Universidad de Costa Rica

Escuela de Ciencias de la Computación e informática

CI0117- Programación Paralela y Concurrente

Documentación de métodos de sincronización

“Pthreadmon"

Estudiantes:

Gianfranco Bagnarello Hernández B70866

Katherine González Arias B22867

Profesor: Jose Andrés Mena Arias

II Semestre, octubre 2020

Métodos de sincronización utilizados en este proyecto

El objetivo de este trabajo se centró en la realización de un programa escrito en el lenguaje de programación C, en el cual se implementa el modelo de programación paralela y concurrente, donde se llevó a cabo un simulador de batallas Pokémon. En esta simulación se enfrentaron dos jugadores, cada uno con tres hilos de tipo Pokémon.

Para lograr la implementación de este proyecto, se utilizó la librería pthread.h, con la cual se aprovecha el uso de hilos para lograr la eficiencia en la separación de asuntos. Se dividen las responsabilidades entre los diferentes hilos, o procesos, y se recurre a la técnica de divide y vencerás con el modelo de vista controlador (MVC).

Para lograr la concurrencia compartida se recurre a la utilización de mutex, lo que permite una sincronización primitiva que pone una restricción alrededor de una sección crítica para evitar data races, lo que garantiza la atomicidad, asegurándose de que solo un hilo acceda a la sección crítica a la vez.

Es así como en el programa se crearon dos mutex, en vez de un arreglo de mutex como se había propuesto originalmente. Hay un mutex para la función que ejecuta la lista 1 de Pokémon y otro mutex que ejecuta la lista 2 de Pokémon, de esta manera se logra que uno solo thread por lista pueda acceder a la función fight(). Es decir, solo un Pokémon por lista puede acceder a pelear, previniendo que de esta manera todos quieran entran a pelear al mismo tiempo. Además, con la creación de un mutex por cada lista

de Pokémon, se logra que los Pokémon entren a pelear en orden, primero pasa el primero Pokémon elegido, luego el segundo y luego el tercero.

Además, en este programa se implementan el uso de barrier, ya que cuando varios subprocesos trabajan juntos, es posible que sea necesario que los subprocesos se esperen entre sí en un evento o punto determinado del programa antes de continuar. Por lo cual, utilizamos un Barrier dentro método fight(), antes de implementar el mutex, y de esta manera se permite que cada una de las dos listas de Pokémon esperen a que cada una tenga tres Pokémon antes que empezar a pelear.

Problemas de sincronización detectados en la implementación.

Durante la programación de este programa se presentó starvation durante la implementación del arreglo de mutex, uno para cada thread Pokémon, se apreció que durante la ejecución un hilo agarró un mutex y les impidió a los otros hilos del arreglo entran. De esta manera, ese thread con el mutex entró a atacar a los hilos Pokémon del jugador rival. En este escenario ese único thread logró matar a todos lo Pokémon de la otra lista y el programa quedó esperando que los otros dos hilos pudieran acceder al mutex, pero al no haber más pelea, porque un solo hilo derrotó a todos los contrincantes se generó starvation y que no hubiera más pelea. Por esta razón es que se decidió que el arreglo de mutex no era la mejor solución y se cambió por un único mutex y dos listas de Pokémon, una por cada jugador, de esta manera cada Pokémon de cada lista pasaba en orden a atacar, bloqueaba el mutex, atacaba y lo liberaba para que el siguiente hilo lo pudiese bloquear y pasar en orden a atacar.

También, en este programa se originó un escenario con el problema de deadlock, ya que se presentó un error en donde un hilo procedió a bloquear un mutex dos veces, es así como mientras que los otros hilos estaban esperando a que se desbloqueara el mutex, el mutex que lo tenía caía en un ciclo ¨*loop*¨ que no lograba desbloquear el mutex, porque ese segundo bloqueo caía sobre el mismo mutex, lo cual no permitía que hubiese esa liberación, por lo cual se generaba deadlock, al igual que con el problema de starvation, este deadlock se resolvió al no implementar el arreglo de mutex planteado en un inicio, y de esta manera, al implementar uno mutex por cada una de las dos listas fue que se dejó de presentar este comportamiento.

Anexo:

Para la implementación de este proyecto se hizo uso de la librería SDL, la cual nos proveyó recursos que nos permitiera explorar más los herramientas que posee C, con la cual se agregó audio a la interfaz. De esta manera, a la hora de inicializar la ejecución de la simulación de la batalla entre hilos Pthreadmon se empezará a reproducir en archivo de audio añadido.

Para consultar las fuentes de los archivos de audio se puede consultar las referencias bibliográficas.

Referencias bibliográficas:

"Playing a WAV File Using SDL2 - Gigi Labs", Gigi Labs, 2020. [Online]. Available: https://gigi.nullneuron.net/gigilabs/playing-a-wav-file-using-sdl2/. [Accessed: 21- Oct- 2020]

"How do I install SDL on ubuntu?", Gist, 2020. [Online]. Available: https://gist.github.com/BoredBored/3187339a99f7786c25075d4d9c80fad5. [Accessed: 21- Oct- 2020]

[Online]. Available: https://www.youtube.com/watch?v=EFVIN01jUUU. [Accessed: 21- Oct- 2020]

[Online]. Available: https://www.youtube.com/watch?v=0j4sfjIXTA0. [Accessed: 22- Oct- 2020]