Universidad Nacional Autónoma de México

Facultad de Ingeniería

Sistemas Operativos

Prof. Gunnar Eyal Wolf Iszaevich

Grupo 06. Semestre 2024-1

PROYECTO #2

“Una situación cotidiana paralelizable”

Galván Zúñiga Adrián Ricardo

316034616

Ingeniería en Computación

Identificación del problema

A lo largo de nuestras vidas, todas las personas nos hemos visto alguna vez en una situación de cierta *emergencia*; requerimos hacer nuestras necesidades y no queda otra opción más que entrar a un temido *baño público.*

Ahora bien, existe una situación bastante *particular* en la que nos vemos envueltos las personas de sexo masculino al momento de entrar a uno de estos lugares, la cual se describe la llamada *urinal etiquette*, o *etiqueta de urinal*.

En un artículo publicado por Laura Abernethy en Metro UK (2019) se describen varias de las reglas *no escritas* que componen esta etiqueta, sin embargo, nosotros nos enfocaremos en una que enuncia algo como esto:

Nunca elijas el urinal próximo a alguna otra persona, y en caso de que no haya uno disponible que cumpla con esto, deberás utilizar un cubículo o esperar.

En esta implementación hacemos de cuenta que los usuarios solo tienen dos opciones: elegir un mingitorio que se encuentre *dentro de las reglas*, o bien, esperar a que alguien libere uno.

Naturaleza del problema

Entendemos la situación y de qué manera se presenta, pero ¿por qué tener reglas?, ¿Dónde pueden verse las consecuencias nocivas de la concurrencia?

Bueno, el no respetar esta restricción puede ser interpretada como una conducta *extraña,* por parte de quién lo haga; ¿por cuestiones de privacidad?, ¿de respeto al espacio vital?, ¿es algo meramente tradicional/cultural?, independientemente de la razón original, creo que la mayoría nos sentiríamos algo incómodos si nos vemos envueltos en una situación así:

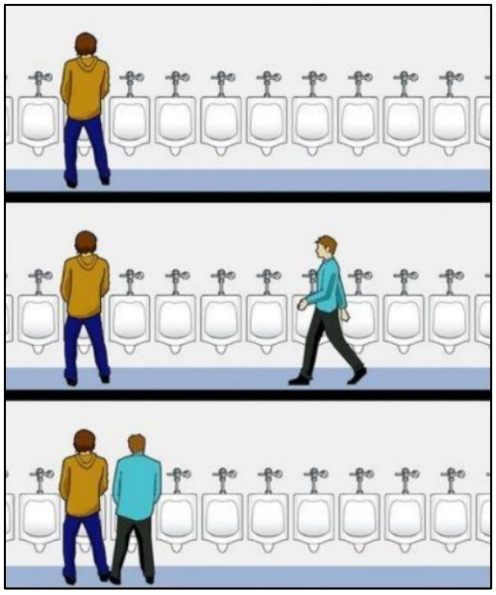


Imagen de imgflip

Asimismo surge la pregunta: ¿Qué eventos pueden ocurrir que queramos controlar?

Me parece que la mejor manera de responder esta pregunta es mediante otra ilustración (encontrada por ahí), que describe perfectamente una situación *de pesadilla.*

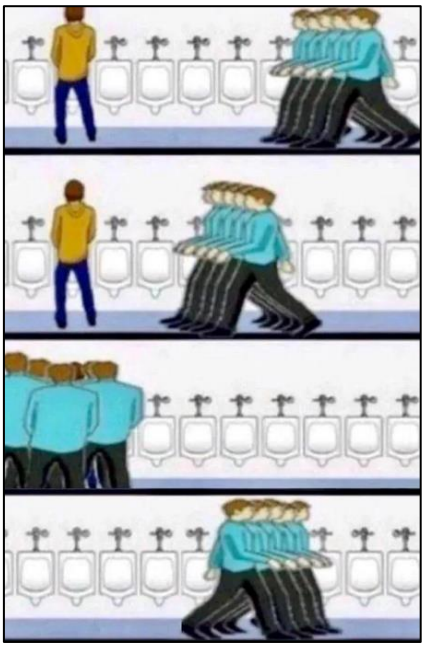


Imagen de Reddit

Por tanto, llevaremos este ejemplo de la vida cotidiana a una implementación en programación concurrente.

Herramientas empleadas

Se seleccionaron las siguientes herramientas para desarrollar y probar la solución:

Sistema operativo *Windows 10 Pro,* versión 10.0.19045.
Entorno de desarrollo *Visual Studio Code,* versión 1.84.0 (codificación y ejecución).
Lenguaje de programación *Python3,* versión 3.11.4.
Con los módulos:
Colorama. Para producir salida de texto en color.
Random. Para obtener números aleatorios empleados en tiempos de espera.
Threading. Para paralelizar procesos mediante hilos y semáforos.
Time. Para producir tiempos de espera.

Implementación

Las estructuras empleadas son las siguientes:

Semáforos para representar cada uno de los mingitorios, (agrupados en *mings*).
Tres *Mutex* sencillos:
Uno se utiliza para un contador, el cual permite un mejor seguimiento de la ejecución (*mutexContador*).

Otro se utiliza para asegurar que los hilos modifican e imprimen uno a la vez el esquema visual (*mutexImprime*).
El último se usa para evitar conflicto con varios hilos intentando acceder a los mismos semáforos

El estado compartido es el *arreglo* de mingitorios. Todos los usuarios (hilos) comparten todos los mingitorios (semáforos), pero solo uno puede usar uno de ellos a la vez, además de que no pueden usar aquellos que estén próximos a uno en uso.

Descripción Algorítmica

Lo primero que se hace es crear a los usuarios (lanzar a los hilos), la cantidad que defina la variable *numUsuarios.*

Después, cada usuario (hilo) lanza su primer mensaje: "*Existo*", para notificar su existencia (inicialización del hilo), y entra al lugar donde se desarrollan los eventos mediante la función *entra.* A partir de aquí, los hilos que hayan llegado continúan volviendo de manera indefinida, hasta que el programa se interrumpa manualmente.

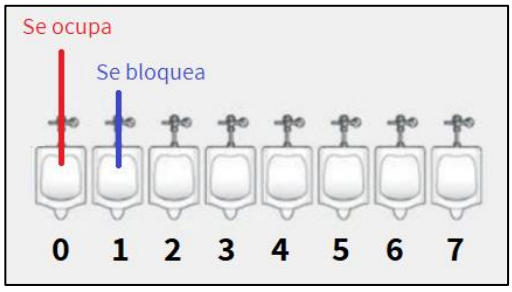
Una vez que el usuario llega, comienza a buscar un mingitorio disponible, y selecciona uno al azar para intentar utilizarlo. Esto es, se genera un número de manera aleatoria mediante *random*, y se asigna este valor a la variable *cual*, que definirá el semáforo que intentará adquirir el hilo.

A partir de aquí se presentan dos casos:

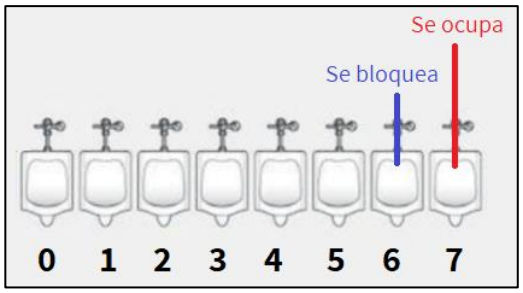
Si el mingitorio seleccionado no se encuentra disponible o *no es válido,* el usuario elige otro y vuelve a intentar, es decir, si el semáforo que se intenta adquirir no está disponible, se genera otro número y el proceso se repite.

Si el mingitorio seleccionado se encuentra disponible, el usuario lo ocupa, bloqueando el o los que estén próximos. Esto es, el hilo adquiere el semáforo, y mediante condicionales se especifica la posición del lugar ocupado, esto aprovechando que las posiciones se encuentran en una lista, por lo que podemos utilizar los índices de la lista para representar las posiciones:

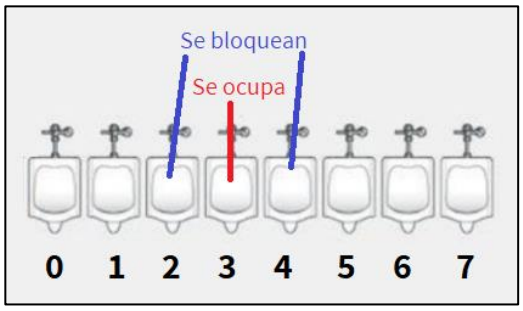
Si la posición *cual* ocupada es la 0, se trata de la posición en el extremo izquierdo, por lo que únicamente tenemos que bloquear la posición *cual+1*; la posición de la derecha:



Si la posición *cual* ocupada es la 7, se trata de la posición en el extremo derecho, por lo que únicamente tenemos que bloquear la posición *cual-1*; la posición de la izquierda:



Si la posición *cual* ocupada es intermedia (1-6), tenemos que bloquear las posiciones *cual-1* y *cual+1*; las posiciones próximas a la ocupada:



La forma de hacer esto es haciendo que el hilo que entre en esta región adquiera los semáforos correspondientes a las posiciones que debe ocupar, y a las que se deben bloquear.

Al ocupar y bloquear las posiciones, el hilo adquiere otro semáforo *mutex* para poder modificar los espacios *de golpe.* Esto es, que cada hilo realice la ocupación y los bloqueos antes de que otro hilo interrumpa este proceso y acceda a los mismos semáforos.

Esta adición del *mutex* es una corrección implementada posteriormente, pues noté que el programa se quedaba esperando infinitamente sin razón alguna (aparentemente). Deduje que este problema se debía a que un hilo podía ocupar una posición, y antes de bloquear las demás, otro hilo podía ocupar en ese momento una posición que debía ya estar bloqueada, generando un conflicto de lógica.

Cuando el hilo termina de manejar sus semáforos correspondientes, libera el mutex y continúa.

La siguiente acción es que el hilo adquiere otro *mutex* para modificar la variable *contador* la cual lleva un registro de las “fases” del programa; se produce cada que un hilo ocupa un lugar.

Después, se actualiza el esquema visual que representa a los mingitorios.
El esquema consta de 8 caracteres, el guión '-' representando un espacio vacío, y la letra 'X' representando un usuario ocupando ese espacio.

El objetivo entonces es que no se pueda dar el caso donde dos letras 'X' se encuentren juntas.

Después que el hilo modifica el esquema adquiriendo primero un *mutex* para que cada hilo lo haga después de su participación. Luego manda a llamar al método *mostrarEsquema* para imprimirlo, y libera el *mutex* para imprimir el esquema.

Después, cada usuario se toma su tiempo para hacer sus necesidades… es decir, cada hilo se detiene durante pequeño tiempo aleatorio antes de continuar con la ejecución.

Una vez que termina el tiempo de espera, el usuario notifica mediante un texto que ha terminado, y procede a llamar la función *desocupaLugar.*
En esta función, el hilo sigue el mismo procedimiento que realiza al ocupar un lugar, sólo que de manera opuesta:

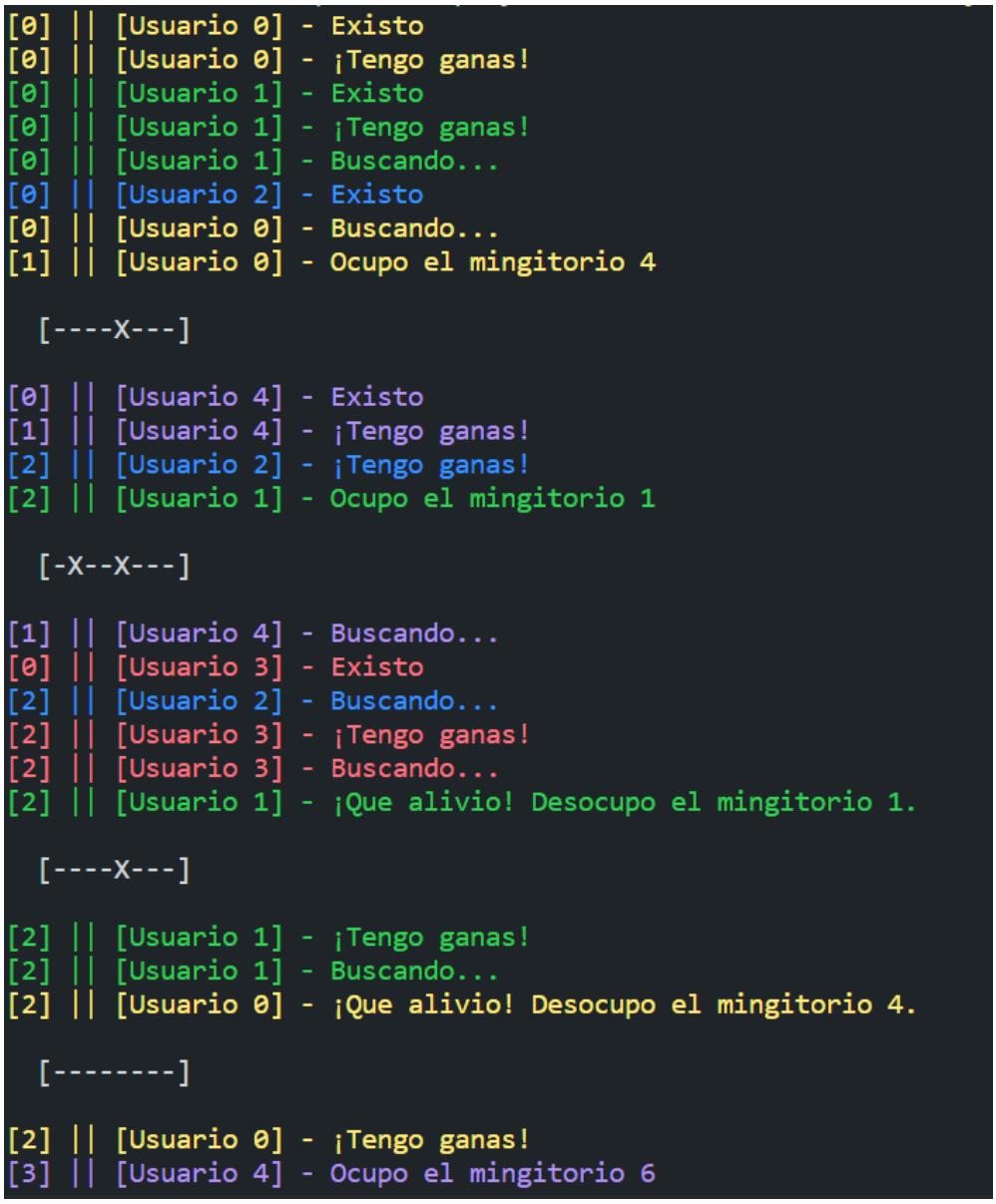
Libera el semáforo correspondiente al lugar que ocupó; toma el *mutex* para modificar e imprimir el esquema; suelta el *mutex;* verifica si la posición ocupada era intermedia o de los extremos, y dependiendo del caso llama a la función *desbloqueaLugar*, donde se liberan los semáforos de los lugares bloqueados.

Con esto, el hilo concluye su paso por el *temible* baño público.

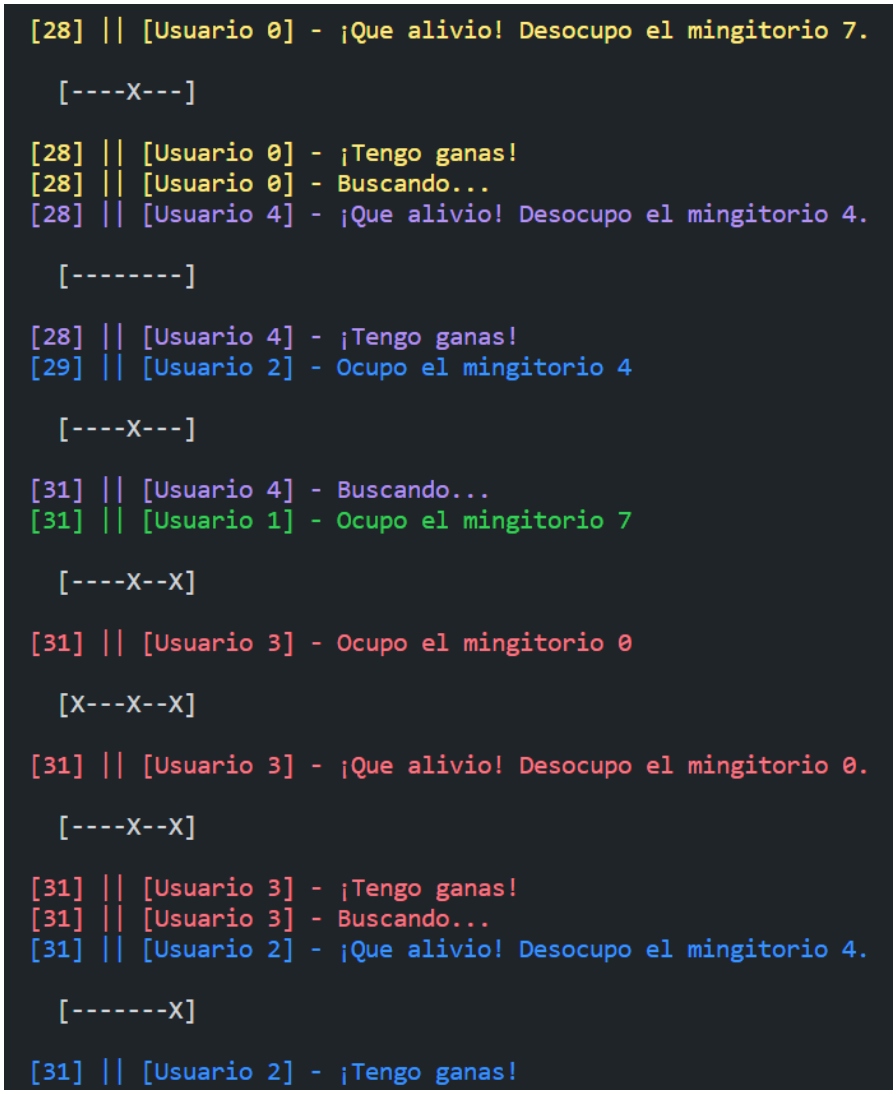
Pruebas exitosas de ejecución

Generamos una primera prueba corriendo el programa, y verificamos que las condiciones se cumplen satisfactoriamente (el programa se detiene manualmente después de *contador = 5*):

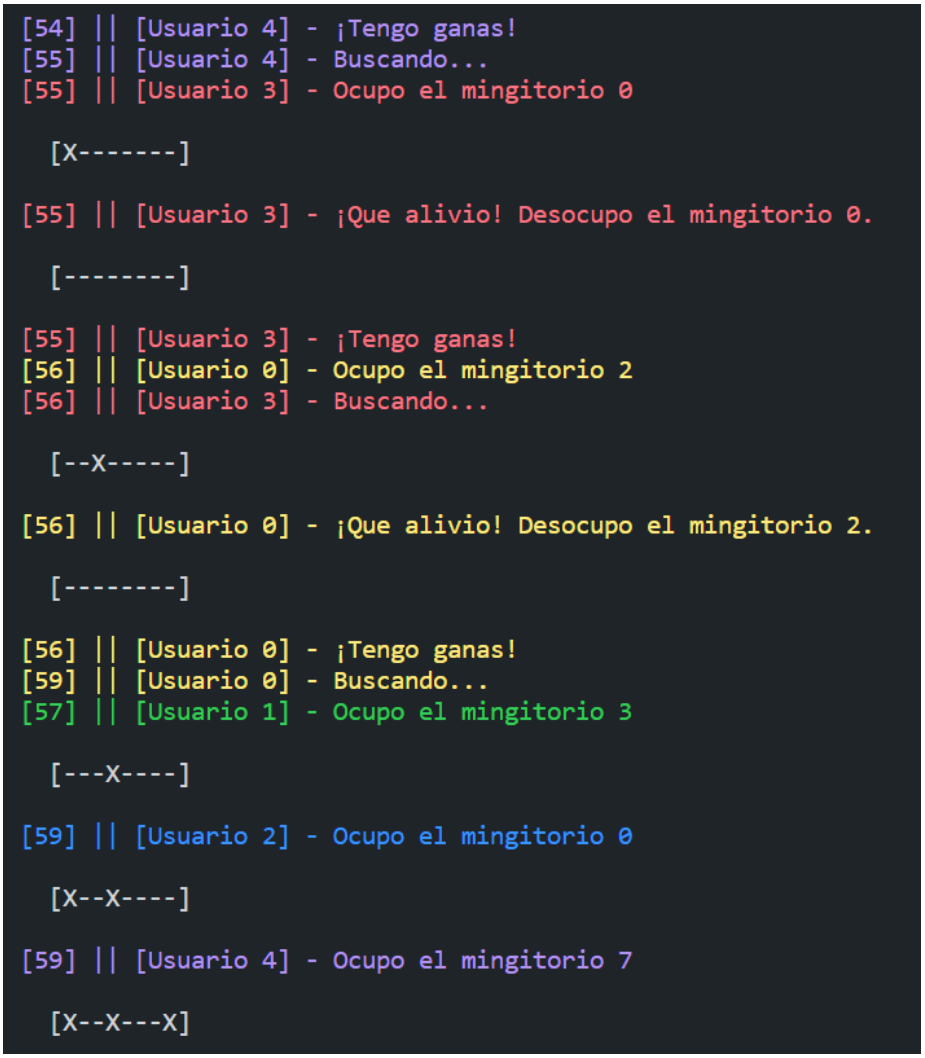
Generando una segunda ejecución, comprobamos que el programa una vez más realiza el procedimiento adecuado:



Y continúa...


[28] || [Usuario 0] - ¡Que alivio! Desocupo el mingitorio 7.
[----X---]
[28] || [Usuario 0] - ¡Tengo ganas!
[28] || [Usuario 0] - Buscando...
[28] || [Usuario 4] - ¡Que alivio! Desocupo el mingitorio 4.
[--------]
[28] || [Usuario 4] - ¡Tengo ganas!
[29] || [Usuario 2] - Ocupo el mingitorio 4
[----X---]
[31] || [Usuario 4] - Buscando...
[31] || [Usuario 1] - Ocupo el mingitorio 7
[----X--X]
[31] || [Usuario 3] - Ocupo el mingitorio 0
[X---X--X]
[31] || [Usuario 3] - ¡Que alivio! Desocupo el mingitorio 0.
[----X--X]
[31] || [Usuario 3] - ¡Tengo ganas!
[31] || [Usuario 3] - Buscando...
[31] || [Usuario 2] - ¡Que alivio! Desocupo el mingitorio 4.
[-------X]
[31] || [Usuario 2] - ¡Tengo ganas!

Y continúa...



Indefinidamente...