Correia, M. B.
Pág. 253-267
Recibido: 14/07/2019
Aceptado: 19/10/2019
Publicado: 09/12/2019

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

GARANTIZANDO EL ACCESO CONCURRENTE DE MÚLTIPLES PROCESOS A UN BUFFER LIMITADO

ENSURING CONCURRENT ACCESS OF MULTIPLE PROCESSES TO A LIMITED BUFFER

Autor: Mauro Bruno Correia Barbosa

Institución: Universidad de Ciego de Ávila Máximo Gómez Báez, Cuba

Correo electrónico: barbosa@unica.cu

RESUMEN

Un sistema de multiprocesamiento consiste en una computadora que tiene más de un procesador (o bien en una serie de computadoras con *CPU*[1]*'s* o procesadores independientes). La mayoría de computadoras supervisoras se diseñan específicamente para dar soporte a múltiples procesadores. Incluyen un *bus*[2] de altas prestaciones, decenas de MB para memoria con corrección de errores, sistemas de disco redundantes, arquitecturas avanzadas de sistemas que reducen los cuellos de botella utilidades redundantes, como múltiples fuentes de alimentación. Sin embargo, el multiprocesamiento y la necesidad de compartir ciertos recursos del sistema, trajeron como consecuencia un alto nivel de concurrencia y con ello la necesidad de establecer una adecuada comunicación entre los procesos en ejecución. Es por ello que el presente trabajo tiene como objetivo la creación de una aplicación que demuestre visualmente el funcionamiento de la multiprogramación para darle solución al problema del productor-consumidor. Donde se especifica como el procesador maneja los procesos en cola para asegurar que ambos productos se ejecutan simultáneamente y se «despiertan» o «duermen» según el estado del *buffer*. Todo lo cual es de interés para el mejor aprovechamiento de la multiprogramación en los equipos de cómputo y ayuda a comprender mejor el problema del productor-consumidor que es un ejemplo clásico de problema de sincronización de

[1] Unidad central de procesamiento
[2] Es un sistema digital que transfiere datos entre los componentes de una o varias computadoras

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

multiprocesos.

Palabras Clave: Concurrencia, Multiprocesamiento, Multiprogramación, Recursos, Sincronización.

ABSTRACT

A multiprocessing system consists of a computer that has more than one processor (or a series of computers with CPU's or independent processors). Most supervisory computers are designed specifically to support multiple processors. They include a high-performance bus, dozens of MB for error-corrected memory, redundant disk systems, advanced systems architectures that reduce bottlenecks redundant utilities such as multiple power supplies. However, multiprocessing and the need to share certain resources of the system have resulted in a high level of concurrency and with it the need to establish an adequate communication between the processes in execution. This is why the present work aims to create an application that visually demonstrates the operation of multiprogramming to solve the problem of producer-consumer. Where is specified how the processor handles queued processes to ensure that both products run simultaneously and "wake up" or "sleep" according to the state of the buffer. All of which is of interest for the best use of multiprogramming in computer equipment and helps to better understand the producer-consumer problem which is a classic example of a multiprocessing synchronization problem.

Keywords: Concurrency, Multiprocessing, Multiprogramming, Resources, Synchronization.

INTRODUCCIÓN

La motivación inicial que dio lugar a la programación multihilos es la solución de problemas que conllevan la necesidad de estar ejecutando simultáneamente o pseudo-simultáneamente dos tareas al mismo tiempo por el mismo programa. La programación multihilos es una herramienta poderosa y peligrosa. En máquinas monoprocesador, mediante su uso se consigue un mayor rendimiento efectivo pero menor rendimiento computacional. Esto básicamente quiere decir que el sistema está ocioso una menor parte del tiempo pero que el número de

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

instrucciones útiles ejecutadas por unidad de tiempo desciende ya que al propio proceso de los hilos hay que sumar el tiempo de conmutación entre ellos (Tanenbaum 1998).

La multiprogramación aparece como consecuencia de la posibilidad de tener varios programas cargados en memoria al mismo tiempo. Así cada tarea se ejecuta durante un tiempo, hasta que tiene necesidad de realizar una operación de entrada/salida. A partir de ese momento en un sistema monoprogramado, la *CPU* permanece inactiva (esperando la finalización de la operación E/S) y cuando finaliza este proceso entonces simplemente se ejecuta el siguiente proceso de la cola, pero en un sistema multiprogramado, se le puede asignar el *CPU* a otra tarea o proceso mientras el que se estaba ejecutando espera por la operación de E/S (Tanenbaum 1998, Tanenbaum 2009).

De esta forma se va alternando entre los diferentes procesos cargados en memoria el tiempo de *CPU*, intentando que esta se mantenga ocupada el mayor tiempo posible. El resultado es un mayor uso del procesador, por lo que se incrementa la productividad del sistema (Tanenbaum 1998, Tanenbaum 2009).

Sin embargo, el multiprocesamiento y la necesidad de compartir ciertos recursos del sistema, han traído como consecuencia un alto nivel de concurrencia y con ello la necesidad de establecer una adecuada comunicación entre los procesos en ejecución. Es por ello que el presente trabajo tiene como objetivo la creación de una aplicación que demuestre visualmente el funcionamiento de la multiprogramación para darle solución al problema del productor-consumidor.

Referentes conceptuales

Proceso: es el conjunto de las fases sucesivas de un fenómeno natural o de una operación artificial. Un proceso puede informalmente entenderse como un programa en ejecución. Formalmente un proceso es "Una unidad de actividad que se caracteriza por la ejecución de una secuencia de instrucciones, un estado actual, y un conjunto de recursos del sistemas asociados" (Gonzalez, 2012: 2).

En general, un proceso es un flujo de ejecución, representado básicamente por un contador de programa, y su contexto de ejecución, que puede ser más o menos

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

amplio. Así, un proceso incluye en su contexto el estado de la pila, el estado de la memoria y el estado de la E/S, mientras que un *thread*[3] típico tiene como contexto propio poco más que la pila. En algunos sistemas es posible determinar el contexto propio de un proceso en el momento de su creación, como ocurre con la llamada al sistema clone () de Linux. (Fernandez, 2007). En adelante, sin perder generalidad, se utilizará siempre el término proceso, independientemente de cuál sea su contexto.

Un proceso informático, por otra parte, puede atravesar diferentes estados, y obedecen a su participación y disponibilidad dentro del sistema operativo y surgen de la necesidad de controlar la ejecución de cada proceso. En la medida que un proceso se ejecuta, va cambiando de estado. Los estados en los cuales se puede encontrar un proceso son (Fernandez, 2007, Porto y Gardey, 2015):

- New (nuevo): el proceso es creado.
- *Running* (en ejecución): las instrucciones están siendo ejecutadas.
- *Waiting* (en espera): el proceso espera que ocurra algún evento.
- *Ready* (preparado): el proceso espera que se le asigne un procesador.
- *Terminated* (terminado): el proceso terminó su ejecución.

El estado está definido según la actividad en que se encuentra un proceso

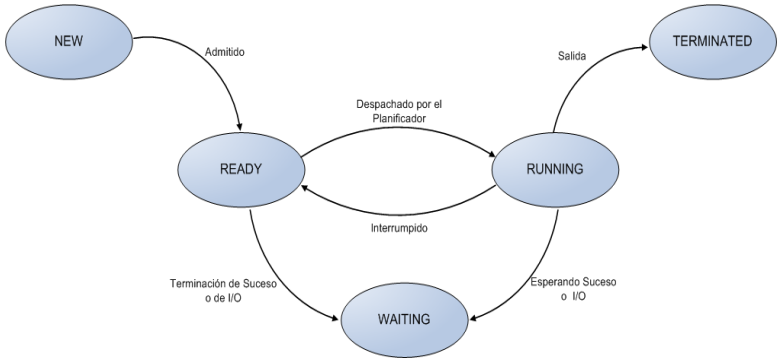

Figura 1. Estados de un proceso

Comunicación y Sincronización de procesos

[3] **hilo de ejecución**, **hebra** o **subproceso** es la unidad de procesamiento más pequeña que puede ser planificada por un SO.

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

El sistema operativo, y los programas de usuario, son un conjunto de procesos que se ejecutan de forma asíncrona. Algunos de ellos cooperan. Cuando es necesaria comunicación entre ellos ha de existir una coordinación o sincronización (Turmero, 2012). La comunicación entre los procesos es necesaria si se desea que varios procesos puedan colaborar para realizar una misma tarea. La Sincronización es el funcionamiento coordinado en la resolución de una tarea encomendada (Domínguez, 2012).

El sistema operativo ofrece mecanismos básicos de comunicación, que permiten transferir cadenas de bytes. Deben ser los procesos que se comunican quienes interpreten el significado de las cadenas transferidas para su labor coordinada. Los mecanismos de comunicación y sincronización son dinámicos. Es decir, cuando se necesita un mecanismo de este estilo, se crea, usa y destruye, de forma que no se establezca de forma definitiva ningún mecanismo de comunicación, ya que ellos podrían producir efectos indeseados. Es decir, la comunicación es algo puntual (Domínguez, 2012).

Los servicios básicos de comunicación son (Domínguez 2012):

- Crear: el proceso solicita la creación del mecanismo
- Enviar o Escribir: el proceso emisor envía información al proceso receptor
- Recibir o Leer: el proceso receptor recibe información
- Destruir: el proceso solicita la destrucción del mecanismo de comunicación

La comunicación puede ser síncrona y asíncrona (Domínguez, 2012):

- Síncrona: los dos procesos han de ejecutar servicios de forma simultánea. El emisor ha de ejecutar el servicio enviar mientras el receptor ejecuta recibir.
- Asíncrona: el emisor hace el envío y prosigue su ejecución. El SO ofrece un almacenamiento intermedio para guardar la información enviada, hasta que el receptor la solicite.

La comunicación entre procesos en un sistema multiprocesador se realiza utilizando memoria compartida. El acceso a esta memoria compartida, o a las variables que contendrá, se debe realizar de forma sincronizada. Los mecanismos

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

de sincronización suelen implementarse mediante rutinas software que descansan en las instrucciones de sincronización proporcionadas por el hardware. Sin esta capacidad, el coste de construir las primitivas básicas de sincronización sería demasiado alto y se incrementaría al incrementarse el número de procesadores. Estas primitivas forman los bloques de construcción básicos para implementar una amplia variedad de operaciones de sincronización a nivel de usuario, incluyendo elementos tales como los cerrojos y las barreras (Llombart, 2012).

Pero, ¿es tan necesaria la sincronización en el acceso a memoria compartida? Para contestar a esta pregunta no tenemos más que mirar el código que aparece a continuación.

Secuencial

```
for (i=0; i<N; i++)
        {
        sum = sum + AA[i];
        }
printf (sum);
```

Versión Paralela

```
for (i=iproc; i<N; i=i+npro)
{
sump = sump + AA[i];
}
sum = sum + sump; /*ojo aquí*/
if (iproc==0) printf (sum)
```

La suma de los elementos de un vector AA ha sido repartida entre n procesadores. Cada proceso tiene un identificador iproc, y se encarga de sumar el contenido de su variable local sump a la variable compartida sum. Si el acceso a esta variable compartida sum no se serializa expresamente, no se podrá garantizar un resultado correcto en la ejecución de este código (Llombart, 2012).

Puede verse la concurrencia de procesos como la ejecución simultánea de varios procesos. Si tenemos un multiprocesador o un sistema distribuido la concurrencia parece clara, en un momento dado cada procesador ejecuta un proceso. Se puede ampliar el concepto de concurrencia si entendemos por procesado concurrente (o procesado paralelo) la circunstancia en la que de tomar una instantánea del sistema en conjunto, varios procesos se vean en un estado intermedio entre su

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

estado inicial y final. Esta última definición incluye los sistemas multiprogramados de un único procesador (Tanenbaum, 2009, Berbesi, 2010).

La concurrencia puede presentarse en tres contextos diferentes (Berbesi, 2010):

- Múltiples aplicaciones: La multiprogramación se creó para permitir que el tiempo de procesador de la máquina fuese compartido dinámicamente entre varias aplicaciones activas.
- Aplicaciones estructuradas: como ampliación de los principios de del diseño modular y la programación estructurada, algunas aplicaciones pueden implementarse eficazmente como un conjunto de procesos concurrentes.
- Estructura del sistema operativo: las mismas ventajas de estructuración son aplicables a los programadores de sistema y se ha comprobado que algunos S.O. están implementados como un conjunto de procesos o hilos.

En un sistema multiprogramado con un único procesador, los procesos se intercalan en el tiempo para dar la apariencia de ejecución simultánea. La multiprogramación tiene ventajas ciertas, pero a su vez crea ciertos problemas (Berbesi, 2010):

1. Compartir recursos globales está lleno de riesgos. Por ejemplo, si dos procesos hacen uso al mismo tiempo de la misma variable global y ambos llevan a cabo tanto lecturas como escrituras de la variable, el orden en que se ejecuten las lecturas y escrituras es crítico.
2. Para el sistema operativo resulta difícil gestionar la asignación óptima de recursos. Por ejemplo, el proceso A puede solicitar el uso y obtener el control, de un canal de entrada-salida particular y suspenderse antes de hacer uso del mismo. No es conveniente que el sistema operativo simplemente bloqueara el canal e impidiera su uso por parte de otros procesos.
3. Resulta difícil localizar un error de programación porque los resultados no son, normalmente ni deterministas reproducibles.

La lección que hay que aprender es que es necesario proteger las variables globales compartidas y otros recursos globales compartidos y que la única forma

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

de hacerlo es controlar el código que accede la variable. Si se impone la norma de que solo un proceso puede acceder a la variable en cada instante y que una vez que la variable el procedimiento debe ejecutarse hasta el final antes de estar disponible para otro proceso, no se producirá el tipo de error expuesto antes (Tanenbaum, 2009, Berbesi, 2010).

Los distintos procesos dentro de un ordenador no actúan de forma aislada. Por un lado, algunos procesos cooperan para lograr un objetivo común. Por otro lado, los procesos compiten por el uso de unos recursos limitados, como el procesador, la memoria o los ficheros. Estas dos actividades de cooperación y competición llevan asociada la necesidad de algún tipo de comunicación entre los procesos (Tanenbaum, 1998, Tanenbaum, 2009).

Dentro de los mecanismos más usados, tenemos el paso por mensajes, la memoria compartida y los semáforos. El paso por mensajes se puede ver como una lista enlazada de mensajes dentro del espacio de direccionamiento del núcleo. Una aplicación, siempre que tenga los derechos necesarios, puede depositar un mensaje (de cualquier tipo) en ella, y otras aplicaciones podrán leerlo. Es posible asignar atributos a los mensajes, de forma que se puedan mantener ordenados por prioridad en lugar de por orden de llegada (Tanenbaum, 1998).

La memoria compartida es un medio que permite establecer una zona común de memoria entre varias aplicaciones. Y los semáforos, que son una herramienta puramente de sincronización. Permiten controlar el acceso de varios procesos a recursos comunes (Tanenbaum, 1998).

MATERIALES Y MÉTODOS

La interacción entre procesos se plantea en una serie de situaciones clásicas de comunicación y sincronización. A continuación se describe una situación y sus problemas para demostrar la necesidad de comunicar y sincronizar procesos en un SO.

Problema del Productor–Consumidor (*Buffer* limitado)

El problema Productor-Consumidor consiste en el acceso concurrente por parte de procesos productores y procesos consumidores sobre un recurso común que resulta ser un buffer de elementos. Los productores tratan de introducir elementos

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

en el buffer de uno en uno, y los consumidores tratan de extraer elementos de uno en uno (Talegaman, 2010).

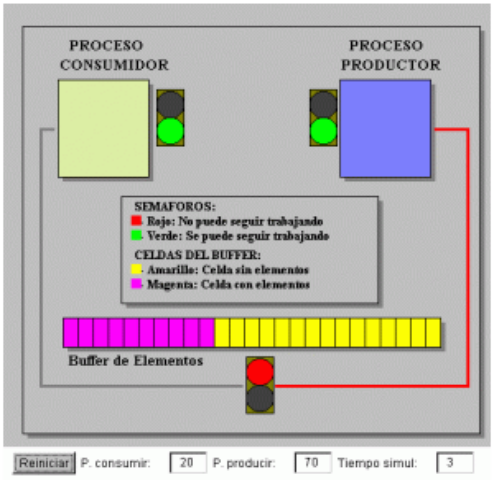

Figura 2. *Problema del Productor-consumidor*

El problema del productor consumidor surge cuando se quiere diseñar un programa en el cual un proceso o hebra produce ítems de datos en memoria que otro proceso o hebra consume (Maldonado, 2012).

Un ejemplo sería una aplicación de reproducción de vídeo (Maldonado, 2012):

- El productor se encarga de leer de disco o la red y descodificar cada cuadro de vídeo.
- El consumidor lee los cuadros descodificados y los envía a la memoria de vídeo para que se muestren en pantalla.

Hay muchos ejemplos de situaciones parecidas (Maldonado, 2012).

- En general, el productor calcula o produce una secuencia de ítems de datos (uno a uno), y el consumidor lee o consume dichos ítems (también uno a uno).
- El tiempo que se tarda en producir un ítem de datos puede ser variable y en general distinto al que se tarda en consumirlo (también variable).

Para asegurar la consistencia de la información almacenada en el *buffer*[4], el acceso de los productores y consumidores debe hacerse en exclusión mutua.

[4] Es un espacio de memoria en el que se almacenan datos de forma temporal

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

Adicionalmente, el *buffer* es de capacidad limitada, de modo que el acceso por parte de un productor para introducir un elemento en el *buffer* lleno debe provocar la detención del proceso productor. Lo mismo sucede para un consumidor que intente extraer un elemento del buffer vacío (Talegaman, 2010).

Figura 3. Estructura clásica de procesos productor-consumidor.

La idea para la solución es la siguiente, ambos productos se ejecutan simultáneamente y se «despiertan» o «duermen» según el estado del *buffer*. Concretamente, el productor agrega productos mientras quede espacio y en el momento en que se llene el *buffer* se pone a «dormir». Cuando el consumidor toma un producto notifica al productor que puede comenzar a trabajar nuevamente. En caso contrario si el *buffer* se vacía, el consumidor se pone a dormir y en el momento en que el productor agrega un producto crea una señal para despertarlo. Se puede encontrar una solución usando mecanismos de comunicación interprocesos, generalmente se usan semáforos. Una inadecuada implementación del problema puede terminar en un *deadlock*[5] donde ambos procesos queden en espera de ser despertados. Este problema puede ser generalizado para múltiples consumidores y productores (Wikimedia, 2015).

Aproximación errónea

Para resolver el problema cualquier programador podría llegar a la solución que se muestra a continuación. En la misma se utilizan dos bibliotecas, *sleep* y *wakeup*.

[5] es el bloqueo permanente de un conjunto de procesos o hilos de ejecución en un sistema concurrente que compiten por recursos del sistema o bien se comunican entre ellos

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

La variable global *itemCount* contiene el número de elementos en el *buffer (Wikimedia, 2015)*.

```
int itemCount = 0;
procedure producer() {
    while (true) {
        item = produceItem();
        if (itemCount == BUFFER_SIZE)
{
            sleep();
        }
        putItemIntoBuffer(item);
        itemCount = itemCount + 1;
        if (itemCount == 1) {
            wakeup(consumer);
        }
    }
}
```

```
procedure consumer() {
    while (true) {
        if (itemCount == 0) {
            sleep();
        }
        item = removeItemFromBuffer();
        itemCount = itemCount - 1;
        if (itemCount == BUFFER_SIZE
- 1) {
            wakeup(producer);
        }
        consumeItem(item);
    }
}
```

El problema con esta aproximación es que puede caer en un *deadlock*, considere el siguiente escenario (Wikimedia 2015):

1. El consumidor acaba de consultar la variable *itemCount*, nota que es cero y pasa a ejecutar el bloque *if*.
2. Justo antes de llamar a la función *sleep*() el consumidor es interrumpido y el productor comienza a trabajar.
3. El productor crea un objeto, lo agrega al *buffer* y aumenta *itemCount*.
4. Como el buffer estaba vacío antes de la última adición el productor intenta despertar al consumidor.
5. Desafortunadamente el consumidor no estaba durmiendo todavía luego la llamada para despertarlo se pierde. Una vez que el consumidor comienza a trabajar nuevamente pasa a dormir y nunca más será despertado. Esto pasa porque el productor solo lo despierta si el valor de *itemCount* es 1.
6. El productor seguirá trabajando hasta que el buffer se llene, cuando esto ocurra se pondrá a dormir también.

Como ambos procesos dormirán por siempre, hemos caído en un *deadlock*. La esencia del problema es que se perdió una llamada enviada para despertar a un proceso que todavía no estaba dormido. Si no se perdiera, todo funcionaría. Una

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

alternativa rápida seria agregar un bit de espera de despertar a la escena. Cuando se envía una llamada de despertar a un proceso que está despierto, se enciende este bit. Después, cuando el proceso trata de dormirse, si el bit de espera de despertar está encendido, se apagará, pero el proceso seguirá despierto. El bit de espera de despertar actúa como una alcancía de señales de despertar. Aunque el bit de espera de despertar soluciona este caso, es fácil construir ejemplos con tres o más procesos en los que un bit de espera de despertar es insuficiente. Se podría agregar un segundo bit de espera de despertar, o quizá 8 o 32, pero en principio el problema sigue ahí (Wikimedia, 2015).

Aproximación adecuada haciendo uso de semáforos

El siguiente pseudocódigo muestra una solución al problema usando semáforos.

Hay un solo emisor y un solo receptor y las operaciones que realizan son enviar y recibir respectivamente. La información que se envía se pasa a través de una variable compartida, en este caso valor, por lo que el buffer de comunicación tiene longitud uno. El protocolo es síncrono, esto quiere decir que el primer proceso, ya sea el productor o el consumidor que llega a la cita tiene que esperar al otro (Amorós, 2008).

```
var primero : boolean;
mutex : semáforo;
segundo : semáforo;
valor : entero;
procedure
enviar(v:entero);
begin
wait(mutex);
valor := v;
if primero then
begin
primero := false;
signal(mutex);
wait(segundo);
signal(mutex);
end
else
begin
primero := true;
signal(segundo);
end
end;
procedure recibir(var v:
entero);
begin
wait(mutex);
if primero then
. . .
else
begin
primero := true;
v := valor;
signal(segundo);
end
end
proceso productor;
var i: entero;
begin
for i := 1 to 10 do
enviar(i)
end;
proceso consumidor;
var i,v : entero;
begin
for i := 1 to 10 do
begin
recibir(v);
writeln(v)
end
end;
begin
primero := true;
init(mutex,1);
init(segundo,0);
cobegin
productor;
consumidor;
```

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

coend *end.*

El texto que corresponde a los puntos suspensivos es el siguiente (Amorós, 2008):

begin
primero := false;
signal
(mutex);
wait
(segundo);
v := valor;
signal
(mutex);
end

RESULTADOS Y DISCUSIÓN

Aplicaciones

El problema del productor consumidor surge cuando se quiere diseñar un programa en el cual un proceso produce *items* de datos en memoria que otro proceso consume (Maldonado, 2012).

- Un ejemplo sería una aplicación de reproducción de vídeo:
- El productor se encarga de leer de disco o la red y descodificar cada cuadro de vídeo.
- El consumidor lee los cuadros descodificados y los envía a la memoria de vídeo para que se muestren en pantalla.

Hay muchos ejemplos de situaciones parecidas (Maldonado, 2012).

- En general, el productor calcula o produce una secuencia de *items* de datos (uno a uno), y el consumidor lee o consume dichos *items* (también uno a uno).
- El tiempo que se tarda en producir un *item* de datos puede ser variable y en general distinto al que se tarda en consumirlo (también variable) (Maldonado, 2012).

Ejemplo programado

En la siguiente figura se muestra una aplicación de escritorio que resuelve el problema del productor consumidor utilizando semáforos. Cuenta con tres contenedores, el primero almacena los procesos que estarán produciéndose continuamente. El segundo contiene los procesos que se han producido y aun no se han ejecutado porque hay algún proceso ejecutándose en ese mismo instante. Finalmente, el tercero muestra los procesos que se están ejecutando.

El tiempo de espera de los procesos ya sean produciéndose o consumiéndose se puede regular con la variable "*waiting*" que va desde 1 a 10 segundos, la variable

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

"*Capacity*" se refiere a la capacidad del segundo contenedor.

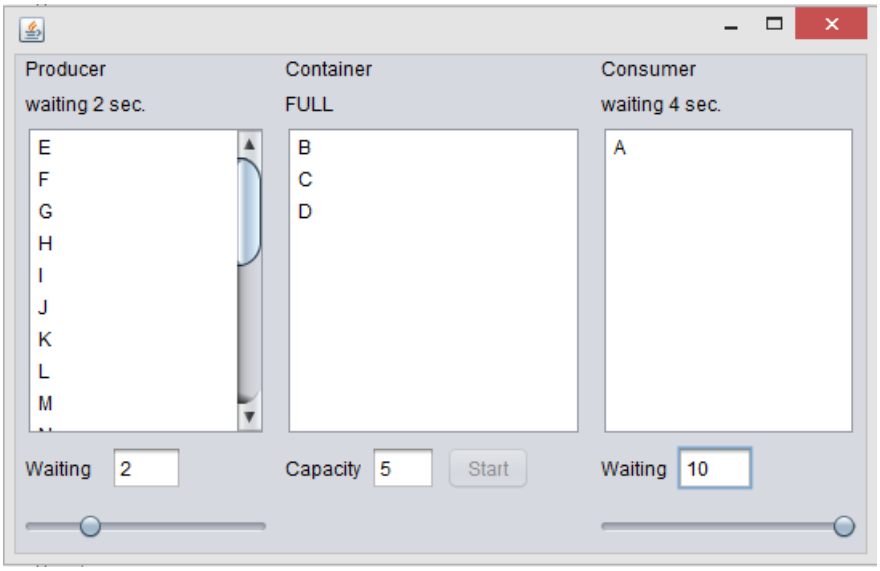

Figura. 3. Aplicación de escritorio que resuelve el problema del productor consumidor.

Al ejecutar la aplicación el usuario debe especificar el tiempo de espera del productor-consumidor y la capacidad del buffer. Si no se especifica, por defecto está definido que el tiempo de espera sea de 10 segundos y la capacidad de 5.

CONCLUSIONES

Durante el transcurso del presente trabajo se desarrollaron diversas actividades de investigación que permitieron ampliar los conocimientos en determinadas áreas de la informática, de ellos se habló del multiprocesamiento, la multiprogramación, procesos, la comunicación y sincronización entre los procesos, se abordó el problema del productor-consumidor y se concluyó hablando de las posibles aplicaciones del algoritmo desarrollado para resolver situaciones reales del problema del productor-consumidor y se obtuvo un software final que ilustra el funcionamiento del algoritmo programado.

BIBLIOGRAFÍA CONSULTADA

AMORÓS, D. F.: *Examen de Programación Concurrente*, 4, 2008.

BERBESI, H.: «Concurrencia, exclusión mutua y sincronización», 14., 2010.

Universidad&Ciencia
Vol. 8, Especial, XLI aniversario de la UNICA, diciembre (2019)
ISSN: 2227-2690 RNPS: 2450
http://revistas.unica.cu/uciencia

DOMÍNGUEZ, J. G.: (2012). «Comunicación entre procesos y sincronización», 2012. Disponible: https://Sites.Google.Com/Site/Sistemasdistribuidosycluster/Comunicacion-Entre-Procesos-Y-Sincronizacion.Visitado en noviembre, 21, 2017.

FERNANDEZ, M. A. P.: "Sincronización entre procesos." 29, 2007.

GONZALEZ, G.: «Los proceso informáticos», 2012. Disponible en: http://losprocesosinformaticos.blogspot.com/. Visitado: noviembre, 20, 2017

LLOMBART, V. A.: *Organización de Computadores: 5.5. Sincronización*, 25. 2012.

MALDONADO, G. M.: «Sistemas Concurrentes y Distribuidos. Práctica 1: Sincronización de hebras con semáforos», 2012. Disponible en: https://github.com/germaaan/trabajos_universidad/blob/master/2GII/SCD/practica_01/README.md. Visitado: marzo, 2017.

PORTO, J. P. Y GARDEY, A.: «Proceso Informático», 2015. Disponible en: https://definicion.de/proceso-informatico/. Visitado noviembre, 20, 2017.

TALEGAMAN: «Problema del productor-consumidor», 2010. Disponible en: https://antologiaso.wordpress.com/2010/04/08/problema-productor-consumidor/ Visitado: febrero, 2017.

TANENBAUM, A.: *Sistemas Operativos - Diseño e Implementación*, En. P. E. Vázquez, PRENTICE HALL HISPANOAMERICA, S.A, 1998.

TANENBAUM, A.: *Sistemas Operativos Modernos*, 2009.

TURMERO, P.: «Concurrencia: exclusión mutua y sincronización. Comunicación entre procesos», 2012. Disponible en: http://www.monografias.com/trabajos106/concurrencia-exclusion-mutua-y-sincronizacion-comunicacion-procesos/concurrencia-exclusion-mutua-y-sincronizacion-comunicacion-procesos.shtml. Visitado: noviembre, 21, 2017.

WIKIMEDIA, F.: «Problema del Productor Consumidor», 2015. Disponible en: https://es.wikipedia.org/wiki/Problema_productor-consumidor. Retrieved febrero, 2017.