Instituto Tecnológico y de Estudios Superiores de Monterrey

Jesús Ramírez Delgado

A01274723

Implementacion de Métodos Computacionales

Programación 12

Problema:

1. Un grupo de estudiantes está desarrollando una actividad integradoras de la materia TC2037. Los estudiantes solo pueden programar mientras comen pizza. Cada alumno ejecuta el siguiente bucle:

```
while(true){
recoger un trozo de pizza;
estudiar mientras come la pizza;
}
```

Si un estudiante descubre que la pizza se ha acabado, el estudiante se va a dormir hasta que llegue otra pizza. El primer estudiante que descubre que el grupo no tiene pizza llama a la pizzerías “All Night Long” para pedir otra pizza antes de irse a dormir. Cada pizza tiene 8 rebanadas. Escribe un código para sincronizar los hilos de los estudiantes y el hilo de entrega de pizza. Tu solución debe evitar interbloqueos y la atención al cliente de la pizzería (es decir, reactivar el hilo de la pizzería) exactamente una vez cada vez que se agota una pizza. Ningún pedazo de pizza puede ser consumido por más de un estudiante.

Para este problema implemente dos librerías que facilitaron la implementación de la solución.

- **<mutex>:** Esta librería es para exclusión mutua, que normalmente se utiliza para controlar el acceso a seccines del código usando varios hilos. En este caso se creó una variable **mtx** para controlar esto.

- **<condition_variable>:** Muy de la mano de mute se implementa para la comunicación entre hilos, en este caso en el código igual se creó una variable para tener control de esto y se implementó en el código para notificar a varios hilos dentro del código.
- **<chrono>:** Utilizada para medir y manipular tiempo, se implemento en la función **comerPizza** para simular tiempo de comer pizza

```
condition_variable cv;  // Variable de condición para esperar a que llegue una nueva pizza
bool pizzaDisp = true;  // Variable booleana que indica si hay pizza
mutex mtx;  // Mutex para asegurar exclusión mutua al acceder a la pizza
```

El siguiente paso fue crear las funciones para este código, el cual concistio de 2 que se comportan para que los alumnos (hilos) coman pizza y otra para que en dado caso de que la variable anterior **pizzaDisp** sea false, se pida una nueva pizza.

```
void comerPizza(int estudiante) { // Funcion accion de comer pizza
    wwhile (true) {
        unique_lock<mutex> lock(mtx);

        // Verificar si hay pizza disponible
        if (pizzaDisp) {
            cout << "Estudiante " << estudiante << " Agarra una reabanada de pizza." << endl;

            // Simular tiempo de estudio mientras se come la pizza
            this_thread::sleep_for(chrono::seconds(2));

            cout << "Estudiante " << estudiante << " termina de comer." << endl;
        } else {
            // No hay pizza disponible, el estudiante se va a dormir
            cout << "El estudiante " << estudiante << " se va a dormir." << endl;
            cv.wait(lock);  // Esperar a que llegue una nueva pizza
            continue;  // Rectificar si hay pizza nueva
        }

        // La pizza se ha terminado, el estudiante llama a la pizzería
        if (estudiante == 1) {
            // Pide otra pizza por Rappi
            cout << "Estudiante " << estudiante << " pide otra pizzeria por Rappi" << endl;
            pizzaDisp = false;  // Marcar que no hay pizza disponible
            cv.notify_one();  // Notificar que se necesita pedir otra pizza
        }
    }
}
```

La función **comerPizza** que representa la acción de comer pizza. Esta recibe un parámetro llamado estudiante que será el número de hilos. Por medio de un ciclo while, primero con ayuda de mutex bloquea la variable **mtx** para tener exclusión mutua y acceder a las variables **pizzaDisp** y **cv**. En este caso primero verifica si hay pizza, si es así agarra una rebanada y simula el tiempo de que se come la pizza, en dado caso de que no haya pizza el estudiante se va a dormir y espera que haya una nueva pizza, finalmente si la pizza se termina, piden otra pizza nueva.

```
void nuevaPizza() { // Funcion accion entrega pizza
    while (true) {
        unique_lock<mutex> lock(mtx);

        // Esperar a que se agote la pizza
        cv.wait(lock, [] { return !pizzaDisp; });

        // Entregar una nueva pizza (8 rebanadas)
        cout << "Llego el repartidor de Rappi" << endl;
        pizzaDisponible = true;  // Hay pizza disponuble si la condicon es true
        cv.notify_all();  // Para notificar que hay pizza
    }
}
```

En la función **nuevaPizza** se representa la acción de pedir o ordenar una nueva pizza. Por medio de un ciclo while mientras sea verdadero (true) volverá a pedir una nueva pizza. Al inicio por medio del objeto unique_lock<mutex> se asegura el bloqueo de exclusivo de las variables **pizzaDisp** y **cv.** Esto permite que se bloquee la ejecución hasta que se cumpla la condición de que no haya pizza.

Finalmente, en la función main se implementan las dos funciones previas y por mediom de ciclos for se crean los hilos y se ejecuta la función **comerPizza** y después con **join** espera a que los estudiantes terminen su pizza.

```
int main() {
    thread estudiantes[3]; // Variable para crear hilos (estudiantes)
    thread pizzeria(nuevaPizza); // Variable hilo de pizzeria

    // Crear hilos (estudiantes)
    for (int i = 0; i < 3; i++) {
        estudiantes[i] = thread(comerPizza, i + 1);
    }

    // Esperar a que los hilos/estudiantes terminen
    for (int i = 0; i < 3; i++) {
        estudiantes[i].join();
    }

    // Esperar a que el hilo de la pizzería termine
    pizzeria.join();

    return 0;
}

```