Nombre: Diego Marco Beisty
NIP: 755232

Decisiones de diseño:

La cola está inicialmente vacía, los procesos lectores iteran 6 veces accediendo a la cola para desencolar un elemento en cada acceso y los procesos escritores iteran 8 veces accediendo a la cola para encolar un elemento en cada acceso.
La sincronización se ha realizado mediante awaits.
Los procesos escritores esperan a que la cola no esté llena y encolan un elemento.

```
<await cola.length < 10
      encolar(cola, elemento)
>
```

Los procesos lectores esperan a que la cola no esté vacía y desencolan un elemento.

```
<await cola.length > 0
      elemento = desencolar(cola)
>
```

La implementación se ha realizado mediante la técnica de paso del testigo con semáforos como se indica en el enunciado de la práctica.

Principales dificultades de la práctica:

Pequeños problemas en la Implementación del constructor y destructor de ConcurrentBoundedQueue, que conseguí resolver.

Comportamiento observado:

Cada thread intenta entrar a la sección crítica (cola) en exclusión mutua obteniendo el semáforo mutex para encolar o desencolar un elemento. Si un lector obtiene el mutex pero la cola está vacía, se bloquea sobre el semáforo b_hay_dato. Si un escritor obtiene el mutex pero la cola está llena, se bloquea sobre el semáforo b_hay_hueco. En caso contrario accede a la sección crítica (cola).
Cuando termina el acceso a la sección crítica, comprueba si hay algún proceso bloqueado esperando entrar y le pasa el testigo (signal sobre el semáforo donde se ha bloqueado).
Si no hay ninigún proceso bloqueado suelta el testigo (signal sobre el mutex).
Cuando finaliza la ejecución de todos los threads, la cola contiene 2 elementos encolados.
Esto se debe a que los threads escritores encolan un total de 32 elementos y los threads lectores desencolan un total de 30 elementos.

Ficheros fuente:

main_p3_e1.cpp
ConcurrentBoundedQueue.cpp
ConcurrentBoundedQueue.hpp