Fundamentos de los Sistemas Operativos
Ficha de entrega de práctica

**: campo obligatorio*
***IMPORTANTE:** esta ficha no debe superar las DOS PÁGINAS de extensión*

Grupo de prácticas*: 1 - 41
Miembro 1: Alejandro Vialard Santana
Miembro 2:

Número de la práctica*: 4 | **Fecha de entrega*:** 13/05/2022

Descripción del trabajo realizado*
Primera parte: en esta parte, se pedía realizar las funciones de sincronización de los hilos de un búfer finito para el acceso concurrente usando pthread(), el cual nos entregaba el profesor sin ninguna herramienta para que esto funcionara correctamente.

Para poder hacer que funcionara, se ha creado un cerrojo (mutex) y dos variables condición para los hilos Productor y Consumidor, que nos ayudará a hacer que esperen o avisen a los otros.
En el método Productor(), se ha añadido un retardo aleatorio para simular un tiempo de entrada de hilos diferentes, luego hacemos que tome el cerrojo si le toca al productor. Hacemos un bucle while para que el hilo productor espere, que será cuando el búfer esté lleno. Después, para poder cumplir que se puedan insertar más de un elemento cada uno, ponemos un bucle que vaya hasta 10, y comprobamos si el bufer no está lleno, si se cumple se inserta el item, y si no sale con un break. Por último, se avisan a todos los consumidores y se desbloquea el cerrojo.

Para el hilo consumidor es similar, lo único que cambia es que este espera cuando el bufer está vacío, si no extrae los item, pudiendo extraer varios el mismo consumidor. Luego, este avisa a los productores y les cede el cerrojo.

Además, en la función main() he utilizado el **srand()** para generar semillas diferentes.

***(Añadido de la corrección)**
Al iniciar los hilos en test_hilos(), he hecho que se lancen de manera pseudoaletoria de la siguiente forma: Creé una variable aleatoria y dos contadores iniciados al valor de productores y consumidores introducidos por consola. Realizo un bucle while que se ejecutará mientras ambos contadores no sean 0. Dentro de este le otorgo un valor random de 0 a 1 a la variable, que cada vez que se ejecute el bucle cambiará. Entonces, si la variable da 0, se ejecuta un hilo productor y se reduce el contador; y si da 1, se ejecuta un consumidor y se reduce su contador. Llegará a un punto en el que uno de los contadores no sea 0, ya que se debe de llenar cada vector de hilos con el tamaño pasado por consola. Por tanto, para solucionar esto, si uno de los contadores es 0 es que un vector se llenó, por lo que se saldrá del while, y entonces hago condiciones, si el contador de consumidor es 0 (se llenó su vector) y el de productor no es 0, se hace un bucle hasta llenar el vector de productores; si es al revés, se hará un bucle hasta llenar el de consumidores, así solucionamos que alguno de los vectores no se llene. (Está implementando tanto en el bufer sin herramientas como con el de herramientas).

Segunda parte: en esta he decidido hacer el **RETO** programando el problema de misioneros y caníbales con concurrencia.
Para realizarlo, he seguido usar la estructura del bufer finito para realizar la práctica. En el archivo “Bote.c” hay funciones como las de “boteInicia()” que inicia el bufer, “boteSubir()” que permite entrar un item al bufer, “boteBajar” que extrae un elemento del bufer, “botePendienteViajar()” que dice cuantos elementos hay dentro del bote y “boteCapacidad()” que da la capacidad del bufer, que en este caso siempre será 3.

Grupo de prácticas*: 1 - 41
Miembro 1: Alejandro Vialard Santana
Miembro 2:

Número de la práctica*: 4	**Fecha de entrega*:** 13/05/2022

Por otra parte, en el archivo "Misios_y_Canibales.c", he programado las funciones principales para realizar el programa, siguiendo la estructura del de productores y consumidores. Primero, inicializo las variables condición junto al cerrojo, además de otras variables como: misionerosABordo, canibalesABordo (que indican quienes están en el bote), misioneros, caníbales (que los uso para poder hacer condiciones de cuando quedan menos de 3 personas) y listoZarpar que actúa como un boolean que permite navegar a los tripulantes. Luego, en cuanto a los métodos:
-retardoAleatorio(): para simular tiempos diferentes de entrada de los hilos.
-checkError(): para ver si hay errores.
-llegaMisionero(): este método actúa como uno de Productor – Consumidor. Primeramente, le añadimos un retardo aleatorio y tomamos el cerrojo. Luego, el hilo esperará si se cumple que el barco está zarpando (listoZarpar = 1), que haya un misionero y un caníbal ya dentros o que esté lleno el bote. Luego hacemos unas comprobaciones para saber si quedan tres o menos personas para subir al bote, lo que generará errores si es menor de tres, ya que no se puede llenar el bote, o si queda dos misioneros y un caníbal, imprimiendo por pantalla el error y llamando a todos los hilos para que finalicen. Después de esto, se irá insertando un misionero siempre y cuando el bote no esté lleno y no esté listo para zarpar, y dentro de esta misma condición, ponemos otra que si el bote está lleno, pone listoZarpar a 1 y sale mensaje, esto lo hice debido a que tenía problemas cuando se bajaban las personas, ya que se quedaban esperando siempre y necesitaba que se bajaran todos, por ello uso este tipo de variables, al igual que la siguiente condición que se ve, que es un bucle while que pone en espera al hilo misionero si no está el bote listo para zarpar, ya que como dije antes, no se bajaban todos del bote, solo uno y los demás quedaban en espera, por lo que hice un ciclo así (No espera por el primer while -> se inserta -> espera es mismo hilo en el bote haciendo que otros entren -> se puede extraer luego), dejando así la posibilidad de que él espere en el bote mientras otros hilos entran. Para añadir a esto, luego hay otra condición que es que cuando el barco esté listo para zarpar, se supone que zarpa y se deben de bajar todos para que suban los demás, por tanto se baja el misionero, se reduce el numero de misioneros a bordo, llama a todos los hilos que estaban esperando (los cuales son los que estaban en el bote para que ellos también puedan bajarse del bote) y comprueba si el bote está vacío, ya que si es así pone listoZarpar a 0 e indica que se vació, llamando a todos los hilos que estaban esperando para subir al bote. Por último, se desbloquea el cerrojo y se termina el hilo.
-llegaCanibal(): tiene la misma función que el anterior, pero cambia el primer bucle de espera, haciendo que el caníbal espere si el barco está listo para zarpar, si está lleno o si ya hay dos misioneros en el bote. Lo demás es igual, pero se inserta y se extraen los caníbales en vez de los misioneros.
-esperaLoteHilos(): método que espera a que los hilos lanzados terminen.

***(Añadido de la corrección)**
-testHilos(): método que lanza los hilos y ve si hay errores. Aquí también hice que se lanzaran de manera pseudoaleatoria como en el del búfer finito, pero esta vez con las variables de misioneros y caníbales correspondientes y sus contadores (tanto en el Misios_y_Canibales como en el Misios_y_Canibales_Incorrecto).

-leeArgumentos(): comprueba los argumentos pasados por la línea de comandos, atribuyendo a variables como misioneros y caníbales sus valores para comprobación.
-main(): función principal que inicia el programa, añadiendo un **srand()** para poner semilla aleatoria y que no siempre sea lo mismo.

También se implementa el mismo ejercicio, pero sin las herramientas de mutex y variables condición.

*Aclaración: la vista del programa se deja en el archivo PDF "PRUEBAS DE LA PRÁCTICA 4".

Grupo de prácticas*: 1 - 41
Miembro 1: Alejandro Vialard Santana
Miembro 2:

Número de la práctica*: 4	**Fecha de entrega*:** 13/05/2022

Horas de trabajo invertidas* Miembro 1: 13 horas aprox

Cómo probar el trabajo*

Primera Parte: para probar esta parte, hay dos carpetas (bufer_completo y bufer_sin_implementar), en el que está el ejercicio del bufer con las herramientas de sincronización y en el otro el que no las tiene, respectivamente. Para acceder a la carpeta utilice: "cd directorio/ practica4-AVS /bufer_completo o cd directorio/practica4-AVS/bufer_sin_implementar). Una vez dentro verá buffer_circular.c con las operaciones del bufer, su respectivo .h y test_hilos.c (y posiblemente el .o pero esto se consigue compilándolo). Si quiere compilar el ejercicio, tanto para el del bufer implementado como el que no, compilar con: "c99 -o "archivo-ejecutable" test_hilos.c buffer_circular.c -lpthread.
Y luego para ejecutarlo, escriba "./archivo-ejectuable capacidad-del-bufer nº-productores nº-consumidores".

Segunda Parte: en la carpeta"misioneros_y_canibales", en la que están los archivos "Bote.c" como bufer, su respectivo .h, el archivo "Misios_y_Canibales.c" como el programa con herramientas de sincronización y "Misios_y_Canibales_Incorrecto.c" que no las tiene. Accede al directorio mediante: "cd directorio/misioneros_y_caniables". Una vez allí, se debe compilar cualquiera de los dos programas de esta manera: "gcc Misios_y_Canibales.c Bote.c -o archivoejecutable -std=c99 -lpthread". Para ejecutarlo debe de escribir: "./archivoejecutable nº misioneros nº caníbales".

Incidencias

En el archivo "Misios_y_Canibales.c", al ejecutar ./archivo 5 1, si entra un caníbal con un misionero de primera se me queda esperando siempre, cosa que no debería de ocurrir pero esto ocurre debido a la inanición que presenta, ya que yo sólo solucioné cuando quedan los últimos 3 esperando, pero no si ocurre al principio. Por tanto, debido a esto, se lo demuestro en las Pruebas.pdf.

Comentarios