Proyecto 2: Memoria Compartida: Productor - Consumidor

Lunes 10 de Mayo

I. Descripción

El propósito de este proyecto es experimentar con el uso de memoria compartida entre procesos *heavyweight*. Cualquier solución con *busy waiting* recibe un 0 de nota. Toda la programación debe realizarse en C sobre Linux.

II. Preliminares

En caso de no tener experiencia en este tema, es necesario investigar el uso de `mmap()` en Linux. Se debe investigar la sincronización de procesos *heavyweight* con semáforos. También, se debe saber como procesar argumentos de la línea de comandos de un programa ejecutado desde consola.

III. Relación Productor - Consumidor

El concepto de productor-consumidor es fundamental en Sistemas Operativos. Numerosos problemas pueden ser modelados y resueltos con variantes de esta relación.

Para este proyecto, los elementos producidos y consumidos serán mensajes a ser depositados en un *buffer* o buzón circular de tamaño finito. Dicho *buffer* es creado e inicializado por un programa independiente a los productores y consumidores, y será finalmente liberado por otro programa que cancela, de manera elegante, a todo el sistema de procesos creados en este proyecto.

Así, tenemos 4 tipos de procesos: creador, productores, consumidores y finalizador.

IV. Creador

Este programa gráfico será responsable de crear el *buffer* compartido por todos los procesos, de inicializar todas las variables de control asociadas al mismo (semáforos, banderas, contador de productores, contador de consumidores, etc.), y de monitorear de manera gráfica el estado de todo el sistema.

El nombre del *buffer*, el tamaño en entradas para mensajes, y cualquier otro parámetro que cada grupo considere conveniente serán recibidos de la línea de comando al ser ejecutado desde consola.

Usando `GTK` o algún equivalente, este programa mostrará constantemente el nombre y contenido del *buffer* (incluyendo indicadores del punto de entrada y el punto de salida del *buffer*), la cantidad de productores y consumidores, y una bitácora (*scrollable*) que registre todos los eventos del sistema. Inclusive, cuando el finalizador haya terminado al sistema, el creador seguirá vivo para poder examinar esta bitácora.

V. Productores

Esta es una familia de procesos, todos usando exactamente el mismo código, que se vinculan al *buffer* reservado por el creador. Con tiempos de espera aleatorios generarán mensajes que colocarán en el *buffer* (administrado circularmente).

Cada productor recibe como argumentos de la línea de comandos el nombre del *buffer* compartido y un parámetro que indique la media en segundos de los tiempos aleatorios, siguiendo una distribución exponencial, que deben esperar antes de agregar un nuevo mensaje en el *buffer*. Por supuesto, este acceso debe darse de manera sincronizada ya que el *buffer* es compartido por múltiples procesos. El usuario puede ejecutar cuantas veces lo desee este programa, creando un nuevo productor que compite por el *buffer*, aunque cada vez podría indicarse una media de espera diferente. Al crearse un productor, este incrementa el contador de productores vivos.

Estos procesos repiten un ciclo de espera aleatoria y fabricación de mensajes hasta que algún tipo de bandera global en memoria compartida indique que el sistema se debe suspender. En este caso, los productores terminan, decrementan el contador de productores vivos, y despliegan su identificación y algunas estadísticas básicas (número de mensajes producidos, acumulado de tiempos esperados, acumulado de tiempo que estuvo bloqueado por semáforos, etc.).

Si no hay espacio en el buffer, el productor queda suspendido hasta que se libere un espacio. No se puede usar *busy waiting*.

El formato específico del mensaje puede ser definido por cada grupo de trabajo, pero debe incluir al menos:

- Identificación del productor
- Fecha y hora de creación
- Llave aleatoria entre 0 y 4.

Cada vez que un mensaje logra ser puesto en el *buffer*, se debe desplegar un mensaje a la consola describiendo la acción realizada incluyendo el índice de la entrada donde se dejó el mensaje y la cantidad de productores y consumidores vivos al instante de este evento.

VI. Consumidores

Esta es una familia de procesos, todos usando exactamente el mismo código, que se vinculan al *buffer* reservado por el creador y que con tiempos de espera aleatorios consumen mensajes tomados del *buffer*.

Cada consumidor recibe como argumentos de la línea de comandos el nombre del *buffer* compartido y un parámetro

que indique la media en segundos de los tiempos aleatorios, siguiendo una distribución exponencial, que deben esperar antes de consumir el siguiente mensaje del *buffer* administrado circularmente. Por supuesto, este acceso debe darse de manera sincronizada ya que el *buffer* es compartido por múltiples procesos.

El usuario puede ejecutar cuantas veces lo desee este programa, creando un nuevo consumidor que compite por el *buffer*, aunque cada vez podría indicarse una media de espera diferente. Cuando se crea un consumidor, lo primero que éste hace es incrementar el contador de consumidores activos.

Estos procesos repiten un ciclo de espera aleatoria y consumo de mensajes hasta que lean un mensaje especial que indique que el sistema se deba suspender, o cuando al leer un mensaje este incluya una llave (número entero entre 0 y 4) que sea igual al *process id* o PID del consumidor módulo 5. En cualquiera de estos dos casos, el consumidor termina, decrementa el contador de consumidores activos, despliega su identificación y algunas estadísticas básicas (número de mensajes consumidos, acumulado de tiempos esperados, acumulado de tiempo que estuvo bloqueado por semáforos, etc.).

Si no hay mensajes en el buffer, el consumidor queda suspendido hasta que aparezca uno. No se puede usar *busy waiting*.

Cada vez que un mensaje logra ser leído del *buffer*, se debe desplegar un mensaje a la consola describiendo la acción realizada incluyendo el índice de la entrada de adonde se tomó el mensaje y la cantidad de productores y consumidores vivos al instante de este evento).

VII. Finalizador

Este programa se encarga de cancelar todo el sistema de procesos, enviando mensajes de finalización a cada consumidor vivo usando el buzón diseñado para este proyecto, e indicándole a los productores que cesen actividades con alguna bandera global en memoria compartida. Una vez que la cantidad de productores y consumidores llega a cero, el *buffer* compartido es liberado. El finalizador deberá dar mensajes y todas las estadísticas posibles de su gestión.

VIII. Generalidades

Los programas son corridos desde consola. El uso correcto de la línea de argumentos será muy tomado en cuenta para la evaluación de este proyecto.

Es indispensable que haya una correcta sincronización de procesos usando semáforos para arbitrar todos los recursos compartidos y para suspender procesos hasta que se den las condiciones que requieren para continuar.

Todos los componentes de este proyecto deben ser muy robustos y considerar cualquier combinación de eventos y cualquier valor en los parámetros. Si la ejecución no es posible, cada programa dará un mensaje apropiado y terminará de manera controlada.

IX. Requisitos Indispensables

La ausencia de uno solo de los siguientes requisitos vuelve al proyecto "no revisable" y recibe un 0 de calificación inmediata:

- La colaboración entre grupos se considera fraude académico.
- Todo el código debe estar escrito en C (no C++).
- El proyecto debe compilar y ejecutar en Linux. Todo debe estar **integrado**, explicaciones del tipo "*todo está bien pero no pudimos pegarlo*"[1] provocan la cancelación automática de la revisión.
- La presentación gráfica debe ser de mucha calidad.
- No debe dar "Segmentation Fault" bajo ninguna circunstancia.
- Cuando se indique, la única interacción válida con los programas es por medio de argumentos de línea de comando.
- Hacer la demostración en una máquina que levante Linux de manera real (puede ser dual), es decir no usar máquinas virtuales.

X. Fecha de Entrega

Demostraciones en clase el **Lunes 10 de Mayo del 2021**. También mandar por correo electrónico a `torresrojas.cursos.01@gmail.com` antes de la clase. Coloque todo lo necesario para compilar, documentar y ejecutar su proyecto en un directorio cuyo nombre esté formado por los apellidos de los miembros de cada grupo separados por guiones. Compacte este directorio en la forma de un `.tgz` llamado igual que el directorio (e.g., `torres-venegas-castro-smith.tgz`) y envíelo por correo. Identifique claramente su correo con el siguiente subject:

`[SOA] Proyecto Programado 2 - Apellido 1 - Apellido 2 - Apellido 3`

Buena Suerte.

[1] esto incluye los supuestos casos cuando alguien del grupo de trabajo no hizo su parte – el profesor no está interesado en sus problemas de organización.