INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

ALUMNOS:

ARISTA BOJORGES VICTOR USIEL
HERNÁNDEZ RADILLA JOSÉ ANGEL
VALLEJO SERRANO EHECATZIN

MATERIA:
APLICACIONES PARA COMUNICACIONES EN RED

PRÁCTICA:
MANEJO Y SINCRONIZACIÓN DE SEMÁFOROS

INTRODUCCIÓN

¿Qué es un semáforo?

Un semáforo es una estructura diseñada para sincronizar dos o más threads o procesos, de modo que su ejecución se realice de forma ordenada y sin conflictos entre ellos.
El por qué no se pueden usar directamente otras estructuras más clásicas, como por ejemplo usar una variable común para decidir si se puede o no acceder a un recurso, se debe a que estamos en un sistema multitarea: hacer esto implicaría realizar una espera activa (un bucle, comprobando constantemente si la variable está o no a 0, y así saber si podemos seguir ejecutando o no).
Por otro lado, puede ocurrir algo mucho peor, supongamos que un proceso comprueba la variable, y ve que el recurso está libre, por lo que procedería a cambiar dicha variable de valor y seguir. Pues bien, si justo después de la comprobación pero antes de que cambie el valor se conmuta de tarea (puede pasar, pues el sistema operativo puede hacerlo en cualquier momento), y el nuevo proceso comprueba la variable, como todavía no se ha actualizado, creerá que el recurso está libre, e intentará tomarlo, haciendo que ambos programas fallen. Lo peor del caso es que se tratará de un error aleatorio: unas veces fallará (cuando se produzca cambio de tarea en ese punto) y otras no.

¿Que es una zona crítica?

En este contexto , una zona crítica hace referencia a aquel , o aquellos bloques de código dentro de un programa, en los que se acceden o se asignan valores a variables compartidas.

¿Qué problema resuelve la sincronización?

*Cada hilo puede ser elegido para ejecución independiente.

Entonces las decisiones de :

- Que hilo se ejecuta en cada momento.
- Cuando se produce cada cambio de contexto.

dependen de un planificador y no del programador de la aplicación. Entonces esto puede ocasionar que un código que es correcto si se ejecuta secuencialmente, pueda dejar de serlo cuando se ejecuta concurrentemente, debido a una situación conocida como condición de carrera (race condition).

Una condición de carrera se produce cuando la ejecución de un conjunto de operaciones sobre una variable compartida deja la variable en un estado inconsistente. Además el resultado final almacenado de la variable depende de la velocidad relativa en que se ejecutan las operaciones.

API

SEMÁFOROS POSIX

#include <semaphore.h>

int sem_init(sem_t * sem ,int pshared,unsigned int value);

int_semdestroy(sem_t * sem)

int sem_wait(sem_t *);

int sem_trywait(sem_t *sem);

int sem_post(sem_t *Sem);

int sem_getvalue(sem_t *sem,int *sval);

PROBLEMÁTICA PLANTEADA

Un proceso crea a 5 hilos. Cada uno puede escribir en la otra zona , solo cuando antes escribió en la primera zona. Cada productor primero produce una letra y después un número, esto será igual a una producción.

Primer productor 10 mil veces una A , y luego produce un número = Una producción

Puras letras

[ ]
[ ]
[ ]
[ ]

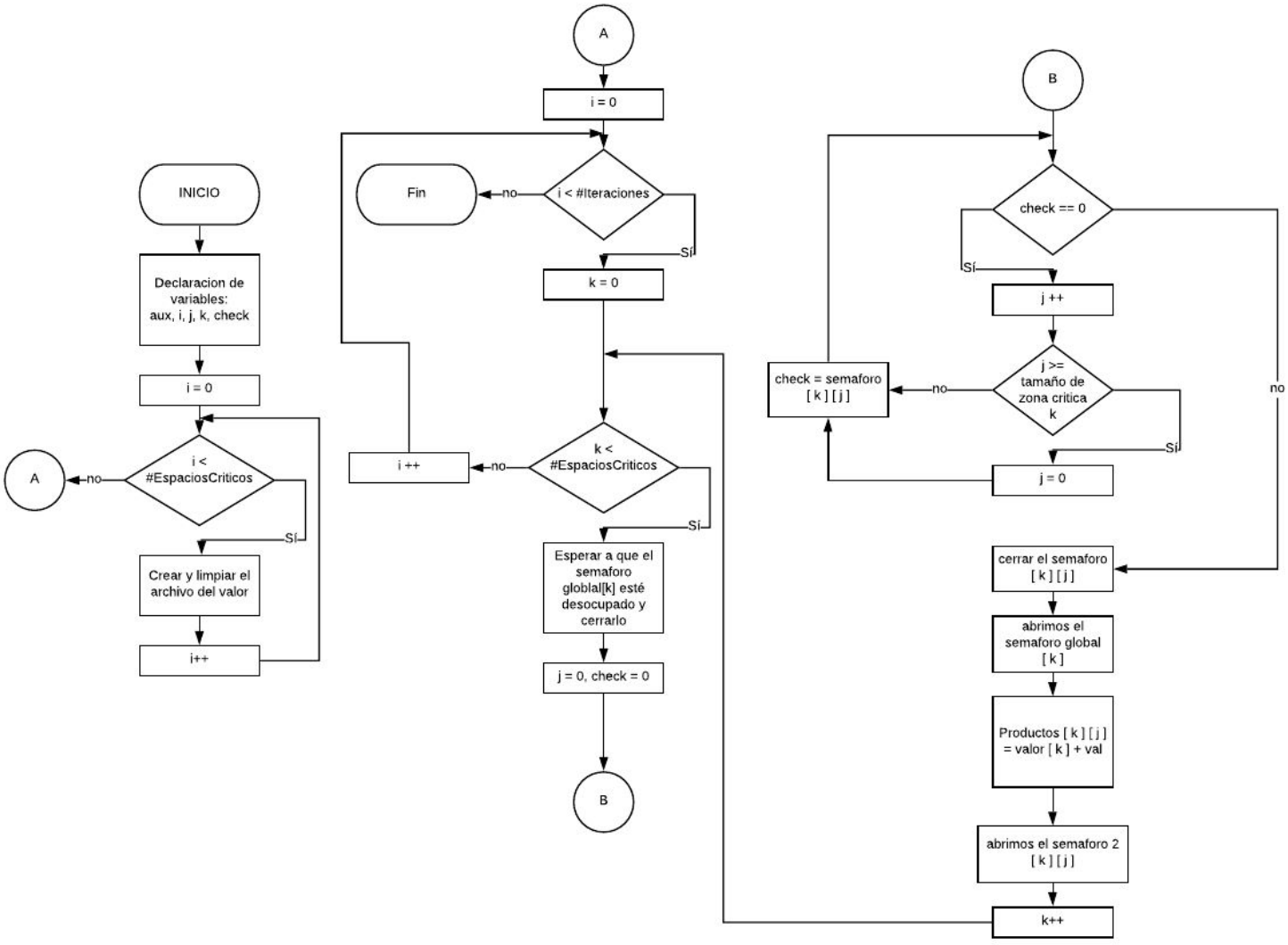
Puros números

[ ]
[ ]
[ ]
[ ]

El objetivo es sincronizar hilos, usando el mecanismo de semáforos. Generando como salida archivos txt, los cuales contienen los elementos producidos por cada consumidor, además de otro,con los elementos consumidos por cada productor.

DIAGRAMA DE FLUJO DE LA SOLUCIÓN

PRODUCTOR:

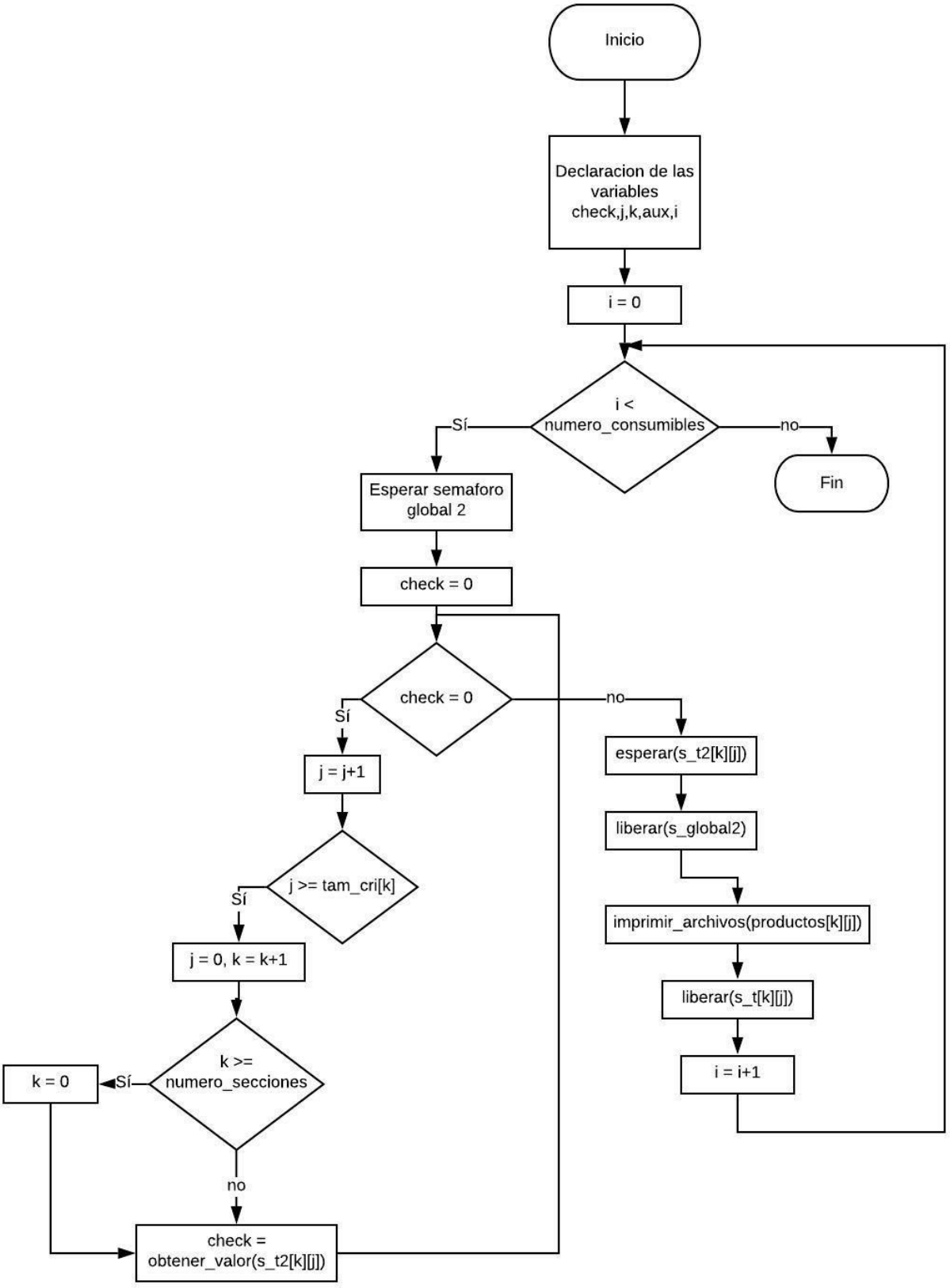


Primeramente se realiza una iteración hasta <n_critica> que indica el número de secciones críticas para inicializar los archivos,

A continuación se realiza una triple iteración, la primera internación interviene en cuántas veces se imprimirá los caracteres seleccionados, estos valores se obtienen mediante la suma del ascii del valor inicial más el ID del hilo, la segunda iteración tiene como objetivo seleccionar la sección crítica en la que le toca escribir, puede ser números o letras, finalmente la tercera iteración busca en qué espacio de memoria es posible escribir, esto se determina sabiendo si el semáforo en cuestión está libre. En caso

de estar libre se realizan todas las acciones necesarias y después se libera el semáforo para que pueda ser utilizado. Así

CONSUMIDOR:



Explicación del consumidor

En el primer bloque de código se tiene un ciclo for que itera el número de consumibles, este bloque está protegido con un semáforo inicializado en uno, si está en 1 (es el turno del consumidor) puede entrar a la zona crítica a consumir, entonces se procede a buscar una sección crítica esto se hace por medio de una bandera llamada check la cual será igual a 1 una vez que encuentre una zona libre, si check es igual a 0 se mantiene en el ciclo incrementando el valor de j (espacio de la sección crítica), si no encuentra una zona crítica vacía y j es mayor que los espacios de la sección crítica entonces j se hace 0 y se incrementa k (sección crítica) y de nuevo si k es mayor que el número de secciones críticas se vuelve k a 0, si ya se encontró una sección crítica libre se procede a consumir de ella y escribirla en el archivo, al mismo momento libera el semáforo de los productores y ya cuando acaba de escribir el consumidor en el archivo libera la sección crítica que estaba ocupando.

Conclusiones:

Hernández Radilla José Ángel .-

En la práctica se aplicaron los conocimientos de aprendidos de hilos y sincronización de semáforos, uno de los retos fue no perder eficiencia en el programa debido a la limitación de que múltiples productores podían estar en zonas críticas peleando por escribir su valor, al final se optó por dejar escribir a un productor y prácticamente después de eso liberar al siguiente productor para ganar la eficiencia necesaria. La sincronizaciòn fue un ligero problema para resolverlo se utilizó semáforos cruzados para productores y consumidores, los semáforos del productor inician en 1 y los del consumidor en 0 y cada productor libera a su consumidor.

Arista Bojorges Victor Usiel .-

Para esta práctica a pesar de ya contar previamente con una parte de lo requiere esta solución costo implementarla debido a uso de memoria dinámica aplicada, la parte más complicada fue la implementación de estas zonas múltiples las cuales pueden tener tamaño distinto entre ellas. Otro problema fue corroborar si se estaban creando bien estas zonas ya que en algún punto hubo una falla en la iteración y se utilizó bastante tiempo al intentar corregirlo pensando que el problema estaba en esta memoria dinámica. Es importante remarcar la importancia de de la sincronización de los semáforos y como en algunos casos es necesario incluso proteger esta sincronización de zonas críticas mediante otra zona crítica.

Vallejo Serrano Ehecatzin

En esta práctica, mi mayor problema fue en pensar , cómo se le iba a hacer y además cuantos semáforos se requieren para lograr el objetivo. Ya que realmente no se me dio este tema. Mi aportación fue tratar de idear el mecanismo que me permitiera lanzar 5 hilos . teniendo solo cuatro zonas críticas, y al final pude hacerlo, mediante el uso de 2 semáforos , a los cuales denomine : “Grandes”, estos me permiten tener un control sobre las zonas críticas , ya que si uno de ellos lo inicializaba en 4 , esto me garantiza que al entrar cada uno de los hilos, estos decrementan al semáforo grande , permitiendome asegurar esas zonas, ya que si cuatro hilos entran, entonces ese semáforo, ya no permitiria un 5to bloqueo. El problema de esto , fue que mi idea fue realizar una función para cada uno de los consumidores y productores. Esto chocó con la idea de implementación de mis compañeros, ya que en el resultado final, solo se usó una función para

producir, y una función para consumir. A pesar de esto, todos llegamos a ideas similares de producción y de consumo, por lo que esta parte de mi intento de práctica , no aportó mucho , pero tampoco afectó al resultado final. Además de que mi compañero Usiel lo hizo dinámico, los cual , le da un plus a la práctica realizada.