Trabajo Práctico 2

Paradigmas de la Programación

Pereyra Agustín[1], **Pizzaro Federico**[2]
Universidad de San Andrés, Provincia de Buenos Aires, Argentina
[1]*apereyra@udesa.edu.ar*
[2]*fpizarro@udesa.edu.ar*

11 de junio de 2025

Este trabajo tiene el objetivo de afianzar los conceptos de *Multithreading* o programación multi-hilos, con sincronización y uso de datos compartidos como también serialización, uso de clases y structs, containers y streams en C++ a partir de diferentes ejercicios prácticos.

Aclaración: *Para cada ejercicio se diseñó un archivo Makefile para automatizar la compilación y ejecución con g++ de cada uno de los ejercicios, utilizando las flags −Wall −Wextra, −Wpedantic. En cada sección se detallan los comandos correspondientes para cada uno*

1. Pokedex

2. Control de Aeronave en Hangar Automatizado

2.1. Compilación y Ejecución

Posicionandose dentro de la carpeta exercise2 :

I. make main: compilar
II. make run: compilar y ejecutar
III. make run−valgrind: compilar y ejecutar con Valgrind
IV. make clean: eliminar ejecutables

3. Sistema de Monitoreo y Procesamiento de Robots Autónomos

Para la resolución de esta parte, se hizo uso de una sola estructura llamada Task, la cuál tiene las propiedades de las tareas emitidas por los Sensores y procesadas por los Robots, ambos implementados como hilos (threads). Además está casi todo el código dentro del mismo archivo source, main.cpp, y lo único fuera de él es tanto la implementación como la definición de la estructura.

El struct Task solo contiene el ID del mismo, el ID del sensor que lo generó y una descripción del mismo, la cual se elige aleatoriamente de un array de descripciones genéricas. Además tiene solo un constructor que tiene como función adicional, simular el delay de 175ms de la creación de una tarea.

Cuando el programa comienza, se definen tanto las constantes como las variables globales, entre ellas podemos destacar a los 2 objetos std :: mutex, encargados de sincronizar tanto el uso de la consola como de la std :: queue de tasks, al objeto std :: atomic que lleva la cuenta de los sensores activos y a la variable y a la variable condicional std :: condition \ _variable usada para sincronizar a los sensores con los robots.

Luego se inician los threads definidos como std :: jthreads para que todos sean ejecutados por completo y no puedan ser interrumpidos. Luego el funcionamiento se rige por la siguiente estructura para cada hilo:

3.1. Sensores

I. Por cada tarea a generar
    I. Se bloquea el std :: mutex de la queue para crear la tarea, asignando los atributos y agregándola a la cola.
    II. Se bloque la consola y se hace el *log* respectivo.
    III. Se notifica a los robots (que están en espera) que hay una tarea disponible mediante la variable condicional.
II. Lockeando el mutex destinado a los sensores, se actualiza el contador atómico y se verifica si es el último sensor. Si es el caso se notifica a los robots.

3.2. Robots

I. Mientras que hayan sensores activos o tareas disponibles

    I. Espera hasta ser notificado por los sensores que hay una tarea nueva disponible, o que terminó el último sensor
    II. Se quita la tarea de la cola y se la procesa (esperando el respectivo acceso mediante el mutex)
    III. Se simula el delay del procesamiento de la tarea de 250ms
    IV. Se hace el *log* del procesamiento lockeando primero la consola.

II. Se loggea la finalización del trabajo del Robot

3.3. Compilación y Ejecución

Posicionandose dentro de la carpeta exercise3 :

I. make main: compilar
II. make run: compilar y ejecutar
III. run−valgrind: compilar y ejecutar con Valgrind
IV. make clean: elimina ejectuables

Aprendizaje