SISTEMAS OPERATIVOS

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 3: Concurrencia

Curso 2019/2020

Jorge Rodríguez Fraile, 100405951, Grupo 81, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, 100405834, Grupo 81, 100405834@alumnos.uc3m.es

Índice

Descripción de código

MAIN

Para empezar, comprobamos si el número de argumentos que se nos ha pasado es el correcto, en el caso de que esto no se cumpla lanzamos un error. Seguimos creando un descriptor del fichero que se nos da como argumento y leemos la primera fila del mismo (el número de operaciones totales que tengamos que realizar), que guardaremos en una variable llamada numVal. Una vez hecho esto calculamos el número de líneas totales del fichero mediante nuestra función calculo_lineas, si el resultado del total de líneas-1 es menos que nuestro numVal, lanzaremos un error.

Una vez hecho esto pasamos a la creación de los threads. La división de las operaciones entre productores será equitativa en el caso de que el número de productores y de operaciones sean divisible, en el caso contrario, el último thread que entre tendrá el número equitativo de operaciones más el resto. Para el número de operaciones calcularemos floor(numVal/operaciones) y el del último thread será numVal -(productores*operaciones). Paralelamente, también crearemos el buffer circular (cola), los mutex ring, des y las variables condición lleno y vacío. El uso de cada uno de ellos será explicado en su apartado correspondiente.

Ya establecidos el número de operaciones que hará cada hilo, debemos crearlos y pasarles los parámetros, para esto hemos creado la estructura args, que tendrá por un lado el número de operaciones que realizará el hilo y por otro el id de inicio. Este id de inicio representa el “dominio” de cada hilo, es decir un hilo tendrá acceso al dominio {id_inicio, id_inicio+número de operaciones} para evitar así condiciones de carrera en el fichero. Como se resuelve la concurrencia entre estos se realizará en la función producir. En cuanto al consumidor, simplemente le pasamos numVal, para que sepa cuantas operaciones debe realizar.

Al terminar la codificación de creación de productores y consumidor, los esperamos a todos con un join, y al terminar liberamos todos los recursos utilizados y retornamos el total.

PRODUCIR

Para esta función empezamos asociando la estructura arg pasada a una variable p, lo siguiente que haremos es bloquear el mutex des, es decir el del descriptor de fichero que es compartido entre hilos. El primer paso después del bloqueo será abrir el fichero, lo que llevará el puntero al principio del mismo, y buscamos el comienzo de su dominio recorriendo el fichero y llegando a p->id_incio. Al llegar guardamos el puntero en la variable current, así lo podemos recuperar cuando sea pertinente. Guardado la posición de inicio desbloqueamos el mutex del descriptor y continuamos, esta vez para guardar los datos en el buffer circular.

Para almacenar los datos en el buffer creamos un bucle for que vaya indicando el número de operaciones restantes, dentro de este, empezamos volviendo a bloquear el descriptor y igualando la variable current a una nueva descriptorP, ahora desbloqueamos des y bloqueamos el mutex ring (buffer circular) y creamos nuestra estructura element que guardará los datos en el buffer. Por último, mientras el buffer esté lleno, esperamos a la variable condición lleno, en otro caso simplemente hacemos un put en cola con nuestra estructura elem, y levantamos la variable condicional vacío en el caso de que el consumidor estuviera esperando por ella y hacemos unlock de ring.

CONSUMIR

Para consumir, simplemente empezamos creando un struct element auxiliar llamado data y hacemos un bucle for para saber el número de valores que tenemos que leer. Al iniciar el bucle hacemos lock de ring, y mientras la cola esté vacía, esperamos a que se levante la condición vacío, al pasar estas condiciones cargamos en data el queue_put y realizamos las operaciones mediante un switch de data->type, al terminar el switch levantamos la condición lleno y hacemos unlock de ring.

Pruebas realizadas

Buffer Circular

Probamos a inicializar una cola y a insertar un valor con queue_put, la cola se crea sin ningún problema y para ver si queue_put ha funcionado matamos dos pájaros de un tiro y usamos queue_get, el resultado es correcto, hacemos un destroy para liberar recursos y proseguimos con las pruebas.

Seguimos comprobando con un buffer vacío, y esperamos que queue_empty sea 1, el resultado es el correcto, por lo que pasamos a la siguiente prueba.

Ahora comprobamos con un buffer lleno, le damos un tamaño de 10, y lo llenamos, queue_full vale 1 por lo que todas las funcionalidades del buffer circular funcionan como se espera.

calculator

Introducir un fichero que no existe, probamos un archivo que no esté en el directorio y debe dar error de archivo no encontrado. Tras la ejecución recibimos el error “No such file or directory” que es el esperado.

Introducir un fichero con un número inválido de operaciones, primero probamos un archivo con 20 operaciones descritas y al principio indique que hay 0, otro con -2 y otro con 30, esperamos que nos indique que el número de operaciones no es válido. Ejecutamos las 3 pruebas y recibimos “Error: Se indica un número de operaciones incorrecto” que es el esperado para estos casos.

Introducir un fichero en el que las operaciones no tengan el formato correcto, id tipo tiempo, se introduce con una de las operaciones 121 34 y esperamos que nos indique que no es correcta esa línea. Probamos con el correspondiente fichero y en aquellas líneas que no lo cumplen devuelve el error “Valor no válido”.

Introducir argumentos de más, se prueba con calculator input_file.txt 20 600 23 que nos debe indicar que hay demasiados argumentos. Al ejecutar el mandato anterior recibimos el error “Número de argumentos inválido” que era el objetivo.

Introducir como argumento un número menor o igual a 0 de productores, esperamos que nos de error. Tras la prueba recibimos el error “Error: Numero invalido de productores.” que es el indicado.

Introducir como argumento un número menor o igual a 0 como tamaño del buffer, al ser un valor inválido esperamos un error. Se ejecuta la prueba y salta el error “Error: Tamaño inválido.” que es el correspondiente.

Introducir un mandato válido del que conocemos el resultado esperado, con el mandato ./calculator input_file.txt 200 10 y resultado 145729. Se ejecuta el mandato indicado y recibimos “Total: 145729 €.” que es el resultado esperado para nuestro fichero.

Conclusiones

En este proyecto hemos tenido mucha más libertad a la hora de pensar y crear código, por lo que la mayoría del tiempo que se ha invertido en él ha sido en tener una idea clara de cómo se iba a repartir las operaciones y el control de la concurrencia. Aunque el resultado final cuente con ‘pocas’ líneas, ha sido difícil organizar el código y saber que parte que hacía cada cosa. Por lo demás el proyecto nos ha ayudado a conocer mejor el uso de los mutex y las variables condición y a saber manejar de manera correcta los datos compartidos entre los hilos.