



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



## TAREA 7

### Sistemas Operativos

Integrantes:

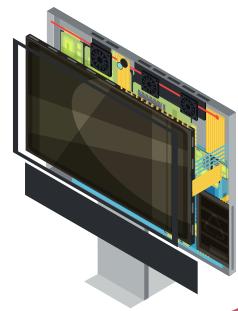
Mora Ayala José Antonio  
Ramírez Cotonieto Luis Fernando  
Torres Carrillo Josehf Miguel Ángel  
Tovar Jacuinde Rodrigo

Profesor:

Cortés Galicia Jorge

# Monitores y Semáforos

Los monitores y los semáforos se diseñaron para resolver problemas de sincronización en un sistema mono o multiprocesador, es decir, un sistema con una o varias CPU's que tienen acceso a una memoria compartida. La comunicación se logra compartiendo una zona de memoria donde los procesos guardan información, los semáforos y monitores se utilizan para sincronizar el acceso a dicha zona. Si pasamos a un sistema distribuido, con varias CPU's, cada una con su memoria privada, unidas mediante una red, estos mecanismos de comunicación entre procesos ya no se pueden aplicar. Además, estos dos mecanismos sirven para la sincronización, pero no para el intercambio de información entre procesos. En el siguiente apartado veremos un mecanismo que solventa ambos problemas.



## CPU

### Tipo Monitor

Un tipo monitor tiene un conjunto de operaciones definidas por el programador que gozan de la característica de exclusión mutua dentro del monitor.

### Tipo Abstracto de Datos

Un tipo, o un tipo abstracto de datos, agrupa una serie de datos privados con un conjunto de métodos públicos que se utilizan para operar sobre dichos datos.

Contiene la declaración de una serie de variables cuyos valores definen el estado de una instancia de dicho tipo, junto con los cuerpos de los procedimientos o funciones que operan sobre dichas variables

Un procedimiento definido dentro de un monitor sólo puede acceder a las variables declaradas localmente dentro del monitor y a sus parámetros formales. De forma similar, a las variables locales de un monitor sólo pueden acceder los procedimientos locales.

La estructura del monitor asegura que sólo un proceso esté activo cada vez dentro del monitor. En consecuencia, el programador no tiene que codificar explícitamente esta restricción de sincronización

## Restricciones

No es lo suficientemente potente como para modelar algunos esquemas de sincronización.

Un programador que necesite escribir un esquema de sincronización a medida puede definir una o más variables de tipo condition:

Las únicas operaciones que se pueden invocar en una variable de condición son:

wait()  
signal ()

### Implementación de un monitor utilizando semáforos

Consideremos ahora una posible implementación del mecanismo de monitor utilizando semáforos. Para cada monitor se proporciona un semáforo mutex inicializado con el valor 1. Un proceso debe ejecutar la operación wait (mutex) antes de entrar en el monitor y tiene que ejecutar operación signal (mutex) después de salir del monitor.

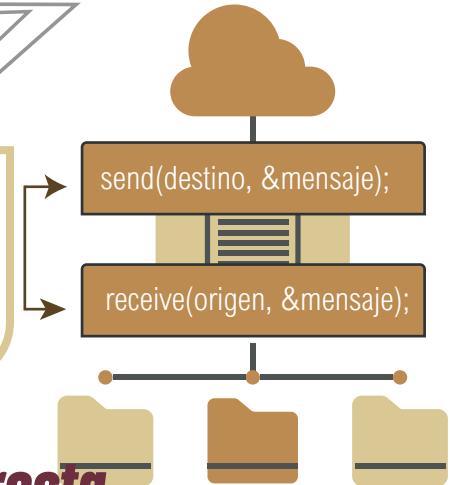
### Reanudación de procesos dentro de un monitor

Si hay varios procesos suspendidos en la condición x y algún proceso ejecuta una operación x.signal ()

Una solución sencilla consiste en usar el orden FCFS, de modo que el proceso que lleve más tiempo en espera se reanude en primer lugar.

# Transferencia de Mensajes

Este mecanismo de comunicación entre procesos se basa en dos primitivas: send y receive, las cuales, al igual que los semáforos, y a diferencia de los monitores, son servicios (llamadas al sistema) proporcionados por el sistema operativo. Los lenguajes de programación tendrán rutinas de biblioteca para invocar tales servicios desde un programa. Dichas rutinas de biblioteca tendrán un formato parecido al siguiente:



## Transferencia de mensajes con designación directa

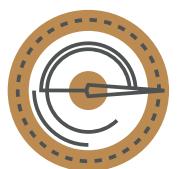
En este sistema, tanto el proceso emisor como el proceso receptor deben especificar explícitamente el proceso destino y origen respectivamente. Las funciones tendrá un formato similar al siguiente:



## Modo de Sincronización "CITA"

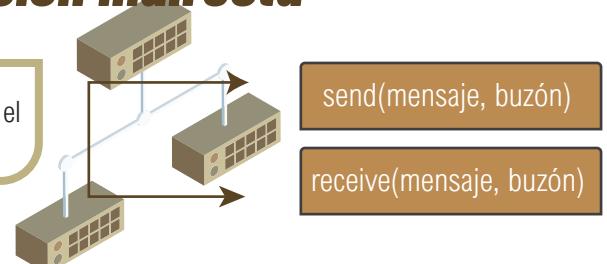
En cuanto a la sincronización, viene determinada por la ausencia de un lugar donde almacenar el mensaje.

- Si se ejecuta send antes de receive en el proceso destino, el proceso emisor se bloquea hasta la ejecución de receive, momento en el cual el mensaje se puede copiar de manera directa desde el emisor al receptor sin almacenamiento intermedio.
- Análogamente, si se ejecuta primero receive, el receptor se bloquea hasta que se ejecute una operación send en el proceso emisor.



## Transferencia de mensajes con designación indirecta

En este sistema, los mensajes se envían o reciben de unos objetos proporcionados por el sistema operativo denominados buzones.



### Buzon

Un buzón es un lugar donde almacenar un número determinado de mensajes, la cantidad de mensajes que alberga el buzón se especifica al crearlo. Al utilizar buzones, los parámetros de dirección en las llamadas send y receive son buzones, y no procesos:

Si se utilizan buzones, el emisor puede almacenar sus mensajes en un buzón, pudiendo el receptor recuperarlos posteriormente.

El sistema operativo debe proporcionar servicios para la creación y destrucción de buzones, así como para la especificación de los usuarios y/o procesos que pueden utilizar un buzón.

## Modo de Sincronización

01

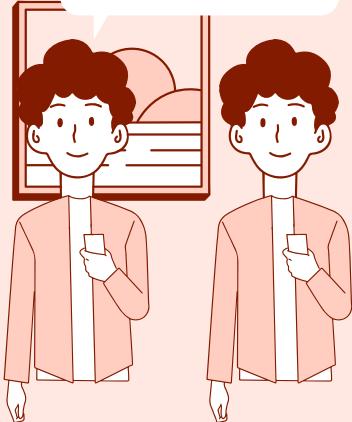
En cuanto a la sincronización, un proceso que haga una operación send sobre un buzón, sólo se bloquea si dicho buzón está lleno.

02

Un proceso que realice una operación receive sobre un buzón, sólo se bloquea si el buzón está vacío, es decir, no contiene mensajes.

# El problema de lectores-escritores

Nosotros  
somos los  
gemelos  
lectores



Los gemelos hacen referencia a los procesos de lectura a la base de datos

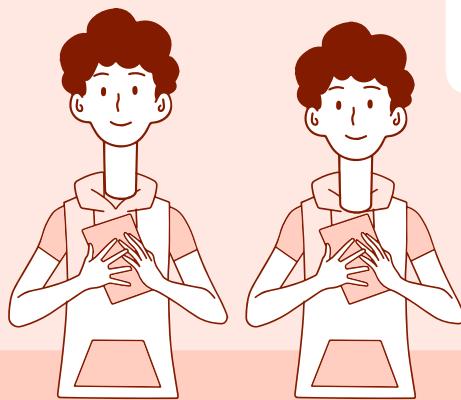
para nuestro caso la base de datos seran las revistas que el par de amigos escriben cada lunes

Nosotros  
somos unos  
amigos con  
una editorial

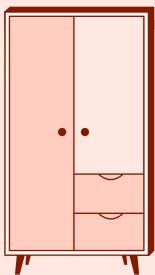
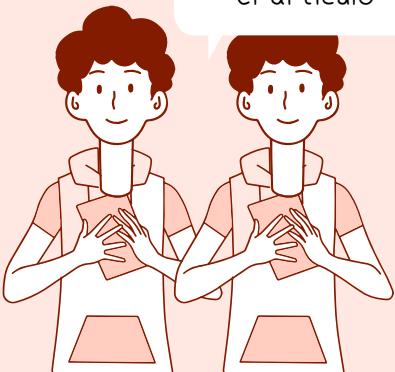


Los amigos escritores por su lado serán los procesos de escritura

Hermano YA  
viste!  
ACaban de  
publicar otro  
capítulo



WOOW! ya  
viste? los dos  
estamos  
leyendo al  
mismo tiempo  
el articulo



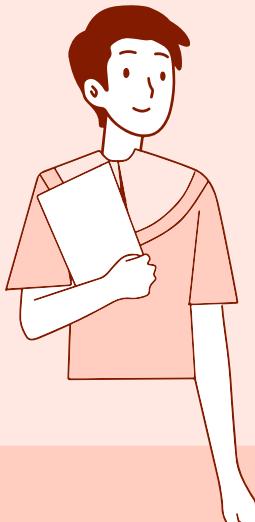
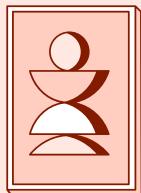
Esta genial la  
nueva película,  
hablare de ella  
en el siguiente  
articulo,  
comenzare a  
escribir



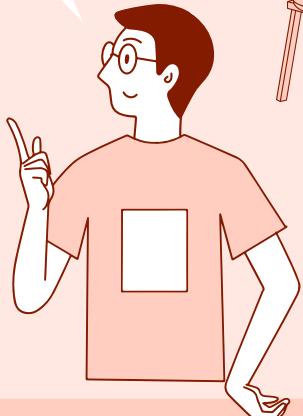
Tengo unas ideas  
geniales para la  
proxima publicacion!  
comenzare a  
escribirlas

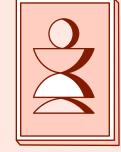


Como nos  
organizamos,  
no podemos  
escribir al  
mismo tiempo

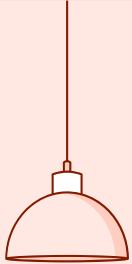
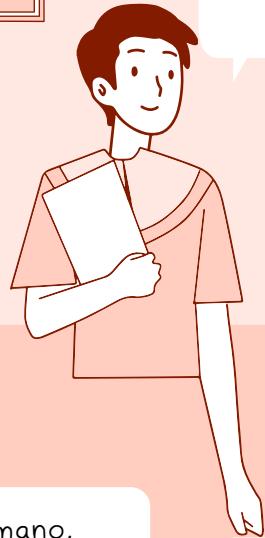


Claro que no podemos,  
de ser asi habria un  
caos asegurado

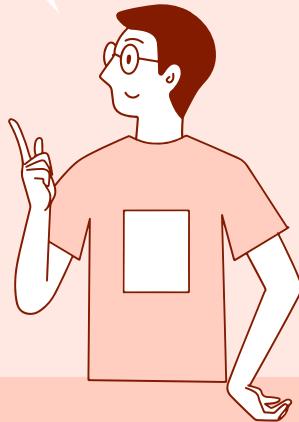




por supuesto y tambien hay que tener un acceso exclusivo y mientras escribimos que nadie tenga acceso



Lo mejor es que uno de los dos lo haga primero

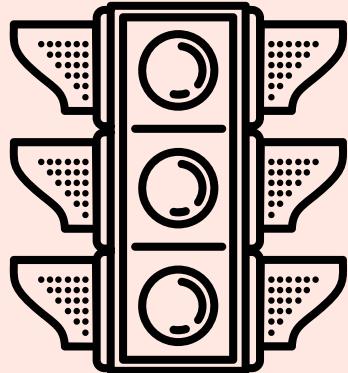


Hermano,  
desde la  
mañana no  
puedo leer

Seguramente alguno  
de los escritores esten  
editando, recuerda que  
mientras lo hacen no  
tenemos acceso



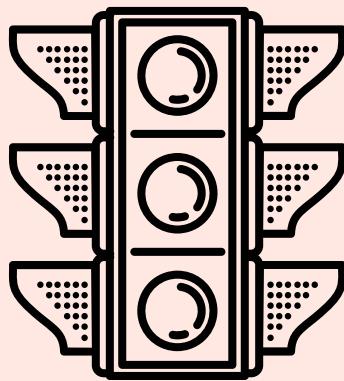
Ninguno de nuestros hermanos lectores tuvo que esperar a que el otro acabara, lo unico por lo que tuvieron que esperar es a que el escritor terminara su proceso



Aquí tenemos al que solucionara nuestro problema, se llama mutex y wrt, cuando esta en mutex se asegura de la exclusión mutua y en wrt funciona igual pero para los escritores y también los suelen usar o el primer o el ultimo lector

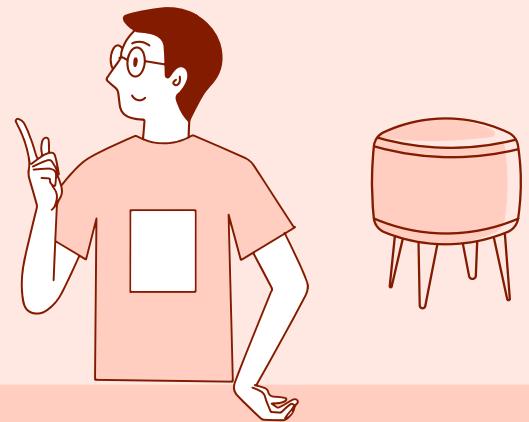


Genial, con ayuda de este semáforo nadie morirá de inanición y estaremos mas organizados!!

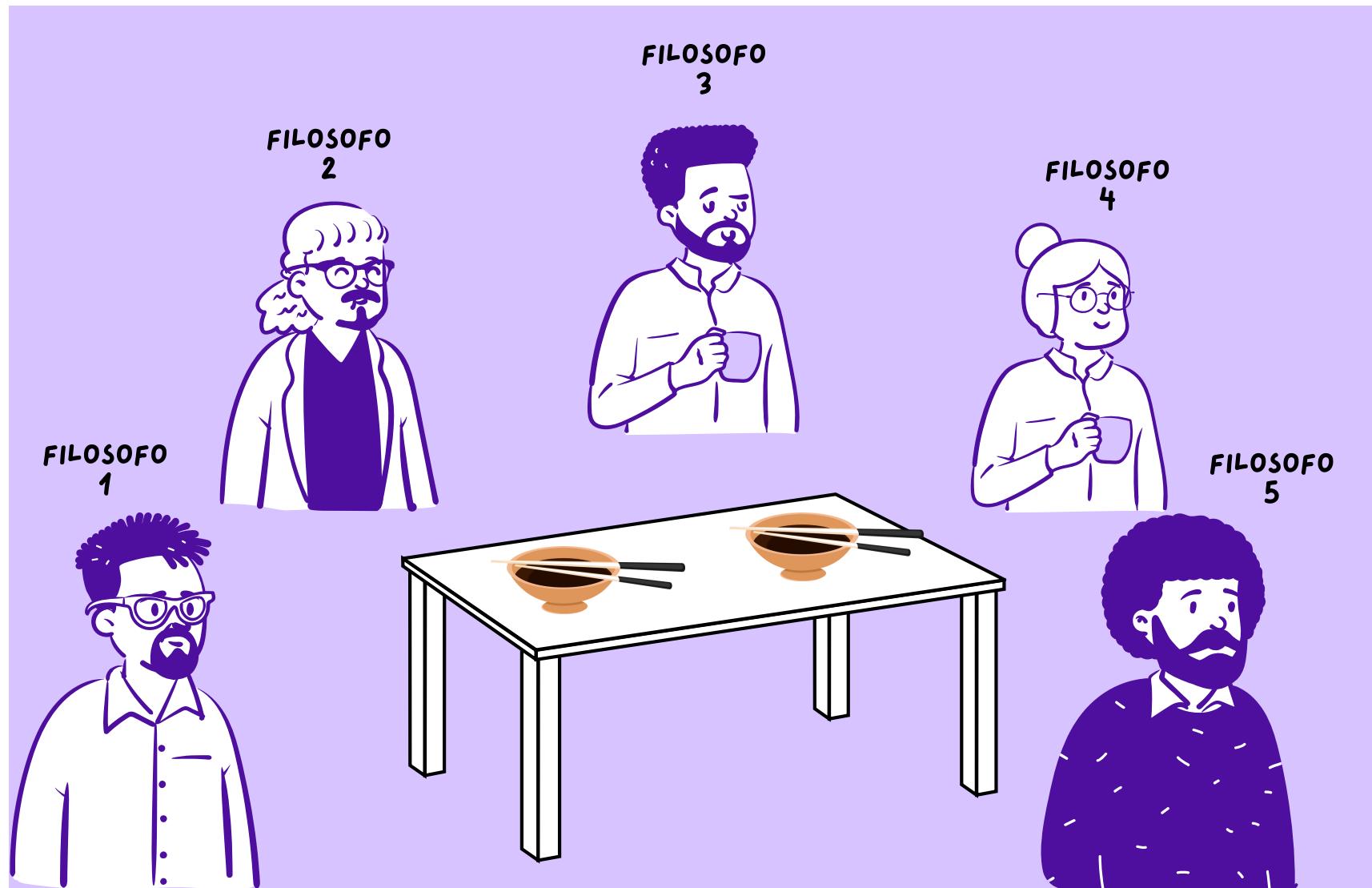


Todo gracias a super semáforo!!!!

Perfecto, esto no nos tendrá tan presionados y así seremos mas eficientes, es un ganar ganar



# Problema de la cena de filósofos



Nuestro queridos amigos solo contaran con 5 palillos, ¿como lograran comer?

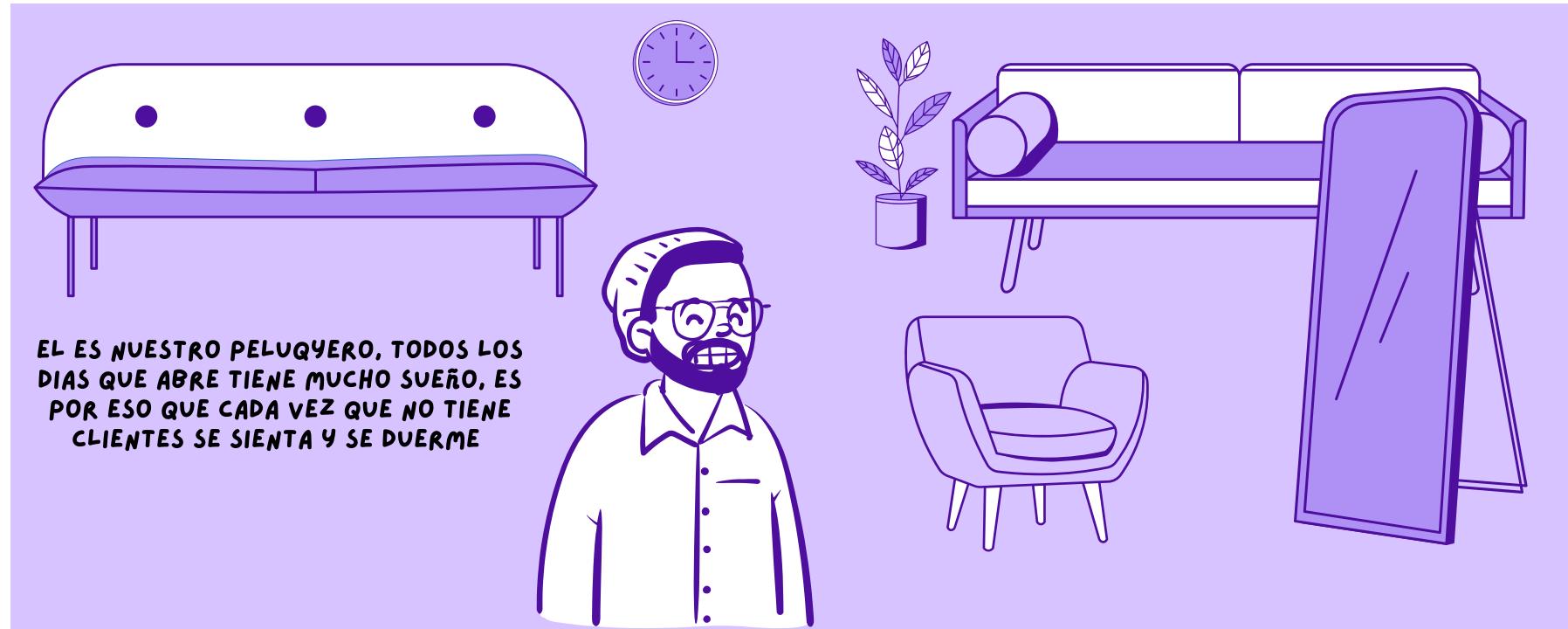


Cuando nuestros 5 colegas filosofos se ponen a pensar no hacen mas que eso, no se preocupan por nada mas.



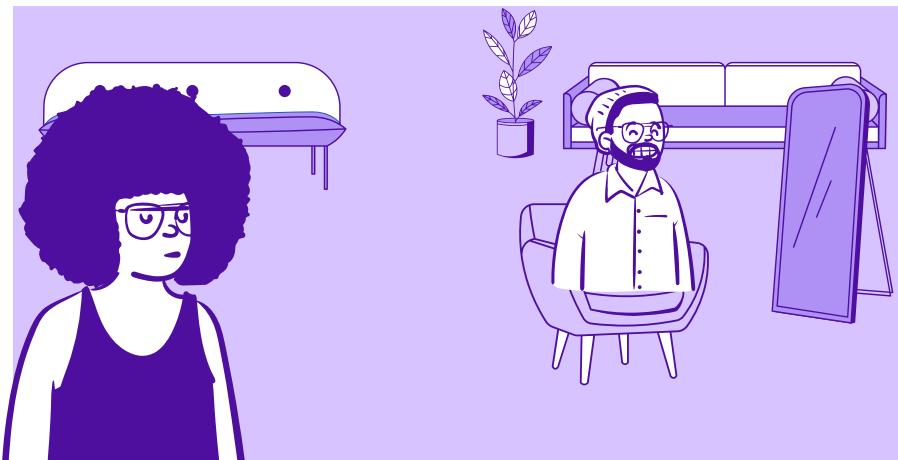


# Problema del peluquero dormido



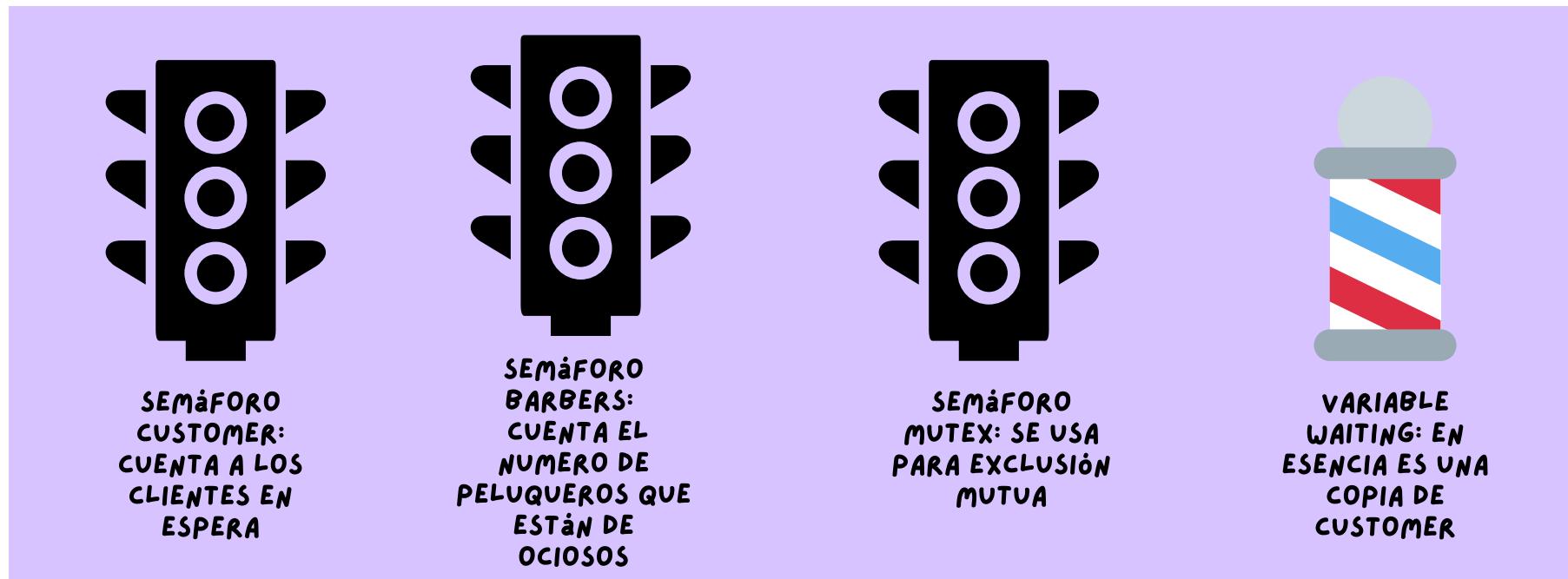
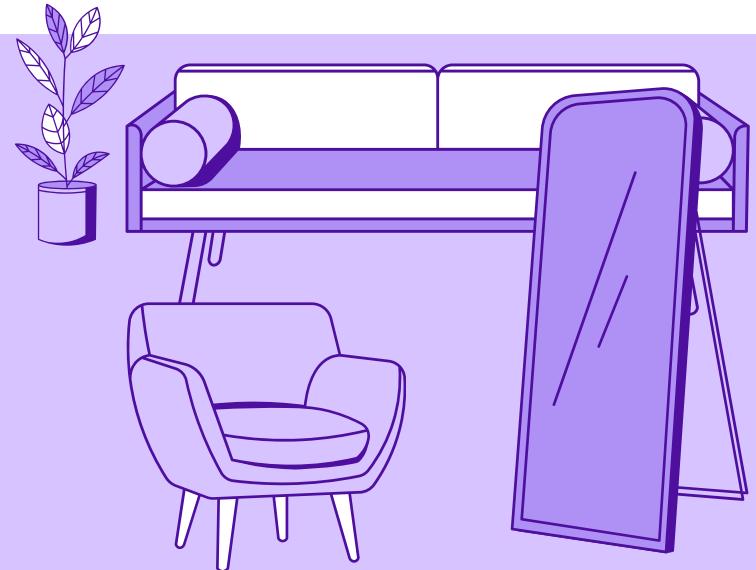
EL ES NUESTRO PELUQUERO, TODOS LOS DIAS QUE ABRE TIENE MUCHO SUEÑO, ES POR ESO QUE CADA VEZ QUE NO TIENE CLIENTES SE SIENTA Y SE DUERME

COMO PODEMOS VER NUESTRO PELUQUERO ESTA DORMIDO, CADA VEZ QUE LLEGA UN CLIENTE TIENE QUE ESTARLO DESPERTANDO, SE LE PERDONA PORQUE ES MUY BUENO EN LO QUE HACE

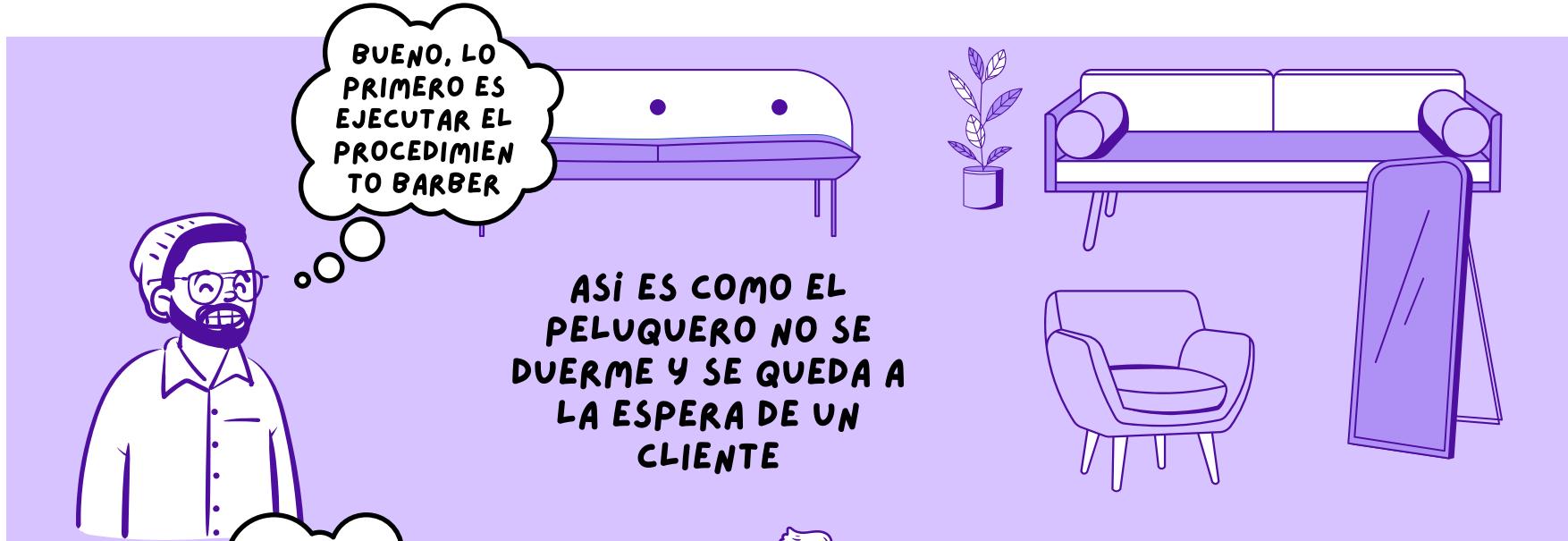




# la solución al problema



# un nuevo día implementando las soluciones





**FIN**