Recolectores y Productores

Ejercicio N° 2

Objetivos	● Diseño y construcción de sistemas orientados a objetos ● Diseño y construcción de sistemas con procesamiento concurrente ● Protección de los recursos compartidos
Instancias de Entrega	**Entrega 1:** clase 6 (19/05/2020). **Entrega 2:** clase 8 (02/06/2020).
Temas de Repaso	● Threads en C++11 ● Clases en C++11
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Orientación a objetos del sistema ● Empleo de estructuras comunes C++ (string, fstreams, etc) en reemplazo de su contrapartida en C (char*, FILE*, etc) ● Uso de **const** en la definición de métodos y parámetros ● Empleo de constructores y destructores de forma simétrica ● Buen uso del stack para construcción de objetos automáticos ● Ausencia de condiciones de carrera e interbloqueo en el acceso a recursos ● Buen uso de Mutex, Condition Variables y Monitores para el acceso a recursos compartido

Índice

Introducción

Inspirados en la infinidad de juegos de estrategia que jugamos durante una reciente cuarentena, vamos a simular el comportamiento de un pequeño poblado ubicado en cualquiera de los universos pertenecientes a estos juegos.

Descripción

Nuestro poblado va a contar con un conjunto de trabajadores, que serán recolectores (de materias primas), y productores (que convierten materias primas en puntos de beneficio). Ambos tipos de trabajadores los vamos a modelar como threads.

Habrá tres tipos de recolectores:

1. **Agricultores**: son recolectores de **trigo**.
2. **Leñadores**: son recolectores de **madera**.
3. **Mineros**: recolectan **hierro** y **carbón**. *Tener en mente que este tipo de trabajador puede recolectar dos tipos distintos de materia prima.*

Y tres tipos de productores:

1. **Cocineros**: convierten 2 unidades de **trigo** más 1 de **carbón**, en 5 **puntos de beneficio**.
2. **Carpinteros**: convierten 3 unidades de **madera** más 1 de **hierro** en 2 **puntos de beneficio**.
3. **Armeros**: convierten 2 unidades de **carbón** más 2 de **hierro** en 3 **puntos de beneficio**.

La configuración de las cantidades de cada tipo de trabajador, estará en un archivo cuyo nombre vamos a extraer de la línea de comandos.

Cada **conjunto de recolectores** del mismo tipo recibirá los recursos desde una cola bloqueante, que deberemos implementar *utilizando condition variables*, dormir durante 50 ms (para simular trabajo), y depositarlos en un inventario de materias primas compartido. *Para disminuir la complejidad, se recomienda usar una cola bloqueante por cada tipo de recolector (no por tipo de recurso, aunque está permitido), y un solo inventario general*.

Todos los **productores** van a consumir los recursos depositados en el inventario general **tan pronto como estén disponibles** (*entender claramente el significado de esto, relacionarlo con condition variables, y describir en el informe la solución*) dormirán durante 60 ms, y depositarán los puntos en un contador de puntos general (que también deberá ser uno solo para todo el escenario. *Nuevamente, esto es clave en el razonamiento y en el informe*).

El hilo principal será el encargado de *spawnear* el resto de los hilos, leer un segundo archivo con el contenido del mapa, y repartir los recursos en las colas bloqueantes, que son las entradas de los recolectores. Luego, este hilo se encargará de orquestar la finalización ordenada del proceso e imprimirá las estadísticas finales.

El único uso permitido de funciones que sirvan para gastar tiempo (usleep) es para simular el trabajo especificado en esta sección. NO está permitido usarlas como mecanismo de sincronización.

Nota: *Si bien hay numerosas maneras de implementar esto, es obligatorio seguir estos lineamientos con el propósito de poner en práctica el uso de todas las herramientas de concurrencia que se ven en clase (threads, mutex, y condition variables). También es muy recomendable incluir explicaciones breves de cómo implementaron estas partes claves en sus soluciones.*

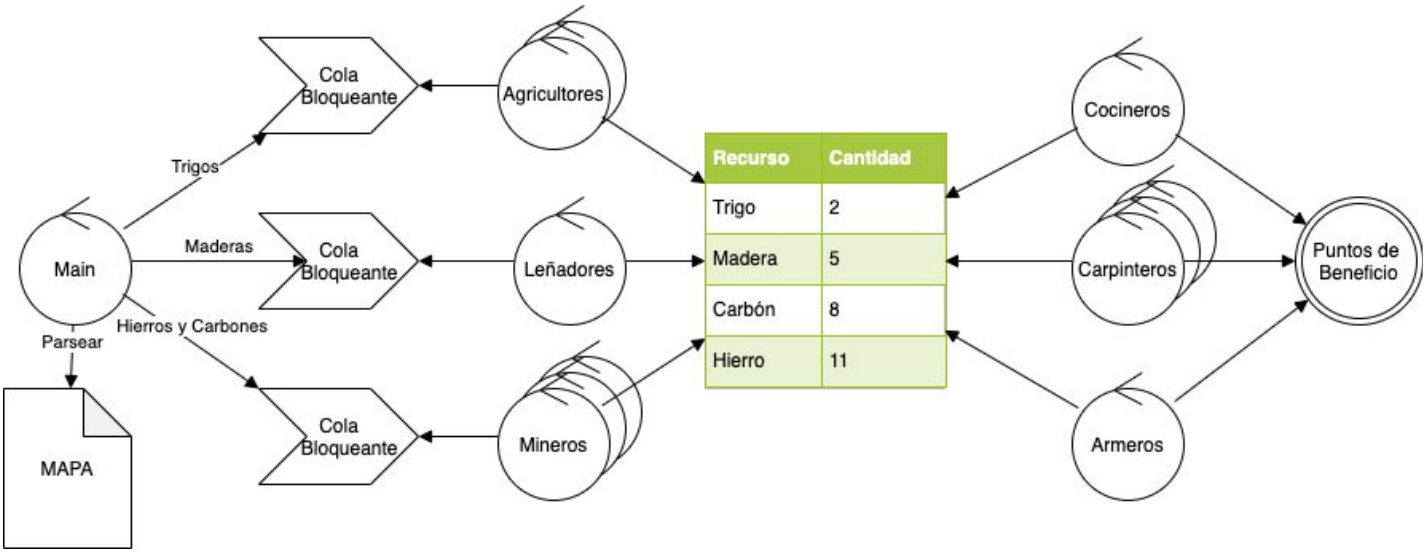

Imagen 1. En la imagen se puede ver un diagrama indicando los hilos participantes y los recursos compartidos. Notar que el sentido de las flechas indica una dependencia y no flujo de datos.
El hilo main lee los datos del mapa y los reparte en las colas bloqueantes; cada recolector pasa las materias primas al inventario (la tabla del centro); y cada productor saca esos recursos del inventario y genera puntos de beneficio, que sumará al contador que está más a la derecha.
Notar que para sincronizar los accesos al inventario se debe esperar por la condición de que "estén disponibles todos los recursos que necesito": esto se puede resolver con una condition variable (para simplificar, usar una sola y no una por cada tipo de productor). También hay que contemplar que no todos los productores necesitan las mismas materias primas, por lo que será importante tener en cuenta la diferencia entre notify y notify_all.

Archivos de entrada

En la sección anterior se mencionaron dos archivos de entrada para nuestro programa.

Configuración de Trabajadores

Consiste en un archivo de texto que indica las cantidades de cada tipo de trabajador. Se muestra un ejemplo con números arbitrarios a continuación:

```
Agricultores=2
Leniadores=1
Mineros=3
Cocineros=1
Carpinteros=3
Armeros=1
```

Mapa de Materias Primas

Las materias primas también serán entrada de nuestro programa, y consistirán en una matriz de caracteres en mayúsculas. Dichas mayúsculas son las iniciales de cada tipo de materia prima: cada '**T**' se interpreta como una unidad de **Trigo**, y los demás casos son análogos.

```
TTTMCHHHC
TTTMMCCCM
MMMMMMMMM
```

Formato de Línea de Comandos

El programa se ejecutará pasando por línea de comandos los nombres de los dos archivos de entrada:

```
./tp trabajadores.txt mapa.txt
```

Códigos de Retorno

El programa retornará 0 si todo se ejecutó correctamente, o 1 en caso de algún error al abrir, leer, o parsear alguno de los parámetros o archivos. En caso de que el alumno encuentre otros tipos de errores, se permite terminar con cualquier valor de retorno, y se garantiza que no habrá casos de prueba que los validen.

Entrada y Salida Estándar

La entrada estándar no será utilizada.
La salida estándar de errores es libre, y el alumno puede utilizarla para imprimir mensajes útiles de error, a fin de corroborar alguna teoría al subir la implementación a Sercom en caso de tener problemas que solamente se reproduzcan en ese entorno.
La salida estándar se utilizará al final de cada ejecución para mostrar cuántos recursos quedaron en el inventario, y cuántos puntos de beneficio se lograron sumar. El formato será el siguiente:

```
Recursos restantes:\n
  - Trigo: <unidades-de-trigo-restantes>\n
  - Madera: <unidades-de-madera-restantes>\n
  - Carbon: <unidades-de-carbon-restantes>\n
  - Hierro: <unidades-de-hierro-restantes>\n
\n
Puntos de Beneficio acumulados: <cantidad-de-puntos-final>\n
```

Notar que antes de cada guión hay dos espacios.

Ejemplos de Ejecución

A continuación, se mostrarán algunos ejemplos de ejecución.

Ejemplo 1: Con recolectores, pero sin productores

Consideremos un archivo de trabajadores llamado "**trabajadores.cfg"** que solamente tiene recolectores:

```
Agricultores=2
Leniadores=2
Mineros=2
Cocineros=0
Carpinteros=0
Armeros=0
```

Y un archivo de mapa "**mapa.txt**" como el que mostramos antes:

```
TTTMCHHHC
TTTMMCCCM
MMMMMMMMM
```

Lo que esperamos en este caso, es que los recolectores trasladen todas las materias primas desde sus colas bloqueantes de entrada hacia el inventario (ver la tabla en el centro de la **Imagen 1**), pero que no haya ningún productor transforme esas materias primas en puntos de beneficio (la segunda columna de threads de dicha imagen estaría vacía).
Por lo tanto, si ejecutamos el programa:

```
./tp trabajadores.cfg mapa.txt
```

Esperaremos ver una salida con puntos de beneficio nulos, pero con los recursos sumados en la impresión del inventario:

```
Recursos restantes:
  - Trigo: 6
  - Madera: 13
  - Carbon: 5
  - Hierro: 3

Puntos de Beneficio acumulados: 0
```

Ejemplo 2: Ejemplo completo

En este caso, vamos a configurar la misma cantidad de trabajadores presentes en la **Imagen 1**:

```
Agricultores=2
Leniadores=1
Mineros=3
Cocineros=1
```

```
Carpinteros=3
Armeros=1
```

Y nuevamente el mismo mapa de materias primas:

```
TTTMCHHHC
TTTMMCCCM
MMMMMMMMM
```

En este caso, vamos a ver que todos los recolectores van a agregar sus respectivos recursos cada 50 ms al inventario, por lo que **es relevante la cantidad de cada tipo de recolector** para el resultado final. Una situación análoga sucede con los productores, y por lo tanto es probable que si hacen pruebas manuales obtengan distintos resultados. *Los casos de prueba en sercom están pensados para que esto no pase*.

- El primer productor en tener sus materias primas disponibles para trabajar será el cocinero, ya que hay dos agricultores y tres (más de un) mineros trabajando durante los primeros 50 ms. Eso hará que durante los siguientes 60 ms esté ocupado, y sume los primeros 5 puntos de beneficio en el milisegundo 110 aproximadamente.
- Luego (*es recomendable ponerse a pensar por qué*) llegará el armero en el milisegundo 160 a sumar otros 3 puntos de beneficio, dejando el contador en 8.
- Alrededor del milisegundo 210, llegarán tanto el cocinero como uno de los carpinteros a sumar 5 y 2 puntos respectivamente. En este punto no podemos estimar quién llegará antes, pero sí que después de ambas operaciones habrá 15 puntos en el contador.
- Luego, el único productor que tendrá recursos durante el resto de la ejecución será el cocinero, lo que dejará el contador en su valor final de 20 puntos.

La salida estándar mostrará entonces:

```
Recursos restantes:
  - Trigo: 0
  - Madera: 10
  - Carbon: 3
  - Hierro: 0

Puntos de Beneficio acumulados: 20
```

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C (C99) / ISO C++11.
2. Está prohibido el uso de variables globales.
3. Leer atentamente los lineamientos y recomendaciones.

Recomendaciones

1. Leer las partes en *cursiva* del enunciado. Se dejaron varias recomendaciones allí.
2. Entender los ejemplos de código en https://en.cppreference.com/w/cpp/thread/condition_variable y en https://es.cppreference.com/w/cpp/thread/condition_variable. No son el mismo ejemplo! Y son dos casos de uso claves.
3. Notar e investigar las diferencias entre lock_guard, unique_lock, y shared_lock, y el problema de lectores y escritores.
4. Generar casos de prueba propios y hacer un análisis parecido al del ejemplo 2. No asustarse si en algunas ejecuciones se ven resultados distintos, pero sí que sean números compatibles entre sí (los puntos de beneficio tienen que tener las materias primas correspondientes en el archivo de entrada, y el sobrante tiene que quedar al final en el inventario)

Referencias

[1] Condition Variables en C++: https://en.cppreference.com/w/cpp/thread/condition_variable
[2] Mutex en C++: https://es.cppreference.com/w/cpp/thread/mutex
[3] Threads en C++: https://es.cppreference.com/w/cpp/thread/thread