Mini TP - Threads y Semáforos

Profesores: Andres Rojas Paredes y Leandro Dikenstein Hidalgo

Alumno: Federico Grande y Nicole Martínez

Materia: Sistemas Operativos y Redes Com 2

Fecha de entrega: 7 de Abril de 2025

La actividad consistió en crear un programa que modele una reunión de estudiantes y profesores en la casa de Manucho para comer un asado en festejo por haber aprobado Sistemas Operativos y Redes. Y como Manucho había ganado la lotería, también contrató un servicio de catering con mozos y un asador para que se encarguen de cocinar y servir. Para poder realizar este trabajo, tomamos la decisión de dividir la reunión en tres momentos principales y a cada momento asignarle un conjunto de threads para realizar una serie de tareas específicas.

El primer momento consiste en que todos los invitados se sienten en la mesa y que Manucho se siente último. Para esto se definió una función que imprimía un mensaje con el número del invitado que se sentó, siendo este número el número correspondiente al thread que llamó a dicha función. Tomamos la decisión de representar a cada invitado con un threads que llamaran a la función “sentarse()”. Dentro de sentarse hay un contador que aumenta cada vez que se ejecuta, y una vez que el contador es igual a la cantidad de invitados se imprime un mensaje avisando que Manucho ya se sentó. De esta forma nos aseguramos de que los invitados se sienten en orden aleatorio pero que Manucho sea el último en sentarse. Cada hilo que ejecute “sentarse()” está accediendo y modificando una variable global, que es el contador de cuantos invitados se sentaron; por lo que para evitar que dos hilos modifiquen al mismo tiempo el contador utilizamos un semáforo que nombramos “sem_sentarse”. El semáforo se inicializa en 1 y antes de modificar el contador se ejecuta un “wait(s)”. Cuando se imprime el mensaje de quien fue el “invitado” que se sentó y se compara si el contador es o no igual a la cantidad de invitados, se ejecuta un “signal(s)” liberando así el semáforo para que otro hilo pueda ejecutar la función.

El segundo momento consiste en que los mozos sirvan los platos. Este proceso debe ejecutarse luego de que manucho se sienta, por lo que implementamos un semáforo llamado “sem_servir” que se inicializa en 0 y una vez que en “sentarse()” Manucho se sienta, se realiza un signal(sem_servir) para que los hilos que representan a los mozos puedan ejecutarse. Al igual que con los invitados, los mozos son representados con threads. Cada vez que se sirve un plato, se imprime por pantalla el mozo (representado por el id del thread que ejecuta la función) que sirvio un plato y cuantos platos faltan por servir. Para llevar este conteo, utilizamos un contador que inicia con la cantidad de platos que deben servirse en total y disminuye cada vez que se ejecuta la función. Al igual que en el primer proceso, cada hilo accede a una variable global y la modifica por lo que utilizando el semáforo sem_servir se controla el acceso de los hilos al contador, repitiendo el mismo patrón que en “sentarse()”. Como la cantidad de mozos es menor a la cantidad de invitados, el método “servirComida()” esta implementado con un “while” que finaliza cuando la cantidad de platos por servir es 0.

Finalmente, el último momento corresponde a cuando los invitados empiezan y terminan de comer y Manucho lanza su pregunta mundialista. Para ello ocupamos la misma metodología, cada invitado y manucho son representados por un thread que ejecuta la función “comer()” que desencadena toda la situación final.

Como la consigna decía que los invitados y Manucho pueden empezar a comer en cualquier momento, decidimos manejar aquello con semáforos. Contamos con un semáforo que se llama “sem_comer” que se incializa en 0. Cuando los mozos sirven un plato, también activan este semáforo con un signal(sem_comer), que habilita que el hilo que se encarga de ejecutar la función “comer()” pueda hacerlo. Dentro de esta función se llama a la función “terminoComer(id)” que imprime un mensaje con el número del invitado que terminó de comer. Si ese invitado es Manucho, se llama a la función “lanzar_pregunta_mundialista()”, que activa un semáforo que permite que un invitado le responda. Una vez lanzada la respuesta, se ejecuta la función “enojarse()” que corresponde a cuando Manucho se enoja por la respuesta recibida y esta función a su vez habilita el semáforo que controla que los invitados puedan levantarse.

Para realizar esta actividad utilizamos un solo proceso, dentro del cual se ejecutan los hilos con distintas funciones; y 6 semáforos para sincronizar la ejecución de los hilos

Si bien la solución que ofrecimos al problema funciona correctamente y no ocurre inanición, si es posible que ocurra cuando se ejecutan los hilos que llaman a la función comer(). Esto se debe a que “comer()” espera a que la función “servirComida()” active el semáforo “sem_comer” que regula la entrada de cada thread al código crítico de “comer()”. Si ocurre un error en los hilos de ejecución de “servirComida()”, los hilos que ejecutan “comer()” esperarán indefinidamente que se libere su semáforo. Este error puede ocurrir si por ejemplo, en la función “sentarse()” nunca se realiza el signal(sem_servir) que habilita que los mozos sirvan la comida. Del mismo modo, si nunca se llama a comer() o Manucho nunca termina de comer, ningún invitado podrá levantarse de la mesa ya que para eso es necesario que se enoje manucho, entonces una función nunca se ejecutará.