Proyecto Final (15%)

Simulación de Línea de Producción de Electrodomésticos
(utilizando procesos, hilos, mutexes y semáforos en Linux Ubuntu)
Fecha de entrega y defensa: lunes 17 de noviembre de 2025

Introducción

Este proyecto consiste en desarrollar una aplicación gráfica en C++ con QT Creator y Ubuntu Desktop que permita mostrar una simulación de una línea de producción automatizada de electrodomésticos con múltiples estaciones de trabajo. Cada estación de montaje será representada como hilos/procesos que se comunican y sincronizan entre sí. Cada estación procesa una pieza y la transfiere a la siguiente estación hasta obtener el producto final. La sincronización entre estaciones se realizará utilizando semáforos, mutexes y variables de condición (condiciones de competencia), para asegurar que no haya conflictos en el acceso a recursos compartidos.

Objetivos

La aplicación gráfica deberá cumplir con los siguientes objetivos:

1. Utilizar el ambiente de programación QT Creator con C++ para implementar una aplicación gráfica en la que múltiples procesos e hilos interactúen de manera sincronizada y segura mediante el uso de semáforos y mutexes.
2. Simular a través de animaciones gráficas una línea de producción automatizada de electrodomésticos, que permita la comunicación y sincronización de procesos e hilos (muy importante).
3. Iniciar, detener y eliminar procesos e hilos en forma automática o por medio de un administrador del sistema.
4. Mostrar un listado de todos los procesos e hilos que están funcionando dentro del sistema.
5. Realizar la comunicación y sincronización de procesos e hilos a través de las librearías QThread, QSemaphore, QMutex, entre otras.

Contexto

Desarrollar una aplicación en QT Creator con C/C++ y ambiente operativo Linux Ubuntu que permita SIMULAR A TRAVÉS DE UNA INTERFAZ GRÁFICA la comunicación y sincronización de procesos e hilos por medio de un SIMULADOR DE LÍNEA DE PRODUCCIÓN AUTOMATIZADA DE ELECTRODOMÉSTICOS. La aplicación gráfica deberá permitir la comunicación, concurrencia y sincronización de procesos e hilos.

Requerimientos del Sistema

La aplicación gráfica deberá cumplir con los siguientes requerimientos:

1. **Clases principales y funcionalidades**
   a. MainWindow (QT GUI): Interfaz para visualizar estaciones, colas, logs y métricas. Permite configurar parámetros de la simulación.
   b. ProductionController: Crea estaciones, administra hilos/procesos y gestiona buffers y sincronización.
   c. Station (clase base): Representa una estación de trabajo. Utiliza hilos (QThread o std::thread).
   d. Assembler / Tester (heredadas de Station): Estaciones de trabajo especializadas en tareas concretas.
   e. Buffer: Cola protegida por mutexes y variables de condición. Sirve para conectar estaciones por medio de pipes.
   f. PipeComm / SocketComm (opcional): Módulos para comunicación IPC (Inter-Process Communication) entre procesos (puede ser por medio de pipes).
   g. Logger: Administra registros y métricas del sistema (tiempos, rendimiento (throughput), estados).

2. **Interfaz Gráfica de Usuario (GUI):** Se deberá diseñar una interfaz gráfica intuitiva y atractiva que permita a los usuarios interactuar con el sistema. Esta interfaz deberá incluir botones, ventanas y otros elementos gráficos para representar los recursos gestionados. Además, animaciones, imágenes, mensajes interactivos a través de procesos e hilos. Tome en cuenta lo siguiente:
   a. Estaciones de trabajo: Se requiere de múltiples estaciones (al menos 5) en la línea de producción (ensamblaje, inspección de calidad, empaquetado, entre otras). Se deberá mostrar el estado de cada estación de trabajo y permitir la gestión individual de cada estación (iniciar, detener, pausar).
   b. Comunicación entre estaciones: Las estaciones se comunicarán mediante señales, pipes, colas de mensajes, entre otros, para indicar el progreso de los productos o realizar alertas de problemas. Los hilos/procesos deberán interactuar entre sí para realizar operaciones concurrentes con los recursos. La comunicación deberá ser efectiva y sincronizada para evitar conflictos y condiciones de carreras (por medio de alertas, mensajes, envío de solicitudes, respuestas a mensajes para asignación o liberación de recursos, entre otros).
   c. Sincronización avanzada: Se deberán utilizar mutexes/semáforos para gestionar la disponibilidad de partes entre estaciones y proteger los recursos compartidos.
   d. Componentes claves:
      - Hilos/Procesos: para el manejo de las estaciones y las diversas tareas de cada estación (ensamblaje, inspección de calidad, empaquetado, entro otros).
      - Semáforos y Mutexes: Para coordinar el flujo de partes entre estaciones y para la protección de variables compartidas y recursos.
      - Señales y Slots: ambas de QT, para la comunicación interna y para la interacción con la interfaz gráfica.
      - Iniciar y detener estaciones (hilos/procesos): en forma automática e interactiva (por medio de botones). Por ejemplo, por medio de etiquetas

(QLabel) se podrá mostrar el estado de cada estación actualizada en tiempo real.

- Asignar recursos a los procesos: en forma automática, a través de íconos, imágenes o etiquetas se podrá mostrar el estado (disponible, en uso, fuera de servicio).
- Mostrar un listado de estaciones y recursos: listado en tiempo real de todos los procesos e hilos que están interactuando con los recursos. Esto puede ser a través de una lista desplegable, una tabla o una sección dedicada en la interfaz.

3. **Persistencia de Datos:** La información de estaciones, tareas de cada estación, recursos asignados y demás elementos involucrados, deberá conservarse de manera adecuada, de modo que los datos no se pierdan después de cerrar la aplicación. Esto se puede lograr mediante el uso de archivos JSON, XML, bases de datos u otros métodos de almacenamiento. Tome en cuenta lo siguiente:
   a. Resguardo de estaciones, tareas y recursos: Toda la gestión de datos debe almacenarse de manera persistente para que puedan recuperarse incluso después de cerrar y reiniciar la aplicación. Esto puede lograrse mediante el uso de archivos de datos o una base de datos local.
   b. Carga de Datos al Iniciar: Al iniciar la aplicación, se deben cargar los datos previamente almacenados para restaurar el sistema al último estado antes de cerrar la aplicación.

4. **Hilos de Mantenimiento:** Adicionalmente, el sistema deberá contar con hilos de mantenimiento prioritarios para su correcto funcionamiento. Estos hilos deberán estar siempre activos, lanzar por pantalla mensajes de funcionamiento, de indicaciones, de advertencias, entre otros. Cada vez que entren en funcionamiento deberán lanzar un mensaje por pantalla indicando su nombre y la acción realizada. Estos hilos serán los siguientes:
   a. GeneralCleanThreads: se encargará de hacer limpieza general (reseteo) de todos los hilos y recursos que existan en la aplicación gráfica. Es decir, deberá eliminar todos los hilos y recursos existentes y volverlos a activar.
   b. GeneralLogs: se encargará de recopilar información de funcionamiento de la aplicación gráfica. Es decir, deberá mostrar una bitácora de funcionamiento del sistema, de hilos lanzados, de recursos utilizados y la actividad realizada (ejemplo: se asignó un nuevo recurso, se eliminó un recurso, se envió una solicitud al proceso o hilo x, ejecutó una acción, entre otros).
   c. GeneralStats: se encargará de mostrar un gráfico con estadísticas de funcionamiento de la aplicación gráfica: gestión de recursos, comunicación entre procesos e hilos, sincronización con mutexes, persistencia, acciones realizadas, hilos lanzados, recursos consumidos, entre otros.

Evaluación del Proyecto:

Interfaz Gráfica de Usuario (GUI)[1]	50pts
Persistencia de Datos[2]	25pts
Hilos de Mantenimiento[3]	25pts
TOTAL	**100pts**

1. Estaciones de trabajo, comunicación (pipes, señales, slots, colas de mensajes), sincronización, componentes claves
2. Almacenamiento de recursos, carga de datos.
3. GeneralCleanThreads, GeneralLogs, GeneralStats

Diseño lógico del sistema (UML de clases)

Incluye las principales entidades y sus relaciones:

1. MainWindow (Qt GUI)
   a. Interfaz gráfica: botones (iniciar, pausar, detener), panel visual con estaciones
   b. Observa el estado de la línea de producción
2. Product
   a. Atributos: id, type, currentState
   b. Métodos: advanceState(), showInfo()
3. WorkStation
   a. Atributos: id, name, taskType, inputBuffer, outputBuffer
   b. Métodos: processProduct(), submitProduct()
   c. Use: mutex + variables de condición para proteger buffers
4. ProductionLine
   a. Atributos: stationList, productsInProcess
   b. Métodos: startProduction(), stopProduction()
   c. Relación: agregación de estaciones de trabajo
5. ThreadManager
   a. Atributos: ThreadList, mutexControl
   b. Métodos: launchThreads(), stopThreads(), monitor()
6. PipeManager
   Administra pipes para la comunicación entre estaciones

Flujo dinámico a través de un diagrama de secuencia

Caso: Un producto que pasa por la línea de producción

1. MainWindow: ProductionLine.startProduction().
2. ProductionLine: crea threads de cada WorkStation.
3. Station 1: toma un producto del buffer, lo procesa, y lo envía al PipeManager.
4. PipeManager: pasa el producto a Station 2 (mediante pipe + condición de disponibilidad).
5. Station 2: repite el proceso anterior y pasa el producto a la siguiente estación por medio del PipeManager.
6. Cuando el producto alcanza la última estación, se marca como producto terminado.
7. ProductionLine: actualiza el estado en la MainWindow para mostrar progreso.

Publicación en mediación

1. El proyecto será publicado en el entorno del curso de la plataforma de mediación virtual (METICS). Cada grupo deberá publicar lo siguiente:

    a. Todo el código fuente (última versión) en un archivo empaquetado en formato .zip ó .rar. Incluyendo todas las librerías, frameworks, y demás recursos utilizados y descargados de internet.
    b. Un archivo tipo **Readme.txt** con información útil y relevante para una óptima utilización del sistema por parte del usuario final.
    c. Toda la bibliografía consultada deberá agregarse en formato APA en un archivo pdf llamado **References.pdf.**

2. Cada grupo deberá realizar una presentación del proyecto con la funcionalidad del sistema. Contempla:

    a. Funcionamiento del sistema (muy importante).
    b. Preguntas y respuestas.

3. El proyecto se desarrollará en los grupos definidos en el curso. Salvo la debida autorización del profesor, no se podrá cambiar esta disposición.