Proyecto 3: Memoria Compartida: Productor - Consumidor

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Profesor: Ernesto Rivera Alvarado

I. Introducción

El propósito de este proyecto es experimentar con el uso de memoria compartida entre procesos *heavyweight*. 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 cómo 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 será responsable de crear el *buffer* compartido por todos los procesos y de inicializar todas las variables de control asociadas al mismo (semáforos, banderas, contador de productores, contador de consumidores, etc.). 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 comandos al ser ejecutado desde consola.

V. Productores

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 generarán mensajes que se colocarán en el *buffer* (administrado circularmente). Cada productor recibe como argumentos de la línea de comandos el nombre del *buffer* compartido y un número 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 *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.).

El formato específico del mensaje puede ser definido por cada grupo de trabajo, pero debe de incluir al menos: identificación del productor, fecha y hora de creación, y una "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 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 comando 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 entre 0 y 4) que

sea igual el 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 tiempos que estuvo bloqueado por semáforos, etc.).

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 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, 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. Requisitos indispensables

El incumplimiento de alguno de los siguientes requisitos vuelve, al proyecto "no revisable" y recibe como nota un cero.

- Todo el código debe de estar escrito C sobre Linux.
- No debe presentarse segmentation fault.

IX. Evaluación y completitud del proyecto

Todas las cualidades descritas en este documento deben integrarse en un solo proyecto. Por cada característica que falte se pierden 3^k puntos, donde k es el número de características que usted no incluyó. Esto aplica para $k > 1$. Se hace la aclaración de que es inadmisible la presentación de diferentes características en programas diferentes que no estén incluídas en un solo proyecto. La evaluación de este proyecto está sujeta a la presentación oral del mismo, en la que el estudiante debe mostrar un dominio completo del trabajo realizado a nivel de código.

X. Desarrollo del proyecto

El proyecto está pensado para desarrollarse individualmente, sin embargo, los estudiantes que deseen reforzar la habilidad de trabajo en equipo pueden entregar el proyecto en grupos de hasta tres personas. El profesor les hace la aclaración de que el trabajo con compañeros conlleva dificultades de coordinación, división de trabajo y sobre todo de "pegar o juntar ambas partes". En experiencias propias del profesor, se les comenta que en ocasiones el trabajo de juntar, acoplar y corregir partes desarrolladas por diferentes personas conlleva más tiempo y trabajo que la realización individual.

XI. Fecha de entrega

Las demostraciones se harán en semana 15 en clases. No se le revisará a los estudiantes que no se encuentren al inicio de la clase.

La carpeta comprimida `.zip` de su proyecto debe contener únicamente los archivos fuentes y su respectivo `Makefile` para compilar y generar el ejecutable del proyecto. El nombre de la carpeta comprimida serán los apellidos de los integrantes del grupo de trabajo (por ejemplo `rivera-alvarado.zip`) y este debe de ser subido al TecDigital el día de la revisión antes de las 6 am.