Trabajo Práctico

Introducción

Una importante cadena de heladerías desea abrir una sucursal completamente robotizada. Los clientes harán su pedido sobre una interfáz gráfica eligiendo gustos y tamaño. Luego un conjunto de robots podrán resolver los pedidos de forma concurrente, tal como lo hacen los heladeros los días de calor. Así mismo, cada contenedor con un gusto de helado podrá ser utilizado solamente por un robot a la vez. Finalmente se le entregará su helado al cliente; y solamente se le cobrará en la tarjeta si efectivamente se pudo completar su pedido.

Objetivo

Deberán implementar un conjunto de aplicaciones en Rust que modele el sistema de la heladería.

Requerimientos

- Una aplicación modelará las interfaces con los clientes, las cuales generarán ordenes de pedido según un archivo de pedidos simulado. Deben soportar varios gustos y tamaños posibles de contenedor. Habrán varias instancias, tantas como pantallas disponibles en el local.
- Otra aplicación simulará los robots, los cuales tomarán un pedido y deberán obtener los ingredientes necesarios para llenar el contenedor. El tamaño del contenedor definirá el tiempo para simular cada pedido. Nuevamente, serán varios los robots que despachen pedidos.
- De alguna forma el sistema deberá modelar los contendores con gustos y el stock disponible en cada uno. Eventualmente algun gusto quedará sin stock, y un robot que necesite ese gusto deberá cancelar el pedido.
- Se debe considerar la captura del pago al momento de realizar el pedido, y el cobro efectivo al momento de entrega, o bien su cancelación. Modelar el gateway de pagos con una aplicación simple que loguea. Considerar que al momento de capturar el pago, el gateway puede rechazar la tarjeta aleatoriamente con una probabilidad.

- El sistema debe ser resiliente, soportando la caida de algunas instancias de las aplicaciones.

Requerimientos no funcionales

Los siguientes son los requerimientos no funcionales para la resolución de los ejercicios:

- El proyecto deberá ser desarrollado en lenguaje Rust, usando las herramientas de la biblioteca estándar.
- Por lo menos alguna, si no todas las aplicaciones implementadas deben funcionar utilizando el **modelo de actores**.
- En el modelado de la solución se deberán utilizar una o mas de las herramientas de concurrencia distribuida mostradas en la cátedra. Por ejemplo exclusiones mutuas distribuidas, elección de líder, algorítmos de anillo, commits de dos fases, etc.
- No se permite utilizar **crates** externos, salvo los explícitamente mencionados en este enunciado, los utilizados en las clases, o los autorizados expresamente por los profesores.
- El código fuente debe compilarse en la última versión stable del compilador y no se permite utilizar bloques unsafe.
- El código deberá funcionar en ambiente Unix / Linux.
- El programa deberá ejecutarse en la línea de comandos.
- La compilación no debe arrojar **warnings** del compilador, ni del linter **clippy**.
- Las funciones y los tipos de datos (**struct**) deben estar documentadas siguiendo el estándar de **cargo doc**.
- El código debe formatearse utilizando **cargo fmt**.
- Cada tipo de dato implementado debe ser colocado en una unidad de compilación (archivo fuente) independiente.

Entregas

La resolución del presente proyecto es en grupos de tres integrantes.

Las entregas del proyecto se realizarán mediante Github Classroom. Cada grupo tendrá un repositorio disponible para hacer diferentes commits con el objetivo de resolver el problema propuesto.

Primera entrega: Diseño

Deberán entregar un informe en formato Markdown en el `README.md` del repositorio que contenga una explicación del diseño y de las decisiones tomadas para la implementación de la solución, asi como diagramas de threads y procesos, y la comunicación entre los mismos; y diagramas de las entidades principales.

Se recomienda fuertemente que las implementaciones de las ideas de diseño se encuentren realizadas mínimamente, para validar el mismo.

Fecha máxima de Entrega: 12 de Junio de 2024

El diseño será evaluado por la cátedra y de no presentarse a término el trabajo quedará automaticamente desaprobado.

La cátedra podrá solicitar correciones que deberán realizarse en el mismo diseño, y puntos de mejora que deberán tenerse en cuenta durante la implementación.

Se podrán hacer commits hasta el día de la entrega a las 19 hs Arg, luego el sistema automáticamente quitará el acceso de escritura.

Segunda entrega:** 26 de Junio de 2024

- Deberá incluirse el código de la solución completa.
- Deberá actualizarse el readme, con una sección dedicada a cambios que se hayan realizado desde la primera entrega. Además debe incluir cualquier explicación y/o set de comandos necesarios para la ejecución de los programas.
- Debera incluir un enlace a un video de no más de 7 minutos donde cada integrante presente su participación principal en el proyecto, explicando las partes más relevantes del código que trabajó. El video debe mostrar además la solución funcionando en diferentes casos de interés.

La cátedra podrá solicitar luego correcciones donde se encuentren errores severos, sobre todo en el uso de herramientas de concurrencia; o bien desviaciones respecto al diseño pactado inicialmente.

De tener correciones para realizar, las mismas deben realizarse y aprobarse con anterioridad a la presentación a examen final.

Entrega final

Durante el examen final, los alumnos deberán poder responder preguntas sobre su diseño e implementación, y su participación personal en el trabajo; además de las preguntas y situaciones problemáticas propias de los contenidos presentados por la cátedra durante toda la materia.

Evaluación

Principios teóricos y corrección de bugs

Los alumnos presentarán el código de su solución primero en video, y luego personalmente, con foco en el uso de las diferentes herramientas de concurrencia. Deberán poder explicar desde los conceptos teóricos vistos en clase cómo se comportará potencialmente su solución ante problemas de concurrencia (por ejemplo ausencia de deadlocks).

En caso de que la solución no se comportara de forma esperada, deberán poder explicar las causas y sus posibles rectificaciones.

Casos de prueba

Se someterá a la aplicación a diferentes casos de prueba que validen la correcta aplicación de las herramientas de concurrencia y la resiliencia de las distintas entidades.

Informe

El informe debe poder dar cuenta de todas las decisiones tomadas para implementar la solución, incluyendo diferentes diagramas y se debe poder utilizar como soporte para que el equipo presente su trabajo.

Organización del código

El código debe organizarse respetando los criterios de buen diseño y en particular aprovechando las herramientas recomendadas por Rust. Se prohibe el uso de bloques `unsafe`.

Tests automatizados

La entrega debe contar con tests automatizados que prueben diferentes casos. Se considerará en especial aquellos que pongan a prueba el uso de las herramientas de concurrencia.

Presentación en término

El trabajo deberá entregarse para la fecha estipulada. La presentación fuera de término sin coordinación con antelación con el profesor influirá negativamente en la nota final.