CI-0117 Laboratorio 2 Paralela y Concurrente
Rodrigo Vilchez Ulloa | B78292

---

1. Llamados al Sistema Operativo

• El llamado **shmget()** retorna un valor entero que identifica al manejador de la memoria, posee tres parámetros de los cuales el primero es la llave que identifica a la memoria compartida, el segundo es el tamaño de dicha memoria, que generalmente corresponde con la estructura a la que va a ser atada y el tercero son las banderas en caso de que se cree el espacio de memoria o si se utiliza uno ya existente.

• La función **shmat()** lo que hace es atar ese espacio de memoria a un proceso mediante una estrucutra de datos ya existente, para que la interpretación de los datos sea la adecuada al momento de leer o escribir en la memoria. Recibe tres parámetros, donde el primero es el valor entero que retornó la función anterior, seguido de dos parámetros que usualmente se dejan en 0, pues especifican una dirección de mapeo y una bandera para su manejo, de esta forma el sistema se encarga de realizarlo.

• Por otro lado, **shmdt()** se encarga de desatar la memoria, recibe un único argumento que es el puntero de la estructura de datos a la cual la memoria fue atada.

• Por último, **shmctl()** se encarga de borrar el segmento de memoria compartido, recibe tres parámetros: el primero es el entero que identifica al manejador de dicha memoria, el segundo es para especificar alguna operación que se quiera realizar sobre dicho segmento y el tercero es un puntero a una estructura que es útil para alguna de las operaciones. Algunas de esas operaciones son para enviar la información de la estructura a la que está atada el segmento a otra estructura antes de ser borrada, o incluso enviar a otra estructura la información de los permisos.

2. Correr el programa prueaShM

• Demostración:

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$ ls
Makefile  PiPorSeries.c  pruebaShM  pruebaShM.cc  pruebaShM.o  Semaphore.cc  Semaphore.h  Semaphore.o
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$ ./pruebaShM
Escriba un numero y <ENTER> para continuar ...
La variable compartida es: Area de memoria compartida 2020
1
Variable es Area de memoria compartida 2020
Fin del programa ...
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$
```

3. PiPorSeries con Memoria Compartida

• Para solucionar este problema, lo primero que se hizo fue crear un struct que dentro contiene un array de doubles ya proporcionado por el código del profesor. Mediante la instrucción **shmget()**, se crea un controlado del segmento de memoria y luego es atado al proceso mediante un puntero que referencia a esa estructura. Se modificó el código proporcionado para que, en el momento en que se usara el vector, se usara realmente el puntero que referenciaba a la estructura y por ende al array, para que todos los procesos utilizaran y modificaran el mismo vector, además de usar la sintaxis típica de punteros. Por último, se desata el segmento del proceso y se procede a borrar el controlador para que no haya problema al volver a ejecutar el código.

- Demostración:

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$ gcc PiPorSeries.c
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$ ./a.out
Corriendo PiPorSeries con memoria compartida...
Creating process 3869: starting value 0, finish at 100000
Creating process 3870: starting value 100000, finish at 200000
Creating process 3871: starting value 200000, finish at 300000
Creating process 3872: starting value 300000, finish at 400000
Creating process 3873: starting value 400000, finish at 500000
Creating process 3874: starting value 500000, finish at 600000
Creating process 3875: starting value 600000, finish at 700000
Creating process 3876: starting value 700000, finish at 800000
Creating process 3877: starting value 800000, finish at 900000
Creating process 3878: starting value 900000, finish at 1000000
Valor calculado de Pi es  3.14159  con 1000000 terminos
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2$
```

4. Buzones, recibir y enviar con clases

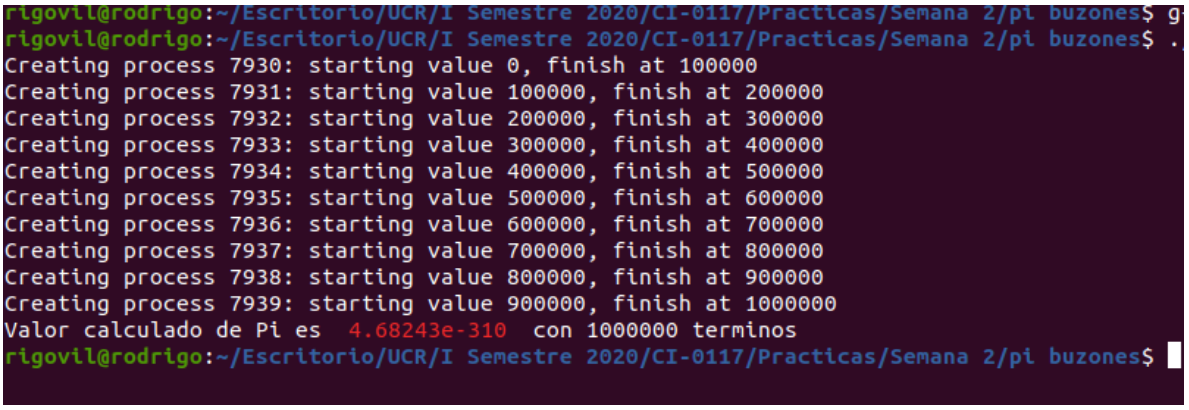
- Se implementaron 3 funciones, la primera el constructor que simplemente creaba el objeto y guardaba el id a partir de la llamada **msgget**. Posteriormente, el método que envía mensajes, lo que hace es tener una estructura msgbuf, con tres atributos los cuales son: el tipo de mensaje, la cantidad de veces y el mensaje como tal guardado en un array. Mediante el llamado a **msgsnd** y recibiendo el puntero del array donde se encuentra el mensaje, primero se crea el struct y se llenan sus atributos para ser enviado mediante esa llamada al sistema. Por último, la función de recibir, utiliza el llamado al sistema **msgrcv**, este recibe el puntero del array que va a almacenar el mensaje. Este método también posee el struct, pues este debe calzar con el struct del mensaje que ha sido enviado. No se implementó el destructor pues al correr el programa enviarConClases, el buzón se destruía al finalizar ese programa, por tanto el programa de recibirConClases no detectaba ninguna cola de mensajes, no encontré una solución a este problema. Por motivos que aun no entiendo, el mensaje se recibe un poco distorsionado con respecto al mensaje real.
- Demostración:

```
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/buzones$ g++ enviarConClases.cc Buzon.cc
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/buzones$ ./a.out
Label: abc, status 0
Label: bcd, status 0
Label: cdef, status 0
Label: defg, status 0
Label: efgh, status 0
Label: li, status 0
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/buzones$ g++ recibirConClases.cc Buzon.cc
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/buzones$ ./a.out
Label:, status 48
Label: b, status 48
Label: cd◆<, status 48
Label: def◆<, status 48
Label: efgh@PQrfU, status 48
Label: ligh@PQrfU, status 48
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/buzones$
```

PiPorSeries con Buzones

⋮

- Se creó una estructura para la emisión y recepción de mensajes, con dos variables: tipo y n, que n va a ser el valor calculado por cada proceso y que luego va a ser tomando en cuenta para calcular el valor de pi. Se utilizaron los llamados al sistema de buzones msgget, msgsnd, msgrcv.
- Demostración:


rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/pi buzones$ g
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/pi buzones$.
Creating process 7930: starting value 0, finish at 100000
Creating process 7931: starting value 100000, finish at 200000
Creating process 7932: starting value 200000, finish at 300000
Creating process 7933: starting value 300000, finish at 400000
Creating process 7934: starting value 400000, finish at 500000
Creating process 7935: starting value 500000, finish at 600000
Creating process 7936: starting value 600000, finish at 700000
Creating process 7937: starting value 700000, finish at 800000
Creating process 7938: starting value 800000, finish at 900000
Creating process 7939: starting value 900000, finish at 1000000
Valor calculado de Pi es 4.68243e-310 con 1000000 terminos
rigovil@rodrigo:~/Escritorio/UCR/I Semestre 2020/CI-0117/Practicas/Semana 2/pi buzones$