



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2523 — Sistemas Distribuidos — 2024 - 2

Tarea 2

Martes 15 de octubre, 2024

Fecha de entrega: Miércoles 30 de octubre, hasta las 23:59

Instrucciones

Esta tarea tiene como objetivo aplicar sus conocimientos sobre algoritmos distribuidos para resolver un problema relacionado a la detección de terminación e instantáneas globales, mediante el uso del lenguaje Go 1.21.

El entregable corresponderá a un archivo **zip** con el nombre **numero_alumno.zip** tal que al descomprimirlo se muestre la carpeta correspondiente y dentro contenga un archivo **main.go** y un archivo **bibliografia.txt** con todas las referencias que hayan utilizado para implementar su tarea.

Si tienen dudas, por favor utilicen **Foro de Canvas** del curso. Además, el próximo jueves 24 de octubre en horario de ayudantía tendrán la primera **sala de ayuda** para responder dudas acerca de la tarea. También habrá una segunda **sala de ayuda** el miércoles 30 en horario de clases.

Introducción

Luego de haber resuelto satisfactoriamente el problema de las transacciones ocurrido en la Fonda Don Yadrán, el Departamento de Ciencia de la Computación se prepara para organizar otro evento junto a los estudiantes: el gran **DCCalabaza**.

Ante esto, los participantes del evento han decidido realizar una actividad para intercambiar dulces entre ellos. Sin embargo, el profesor Yadrán quiso realizar una pequeña travesura. Sin que nadie se diera cuenta, cambió uno de los dulces de uno de los estudiantes por un trozo de ajo envuelto en papel de dulce, el cual se fue traspasando entre cada participante.

Es por esta razón que tú, como estudiante de sistemas distribuidos, decides implementar un algoritmo para resolver este problema de instantáneas globales y así determinar cuál participante se quedó con el ajo en algún momento determinado.

Problema a resolver

Se les entregará un archivo de texto llamado **acciones.txt** con las acciones realizadas entre los participantes. Este archivo tendrá el siguiente formato:

Primero una fila con los números **n,m,l** indicando que habrán **n** participantes, **m** acciones y **l** será el **id** del participante que tenga el ajo inicialmente. Luego, habrán **m** filas con las acciones en el siguiente formato:

participante_1:acción:valor

Esta acción se lee del siguiente modo: **participante_1** ejecuta la acción **acción** con un valor de **valor**.

Las acciones posibles son las siguientes:

1. **SEND**: Acción donde un participante envía un dulce a otro con **id** igual a **valor**. Si el emisor posee el ajo, entonces se lo enviará al receptor. En otro caso, se envía un dulce.
2. **RECEIVE**: Acción donde un participante recibe un dulce o bien un *marker* de otro participante con **id** **valor**.
3. **WAIT**: Acción donde un participante debe esperar **valor segundos** antes de ejecutar la siguiente acción.
4. **SNAPSHOT**: Acción donde un participante emite una instantánea con **id** **valor**. Al hacer esto, el emisor debe enviar un *marker* con el **id** del **SNAPSHOT** a todos los demás participantes por medio de los respectivos canales.

Un ejemplo del archivo **acciones.txt** es el siguiente:

```
1 3,11,0
2 0:SEND:1
3 0:SNAPSHOT:0
4 0:RECEIVE:1
5 0:RECEIVE:2
6 1:RECEIVE:0
7 1:RECEIVE:0
8 1:RECEIVE:2
9 2:WAIT:4
10 2:RECEIVE:0
11 2:RECEIVE:1
12 2:WAIT:3
```

Flujo

Al momento de leer el archivo **acciones.txt**, deberán hacer que los **n** participantes ejecuten sus respectivas acciones de manera **concurrente** y en el **mismo orden** en el que se indica en el archivo. En caso de que la acción sea un **SNAPSHOT**, se debe enviar un *marker* a todos los canales.

Si la acción realizada es un **RECEIVE** y se recibe un ajo, entonces el participante receptor será el nuevo poseedor del ajo. Por otra parte, si el mensaje recibido es un *marker* y este corresponde al primero del respectivo **SNAPSHOT**, se debe registrar el estado **stateatRecord**, **messageatRecord** y **ajoaatRecord**. Por último, luego de haber recibido todos los demás *markers* del resto de los participantes, se debe registrar el **channelatRecord**, el cual indica el estado de cada canal de entrada, además de **ajoaatMarker** que indica si el ajo estuvo en un canal o no.

Para esto, deben escribir la información de cada nodo en el archivo **snapshot_n.txt** donde **n** es el número de **SNAPSHOT** realizado, empezando desde el 0. En este archivo se debe escribir por cada nodo la siguiente información:

```
1 numero_nodo: int
2 stateatRecord: list[int] # últimos mensajes enviados por cada canal de output.
3 messageatRecord: list[int] # últimos mensajes recibidos por cada canal de input.
4 ajoaatRecord: bool # boolean que indica si el nodo tiene el ajo o no.
5 channelatRecord: list[int] # cantidad de mensajes en el canal de input en el momento.
6 ajoaatMarker: list[bool, int] # boolean que indica si el ajo esta en el canal, y cuál
```

En este caso, **stateatRecord** y **messageatRecord** son un arreglo de largo **n** inicializado con 0, el cual indica que no se ha enviado o recibido ningún mensaje por ese canal. El índice de cada posición indica el número de nodo por el cual se asocia cada canal. Luego, por cada mensaje enviado o recibido, se debe aumentar en 1 el

contador en la posición correspondiente. Por último, `channelatRecord` indica la cantidad de mensajes que habían en el canal de `input` desde el primer *marker* hasta recibir todos los *markers* de los demás canales.

Un ejemplo de `snapshot_n.txt` es el siguiente:

```
1 0:
2 stateatRecord: [0 1 0]
3 messageatRecord: [0 0 0]
4 ajoatRecord: false
5 channelatRecord: [0 0 0]
6 ajoatMarker: [false -1]
7
8 1:
9 stateatRecord: [0 0 0]
10 messageatRecord: [1 0 0]
11 ajoatRecord: true
12 channelatRecord: [0 0 0]
13 ajoatMarker: [false -1]
14
15 2:
16 stateatRecord: [0 0 0]
17 messageatRecord: [0 0 0]
18 ajoatRecord: false
19 channelatRecord: [0 0 0]
20 ajoatMarker: [false -1]
```

Consideraciones

Para evitar problemas de consistencia, al momento de realizar un **SNAPSHOT** y el nodo envíe automáticamente los *markers*, puedes asumir que los archivos de `acciones.txt` tendrán la cantidad suficiente de **RECEIVE** para recibir tanto los dulces enviados como los *markers* de otros participantes.

Entregable

La tarea debe ser un archivo `zip` con el nombre `numero_alumno.zip` tal que al descomprimirlo tenga una carpeta `numero_alumno` con la siguiente información

```
└─ numero_alumno
   └─ main.go
   └─ bibliografia.txt
```

La tarea debe compilar con `go build` y debe ejecutarse con

```
./main path_input
```

Donde `path_input` es la ruta al archivo de acciones. Por ejemplo: `acciones.txt`

Al momento de compilar con `go build` y ejecutar `./main path_input` se deben generar los archivos `snapshot_n.txt` correspondientes con los resultados.

Su tarea será revisada con cuidado y deben adherirse al Código de Honor de la Universidad. Además, en el archivo `bibliografia.txt` deben referenciar adecuadamente las fuentes bibliográficas que usen de acuerdo al reglamento de Integridad Académica, y el código que entreguen debe ser de su autoría.

Evaluación

La evaluación consta de dos partes:

- **Correctitud (4 pts.):** La solución entregada debe resolver el problema propuesto registrando los resultados esperados en los archivos `snapshot_n.txt`. Esta parte se corregirá mediante **tests automáticos**. Por lo tanto, sus archivos de salida deben respetar el mismo formato que la solución esperada. La correctitud consta de dos partes:
 - **Nodos (2 pts.):** La solución coincide con la salida esperada para `stateatRecord`, `messageatRecord` y `ajoatRecord`
 - **Canales (2 pts.):** La solución coincide con la salida esperada para `channelatRecord` y `ajoatMarker`
- **Instantáneas (2 pts.):** La solución resuelve correctamente el problema utilizando el algoritmo de instantáneas globales visto en clases.

Si su tarea no está en lenguaje Go y/o no compila con el comando `go build <filename>.go`, o alguna de las soluciones entregadas no ejecuta correctamente con los comandos indicados, se evaluará con nota mínima.

Su nota final corresponderá al total de puntos obtenidos más un punto base.