Proyecto 1: Consumidor-Productor-Shared Memory

Efrén Carvajal, Alexis Gavriel, Sebastián Víquez
carva97valvi@gmail.com, alexis.gavriel98@gmail.com, sebviro09@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Resumen*—La sincronización en procesos y datos es uno de los conflictos típicos en el ámbito de la computación, y más específicamente en el campo de los sistemas operativos y los procesos concurrentes. Uno de los métodos, que se utiliza para lidiar con dicho problema es el conocido como Productor-Consumidor. En el presente proyecto se implementa un sistema de este tipo, con la creación de un buffer de tamaño finito que sirve de buzón circular, en el que este se encarga de interactuar de manera sincronizada por medio de mensajes escritos o leídos a través de procesos productores o consumidores.**

***Palabras clave*—Buffer, Memoria Compartida, Procesos, Productor-Consumidor, Sincronización, Sistemas Operativos.**

I. Introducción

Desde la aparición de las computadoras programables a través de circuitos que funcionaban de manera mecánica, pasando por la generación de los tubos al vacío y las primeras computadoras digitales, hasta la aparición de los transistores y los circuitos integrados, y por último la integración a gran escala, la historia de la tecnología computacional ha tenido avances significativos de una generación a otra.

Sin embargo, de la mano con el avance en el hardware ha venido el desarrollo de software y la implementación de sistemas operativos que se encarguen de la administración de los recursos de un sistema, desde una perspectiva de abajo hacia arriba. Esta es una tarea compleja, ya que con más recursos, mayor es la complejidad.

Con la aparición de la multiprogramación, es posible ejecutar más de un programa de manera simultánea, a través de un particionamiento, en donde a cada proceso se le asigna un trozo de memoria. De esta manera, dos o más procesos pueden ser ejecutados concurrentemente por el procesador.

Es así como aparece la necesidad de sincronización entre los procesos, que establezca de cierta forma un orden o protocolo, que vaya conforme al uso de recursos del sistema y otras operaciones tales como lectura y escritura.

Uno de los problemas clásicos en la sincronización de procesos múltiples concurrentes es el de Productor-Consumidor, propuesto por Edsger Dijkstra [1], en el que se describen tres tipos de elementos: el productor, el consumidor y el buffer.

El productor se encarga de generar datos, almacenarlos en el buffer y generar nuevos, mientras que el consumidor, al mismo tiempo, tiene la tarea de tomar productos uno a uno del buffer. Este buffer común es de tamaño finito y es usado como una cola. El problema de esto consiste en que el productor no sobrepase la capacidad del buffer y que el consumidor no intente tomar un producto si el buffer está vacío [2].

Un caso especial del problema Consumidor-Productor es cuando el buffer está lleno. En el caso del productor, es puesto a "dormir" o debe descartar los datos, mientras que el consumidor, una vez que remueva un ítem del buffer, llama al productor para que esté en modo activo y pueda llenar el buffer de nuevo. En contraparte, si el buffer está vacío el consumidor es puesto a dormir, y el productor es el encargado de llamarlo una vez que coloca un dato en el buffer.

La solución al problema mencionado anteriormente es obtenida por medio de comunicación entre procesos (IPC), el cual engloba los mecanismos provistos por el sistema operativo para sincronización y manejo de datos compartidos entre los procesos. Una de las formas más utilizadas es mediante lo que se conoce como *semáforos*.

Un semáforo es una variable especial usada para controlar el acceso a un recurso compartido por parte de múltiples procesos en un sistema concurrente, en donde exista multiprocesamiento [3]. Por tanto, los semáforos son capaces de resolver el problema de las "llamadas perdidas", que se da cuando el buffer está completamente vacío o lleno, y el consumidor o el productor intentar leer o escribir en el buffer, respectivamente.

II. Ambiente de desarrollo

II-A. C

Es un lenguaje de programación de propósito general asociado a la implementación de sistemas operativos UNIX para crear software de sistema, aunque también se utiliza para crear software de aplicación que abarcan desde computadoras personales hasta sistemas embebidos. Por esto, una de sus principales características es la portabilidad, debido a que no depende del hardware.

C es considerado como un lenguaje de medio nivel, esto debido a que dispone de estructuras típicas de los lenguajes de alto nivel, pero también tiene la facilidad de que posee construcciones del lenguaje que permiten el control a bajo nivel. Por lo tanto, para los propósitos de este proyecto el

lenguaje C es ideal, ya que provee mecanismos de bajo nivel de acceso y manejo de memoria y una cercanía a instrucciones máquina.

II-B. Make

Make es una herramienta de gestión de dependencias que automáticamente construye programas ejecutables y bibliotecas provenientes de un cierto código fuente, a partir de ciertas reglas y comandos especificados en archivos llamados *Makefiles*.

En el ámbito de este proyecto, se hace uso de Makefile en la construcción de los archivos ejecutables para el funcionamiento del sistema, especificando en este dependencias, parámetros y banderas de compilación, para generar de forma automatizada los ejecutables.

II-C. Visual Studio Code

Es un editor de código fuente cuyas características incluyen soporte para depuración, resaltado de sintaxis, autocompletado, refactorización y control de versiones de código para la mayoría de lenguajes de programación más comunes, como en el caso de C, que es uno de ellos. Además, posee una terminal en consola de Linux, que permite la compilación y ejecución de código, así como ejecutar instrucciones necesarias para el control de versiones en Git.

II-D. Terminal Linux

Es una consola de sistema que provee una manera de comunicación entre el usuario y el kernel, mediante la ejecución de comandos en una terminal. Mediante esta herramienta, los sistemas Linux proveen mecanismos para ejecución de código fuente, en este caso la compilación y ejecución del programa en la consola.

II-E. Git

Git es un sistema distribuido de control de versiones cuya función es el rastreo de cambios en el código fuente por los programadores en la etapa de desarrollo de software.

III. Detalles del diseño del programa

III-A. Inicializador

Es el proceso que se encarga de crear e inicializar el buffer compartido de tamaño finito, los semáforos y variables globales como contadores. Además tiene a su cargo crear y mapear los objetos de memoria compartida.

Cuadro I: Parámetros de entrada del inicializador

Flag	Descripción
`-h (--help)`	Ayuda
`-b (--buffer_name)`	Nombre del buffer
`-s (--size)`	Cantidad máxima de mensajes del buffer

III-B. Productor

El productor se encarga de repetir un ciclo de producción de mensajes que son puestos en el buffer después de un tiempo de espera aleatorio, que posteriormente serán consumidos.

Es posible crear n cantidad de productores que están asociados al mismo buffer, que anteriormente fue creado en el proceso del inicializador. Estos compiten por colocar el mensaje en el buffer, esperando un tiempo aleatorio con distribución exponencial.

De esta manera, una vez se da la creación de la instancia, se tiene la dependencia del tiempo de espera y la limitante del espacio disponible en el buffer, de manera que el productor queda suspendido hasta que se libere espacio en el buffer (por un consumidor), en caso de que esté lleno.

Cuadro II: Parámetros de entrada del productor

Flag	Descripción
`-h (--help)`	Ayuda
`-b (--buffer_name)`	Nombre del buffer asociado.
`-s (--seconds)`	Tiempo en segundos del algoritmo de tiempo de espera.

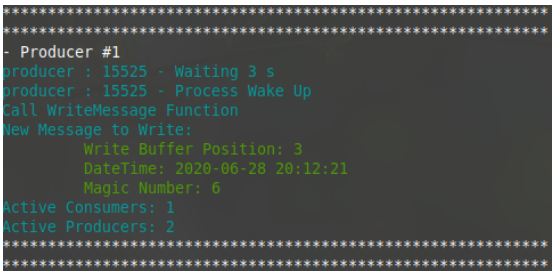

Figura 1: Formato de mensajes del buffer.

III-C. Consumidor

Estos procesos se encargan de consumir mensajes puestos en el buffer por los productores, con un tiempo de espera aleatorio dado una vez que se ejecuta la instancia.

En el caso en el que el buffer esté vacío, el consumidor está suspendido, y vuelve a ser activado por el productor una vez que este crea un proceso y llena el buffer con mensajes.

Hay dos modos de operación: uno automático que va creando mensajes basado en los tiempos de espera, mientras que el manual va consumiendo mensajes cada vez que se presiona la tecla *Enter*.

Cuadro III: Parámetros de entrada del consumidor

Flag	Descripción
`-h (--help)`	Ayuda
`-b (--buffer_name)`	Nombre del buffer
`-s (--seconds)`	Tiempo en segundos del algoritmo de tiempo de espera.
`-m (--mode)`	Modo de ejecución.

III-D. Finalizador

Se encarga de cancelar todo el sistema de procesos, por medio de envío de mensajes a través de una variable global de finalización a los productores y los consumidores. Posteriormente, el buffer es liberado y se muestran las estadísticas de la ejecución.

Cuadro IV: Parámetros de entrada del finalizador

Flag	Descripción
`-h (--help)`	Ayuda
`-b (--buffer_name)`	Nombre del buffer
`-f (--forcefree)`	Forzar liberación de memoria global

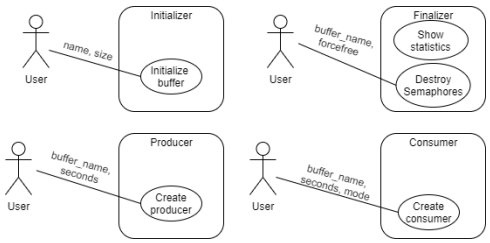

Figura 2: Diagrama de casos de uso.

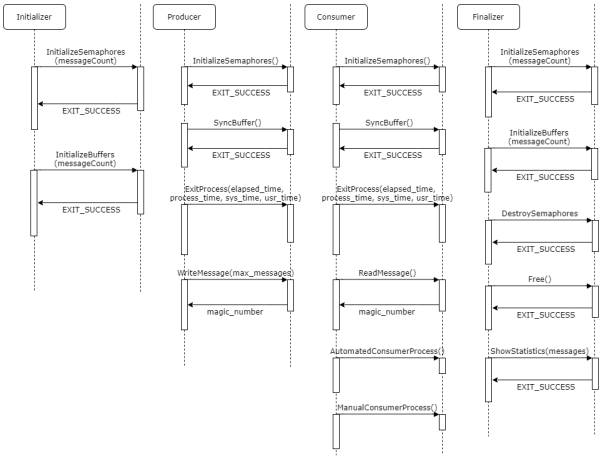

Figura 3: Diagrama de secuencia del sistema.

III-E. Organización del buffer

Cuadro V: Características de los mensajes

Mensajes	Tamaño (Bytes)	Tipo
Process ID	4	`pid_t`
Fecha y Hora	8	`time_t`
Número Mágico	2	`short int`
Message	20	`char`
Total	34	

Cuadro VI: Características de variables globales

Variables Globales	Tamaño (Bytes)	Tipo	Máximo Valor
Cantidad de consumidores	4	`int`	4294967296
Cantidad de productores	4	`int`	4294967296
Límite de mensajes	4	`int`	4294967296
Posición leído por última vez	4	`int`	4294967296
Posición escrito por última vez	4	`int`	4294967296
Tiempo total	4	`double`	4294967296
Histórico total de mensajes	4	`int`	4294967296
Histórico de productores	4	`int`	4294967296
Histórico de consumidores	4	`int`	4294967296
Consumidores eliminados por llave	4	`int`	4294967296
Sumatorio del histórico de tiempos de espera	4	`double`	4294967296
Sumatorio del histórico de tiempo bloqueado total	4	`double`	4294967296
Sumatorio del histórico de tiempo de usuario total	4	`double`	4294967296
Sumatorio del histórico de tiempo de kernel total	4	`double`	4294967296
Mensaje de finalización	4	`short int`	65536
Total	58		

IV. Manual de Instrucción

IV-A. Compilación y Generación de Ejecutables

En la carpeta raíz del sistema se tiene un archivo *Makefile*, el cual contiene las dependencias y comandos necesarios para la compilación del código fuente.

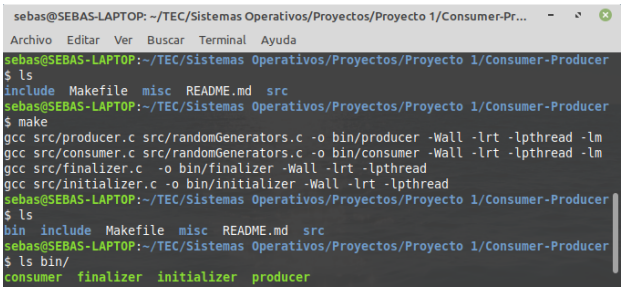

Figura 4: Ejecución del comando `make` para compilación y generación de los ejecutables.

Una vez ejecutado el comando `make`, se puede observar la creación del directorio *bin* en la carpeta raíz. Dentro de este directorio se encuentran cuatro archivos ejecutables: *consumer*, *finalizer*, *initializer* y *producer*, como se muestra en la fig. 4.

Con el *flag* `-h` o `--help` se muestra una ayuda acerca de los parámetros requeridos para la ejecución de cada uno de estos archivos ejecutables.

IV-B. Inicializador

Para la ejecución del Inicializador, se debe ejecutar el archivo ejecutable *initializer* contenido dentro del directorio *bin*. Requiere como parámetros `-b [nombre del buffer]` y `-s [tamaño del buffer]` tal y como se muestra en la fig. 5.

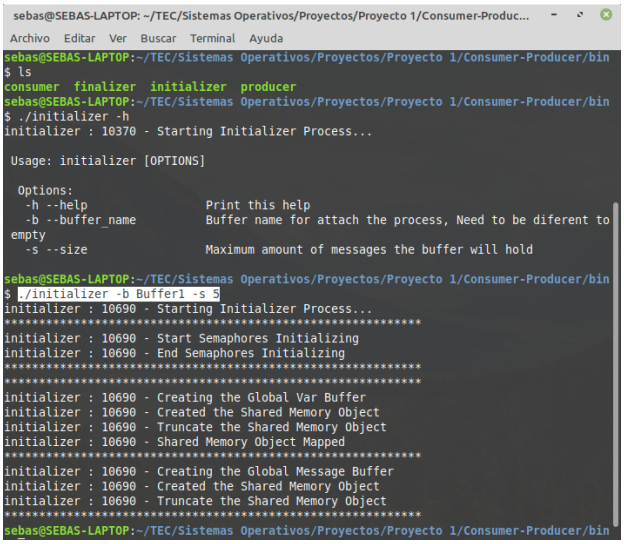

Figura 5: Ejemplo de Inicializador con la ejecución del archivo ejecutable *initializer* y parámetros de entrada.

IV-C. *Productor*

En el caso del productor, se utiliza el archivo *producer*, que como se muestra en la fig. 6 toma como parámetros `-b [nombre del buffer]` y `-s [tiempo]` (en segundos para el tiempo de espera del algoritmo aleatorio).

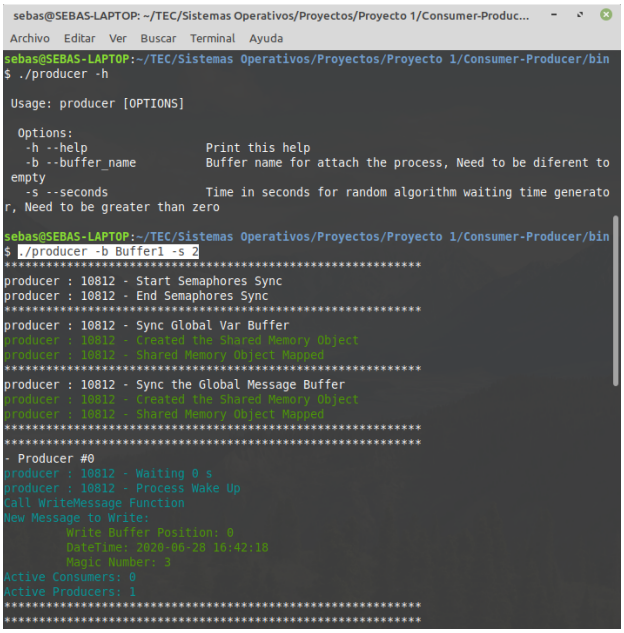

Figura 6: Ejemplo de Productor con la ejecución del archivo ejecutable *producer* y parámetros de entrada necesarios.

IV-D. *Consumidor*

Para la creación de un consumidor, es necesario ejecutar el archivo *consumer* con los parámetros `-b [nombre del buffer]`, `-s [tiempo]` (en segundos para el tiempo de espera del algoritmo aleatorio) y `-m [modo de ejecución]` (M=manual o A=automático).

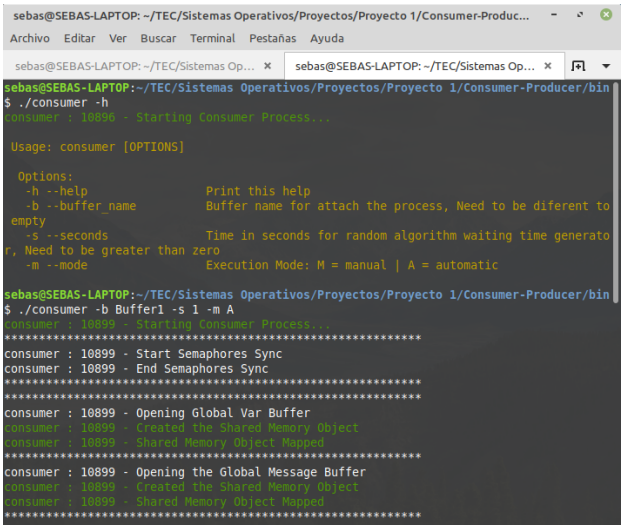

Figura 7: Ejemplo de Consumidor con la ejecución del archivo ejecutable *consumer* y parámetros de entrada necesarios.

Una vez que se ejecuta el consumidor, este va vaciando el buffer de acuerdo a los mensajes que se han producido. Así, tal y como se ve en la fig. 8, se actualiza la estadística de consumidores activos en el proceso del consumidor.

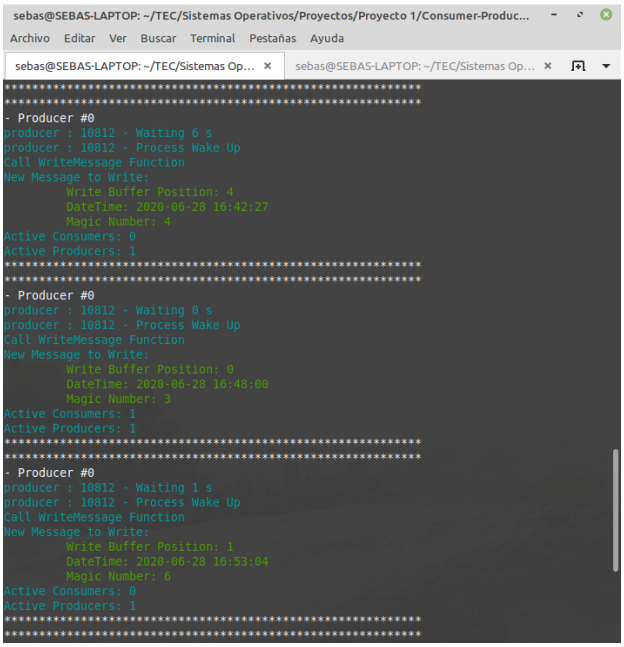

Figura 8: Visualización de consumidor activo por parte del proceso productor.

Posteriormente, una vez que haya consumido todos los mensajes del buffer, finaliza y muestra las estadísticas.

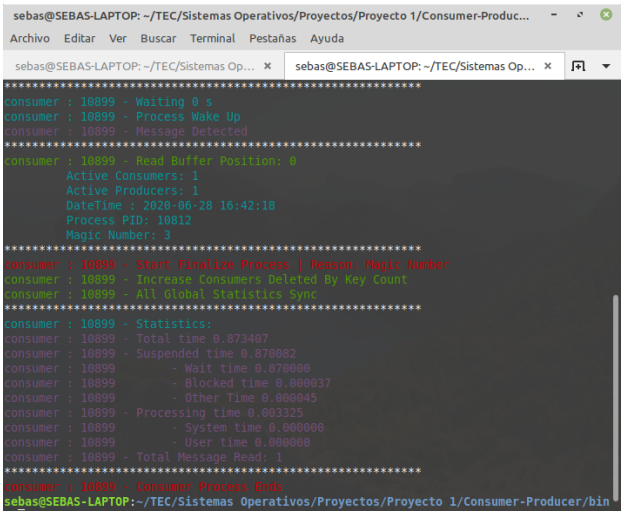

Figura 9: Finalización de proceso consumidor y visualización de estadísticas.

IV-E. Finalizador

Cuando el proceso finalizador es llamado, se debe indicar el nombre del buffer después de ejecutar el archivo ejecutable *finalizer*.

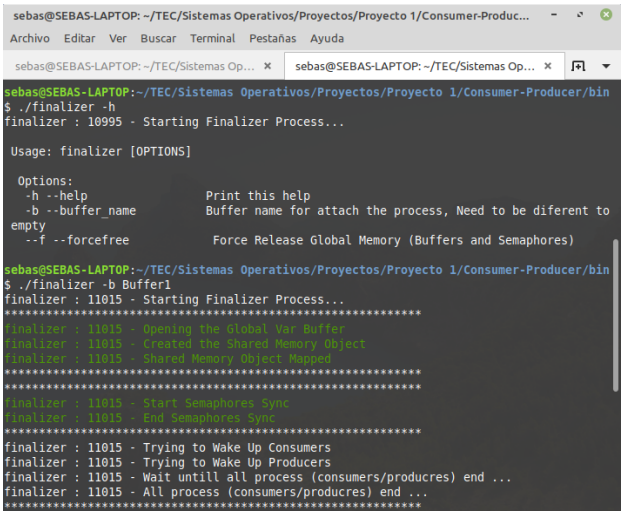

Figura 10: Ejemplo de ejecución del proceso finalizador.

En el proceso del productor se puede ver como una vez ejecutado el finalizador, los productores se detienen por medio de la bandera global compartida, tal y como se ve en la fig. 11.

Además, se muestran las estadísticas relativas a mensajes y tiempos (fig.12).

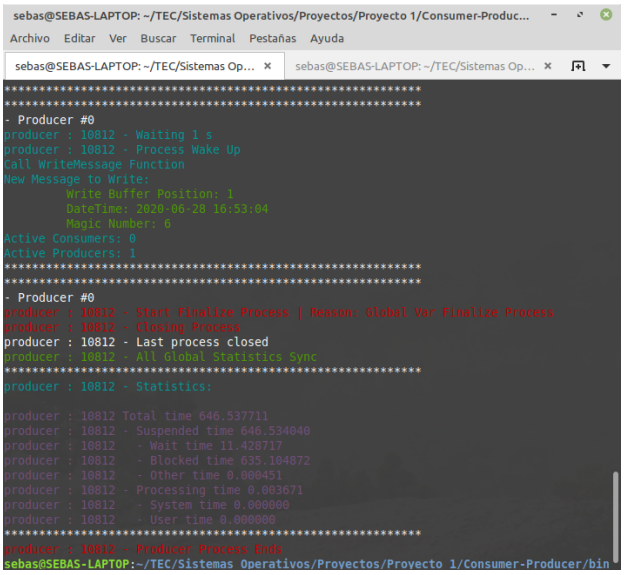

Figura 11: Finalización de productor.

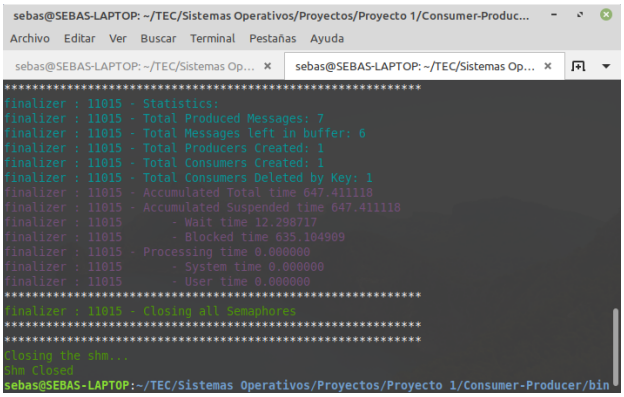

Figura 12: Visualización de estadísticas finales.

V. Bitácora

La bitácora puede ser visualizada en el siguiente link: Archivo de Bitácora

VI. Conclusiones

Dentro de un proceso, los hilos comparten memoria correspondiente al proceso, sin embargo en algunas ocasiones es necesario compartir memoria entre procesos. Para estos casos una alternativa común es solicitar al sistema operativo un segmento de memoria que pueda ser compartido entre dichos procesos, de manera que mediante un objeto de memoria compartida los procesos puedan hacer uso de este espacio.

La sincronización entre procesos es de suma importancia en el ámbito de los sistemas operativos, sobre todo en casos en donde procesos concurrentes comparten recursos, tales como la memoria. Mediante mecanismos como los semáforos es posible preservar la integridad y consistencia del sistema, ya que con la condición de exclusión mutua garantiza que solamente un proceso está usando el recurso y solamente él se encuentra en su región crítica.

VII. Sugerencias y recomendaciones

- El lenguaje de programación C junto con los sistemas operativos Linux brindan herramientas y funciones que permiten acceso y manejo de memoria, de manera que es posible interactuar en un bajo nivel con los procesos.

- En temas de sincronización de procesos es importante realizar un diseño e implementación robusta, que logre lidiar con diversas combinaciones de eventos, y en caso de algún error o fallo que tenga la capacidad de terminar de buena manera y recuperarse.

Referencias

[1] Dijkstra, E. W. "Information streams sharing a finite buffer". Information Processing Letters 1.5 (1972): 179-180.
[2] Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), *Operating Systems: Three Easy Pieces* [Chapter: Condition Variables]
[3] Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014), *Operating Systems: Three Easy Pieces* [Chapter: Semaphores]
[4] "sem_unlink(3): remove named semaphore - Linux man page", Linux Man Page, 2020. [Online]. Available: https://linux.die.net/man/3/sem_unlink. [Accessed: 29-Jun- 2020].
[5] "How to use POSIX semaphores in C language - GeeksforGeeks", GeeksforGeeks, 2020. [Online]. Available: https://www.geeksforgeeks.org/use-posix-semaphores-c/. [Accessed: 29- Jun- 2020].
[6] "Productor - Consumidor con Semáforos", Narkive, 2020. [Online]. Available: https://es.comp.os.linux.programacion.narkive.com/jxXsrNjG/productor-consumidor-con-semaforos. [Accessed: 29- Jun- 2020].
[7] "shm_open(3) - Linux manual page", Man7.org, 2020. [Online]. Available: https://www.man7.org/linux/man-pages/man3/shm_open.3.html. [Accessed: 29- Jun- 2020].