Informe Del Ejercicio 1

1. Introducción.

El problema del productor-consumidor consiste en dos procesos que comparten un buffer de tamaño fijo, en el que el productor introduce objetos y el consumidor los extrae. En el caso de que el buffer está vacío el consumidor debe esperar hasta que el productor agregue un elemento, mientras que si está lleno, el productor debe esperar hasta que el consumidor libere espacio quitando un elemento.

El problema aparece debido a la condición de carrera en la variable que almacena la cantidad de elementos que hay en el buffer, en nuestro caso la variable “tope”. Estas carreras críticas generan situaciones en las que el resultado de la ejecución depende del orden de ejecución de los procesos, ya que si los dos procesos acceden y modifican una variable compartida (en este caso “tope”) sin una sincronización adecuada puede generar inconsistencias en el estado del buffer, provocando errores o pérdidas de datos.

En este informe analizamos estas carreras críticas con una implementación en C para hacerlas visibles.

2. Ejecución.

La implementación en C está dividida en dos programas: prod.c (el productor) y cons.c (el consumidor). Generando los ejecutables compilando ambos programas con estos comandos:

```
gcc -o prod prod.c
gcc -o cons cons.c
```

Ejecutando primero el productor y luego el consumidor, en diferentes terminales, de esta manera:

```
./prod número
```

```
./cons número
```

Siendo el "número" el entero pasado por argumento, que es el tiempo de espera (el entero que se le pasa al sleep) en la región crítica, para aumentar la probabilidad de que haya carrera crítica.

Y para detener la ejecución se debe presionar Ctrl+C en las terminales en las que se estén ejecutando los programas, saliendo del bucle e imprimiendo la cadena local con los elementos producidos y consumidos de ambos programas.

3. Funcionamiento.

Estos programas utilizan mmap() para asignar memoria compartida con una estructura Buffer, la cual simula una pila LIFO de tamaño N = 8. El programa productor (prod.c) genera letras aleatorias usando la

función producirItem(), introduciéndolas en el buffer compartido. Si el buffer está lleno hace una espera activa, y aumenta el valor de la variable tope del buffer (la cual nos indica el número de elementos que hay en la pila). Y el programa consumidor (cons.c) obtiene caracteres del buffer compartido con la función removeItem(), para luego retirarlos y guardarlos en un string local con la función consumeItem(). Si el buffer está vacío hace una espera activa, y disminuye el valor de la variable tope.

4. Carreras Críticas.

En estos programas, para aumentar las probabilidades de que sucedieran las carreras críticas, se añadió en lugares claves la función sleep(). En el productor se añadió después de insertar un elemento y antes de incrementar el tope, mientras en el consumidor se colocó después de retirar un elemento y antes de disminuir el tope. Ejecutando estos programas con estas pausas (y de la manera en las que se especifican anteriormente) se pueden observar errores como: valores de tope incorrectos, pérdidas de datos, pérdidas de caracteres en el buffer, acceso a posiciones incorrectas…

Como se puede observar en esta ejecución, en la que primero ejecutamos el productor con 1 segundo en los sleeps y el consumidor con 3, y luego al revés para ver los diferentes outputs:

```
joel@joel-Victus:~/Escritorio/uni/Sistemas Operativos II/practica2/codigos$ ./prod 1
Introduciendo Z en la posicion 0
Introduciendo O en la posicion 1
Introduciendo I en la posicion 2
Introduciendo N en la posicion 3
Introduciendo G en la posicion 3
Introduciendo O en la posicion 4
Introduciendo Q en la posicion 5
Introduciendo C en la posicion 5
Introduciendo O en la posicion 6
Introduciendo F en la posicion 7
Introduciendo S en la posicion 7
Introduciendo L en la posicion 7
Introduciendo I en la posicion 7
Introduciendo W en la posicion 7
Introduciendo C en la posicion 7
Introduciendo W en la posicion 7
Introduciendo T en la posicion 7
Introduciendo W en la posicion 7
^CContenido de str: ZOINGOQCOFSLIWCWTW
```

```
joel@joel-Victus:~/Escritorio/uni/Sistemas Operativos II/practica2/codigos$ ./cons 3
Retirando Z de la posicion 0
Retirando O de la posicion 1
Retirando G de la posicion 3
Retirando C de la posicion 5
Retirando O de la posicion 6
Retirando O de la posicion 6
Retirando O de la posicion 6
Retirando O de la posicion 6
Retirando O de la posicion 6
Retirando O de la posicion 6
^CContenido de str: ZOGCOOOOOO
```

```
joel@joel-Victus:~/Escritorio/uni/Sistemas Operativos II/practica2/codigos$ ./prod 3
Introduciendo X en la posicion 0
Introduciendo S en la posicion 1
Introduciendo J en la posicion 1
Introduciendo Z en la posicion 1
Introduciendo J en la posicion 1
Introduciendo O en la posicion 1
Introduciendo R en la posicion 1
Introduciendo A en la posicion 1
Introduciendo N en la posicion 1
Introduciendo L en la posicion 1
Introduciendo T en la posicion 1
^CContenido de str: XSJZJORANLT
```

```
joel@joel-Victus:~/Escritorio/uni/Sistemas Operativos II/practica2/codigos$ ./cons 1
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
Retirando X de la posicion 0
^CContenido de str: XXXXXXXXXXX
```

El programa consumidor presenta un error en la lectura de los datos en ambas ejecuciones, ya que extrae valores erróneos, generados seguramente por la mala gestión de la variable de tope de la pila, producido por la carrera crítica generada en nuestros programas. Este es uno de los casos de output que se pueden dar en estos programas.

5. Conclusión.

Este ejercicio nos muestra que al producirse carreras críticas en nuestros programas los datos compartidos se ven comprometidos, lo que lleva a resultados incorrectos. En este caso, el consumidor extrae casi siempre el primer valor almacenado en la pila aunque haya más elementos, lo que nos indica que la variable tope está comprometida, siendo incorrecta por culpa del acceso simultáneo sin control de los dos programas. Para solucionar estos problemas se deben usar herramientas como semáforos, que regulen el acceso a la memoria compartida.