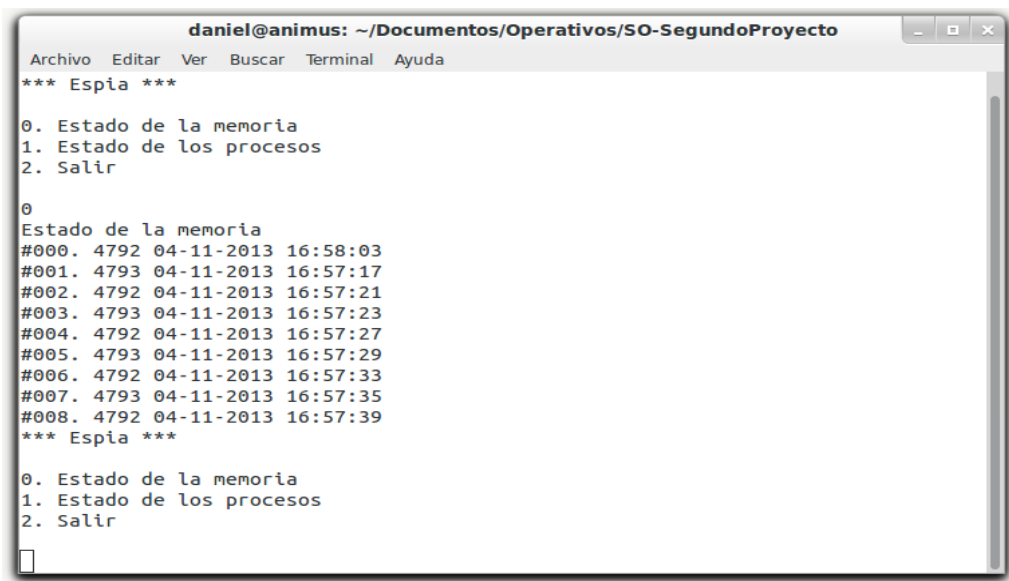


```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./espia
*** Espia ***

0. Estado de la memoria
1. Estado de los procesos
2. Salir


```

En este caso, se consultará primero el estado de la memoria compartida. Para consultar el estado de la memoria, escribir 0 en la consola.



```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda
*** Espia ***

0. Estado de la memoria
1. Estado de los procesos
2. Salir

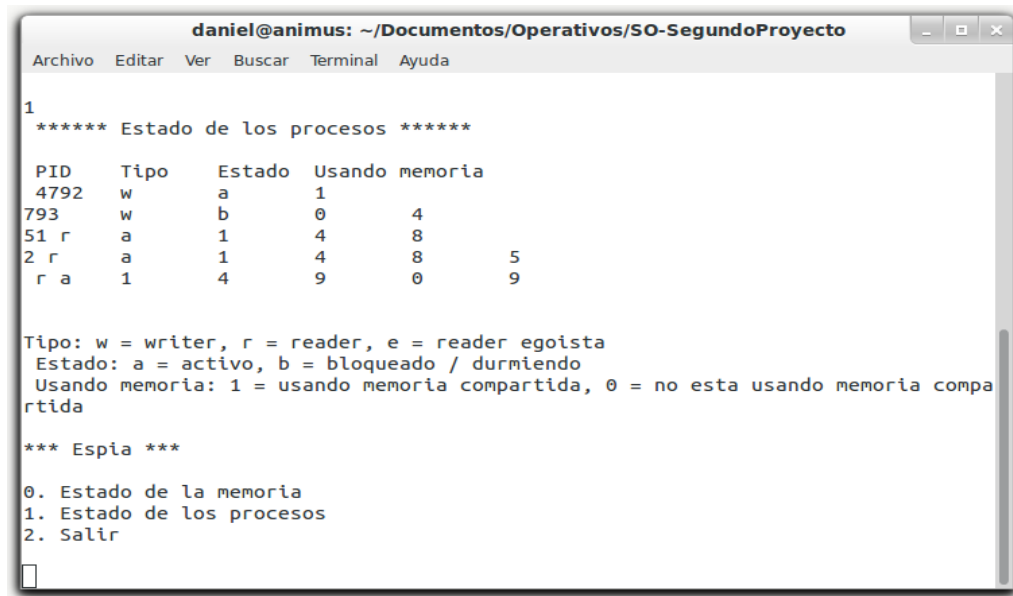
0
Estado de la memoria
#000. 4792 04-11-2013 16:58:03
#001. 4793 04-11-2013 16:57:17
#002. 4792 04-11-2013 16:57:21
#003. 4793 04-11-2013 16:57:23
#004. 4792 04-11-2013 16:57:27
#005. 4793 04-11-2013 16:57:29
#006. 4792 04-11-2013 16:57:33
#007. 4793 04-11-2013 16:57:35
#008. 4792 04-11-2013 16:57:39
*** Espia ***

0. Estado de la memoria
1. Estado de los procesos
2. Salir


```

Para este caso, ya la memoria se encuentra llena debido a que solo se definieron 10 líneas de memoria compartida. En la memoria se puede observar quién fue el que escribió en cada línea, la fecha y la hora de la acción.

Una vez consultada la memoria, se puede seleccionar otra opción del menú del espía. A continuación, se especifica la opción 1:



```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda

1
***** Estado de los procesos *****

PID      Tipo    Estado Usando memoria
4792     w       a       1           4
793      w       b       0           4
51 r     a       1       4           8
2 r     a       1       4           8       5
r a     1       4       9           0       9

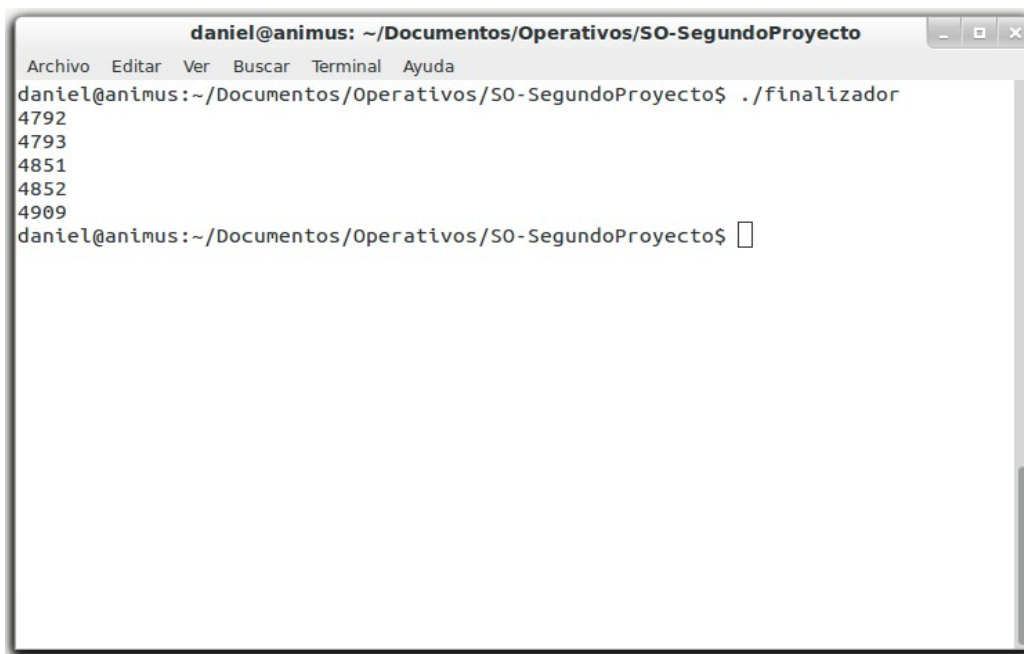
Tipo: w = writer, r = reader, e = reader egoista
Estado: a = activo, b = bloqueado / durmiendo
Usando memoria: 1 = usando memoria compartida, 0 = no esta usando memoria compa
rtida

*** Espia ***

0. Estado de la memoria
1. Estado de los procesos
2. Salir

█
```

7> ./finalizador



```
daniel@animus: ~/Documentos/Operativos/SO-SegundoProyecto
Archivo Editar Ver Buscar Terminal Ayuda

daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ ./finalizador
4792
4793
4851
4852
4909
daniel@animus:~/Documentos/Operativos/SO-SegundoProyecto$ █
```

Al llamar al finalizador, se muestran los PIDs que fueron terminados por este programa y se habrá liberado todos los recursos compartidos utilizados durante la ejecución del proyecto.