Auxiliar 4: Semáforos

CC4302 - Sistemas Operativos

José Astorga

Semáforos

- Herramienta de sincronización. Funciona como sistema de tickets: un semáforo contiene tickets y los threads pueden solicitar y depositar tickets. En el caso que un thread solicite un ticket y no haya tickets disponibles, deberá esperar.
- Se pueden usar para garantizar la exclusión mutua (como mutex) y para esperar (como condición).

Semáforos

- Inicializar un semáforo:

```
void sem_init(sem_t *sem, int pshared, unsigned int val);
```

  - `sem_t *sem`: Puntero a semáforo a inicializar.
  - `int pshared`: Flag para indicar si semáforo será compartido entre threads (pshared = 0) o entre procesos (pshared = 1).
  - `unsigned int val`: Cantidad de fichas iniciales.

Semáforos

- Inicializar un semáforo:

```
void sem_init(sem_t *sem, int pshared, unsigned int val);
```

```
sem_t sem;
sem_init(&sem, 0, 1);
```

Semáforos

- Destruir el semáforo:

```
void sem_destroy(sem_t *sem);
```

  - `sem_t *sem`: Puntero a semáforo a destruir.

Semáforos

- Extraer una ficha:

```
void sem_wait(sem_t *sem);
```

- Depositar una ficha:

```
void sem_post(sem_t *sem);
```

Semáforos

- Los semáforos se pueden usar como mutex o como condiciones.
- Para usar como mutex se inicializa un semáforo con un único ticket:
  - Para entrar a la zona crítica un thread solicita el ticket y al salir lo deposita.

```
sem_t sem;
sem_init(&sem,0, 1);
```

```
// Entra zona crítica
sem_wait(&sem);
...
// Salir zona crítica
sem_post(&sem);
```

Semáforos

- Para usar como condición se inicializa el semáforo sin tickets:
  - El thread que quiere esperar debe pedir un ticket a este semáforo y dado que no tiene tickets se quedará esperando.
  - El thread que despierta a los demás debe depositar tickets para despertar.

```
sem_t wait;
sem_init(&wait,0, 0);

sem_wait(&wait); //Esperamos
```

```
// Otro thread debe despertarlo
// depositando un ticket

sem_post(&wait);
```

Semáforos

- Semáforo como condición:
  - Generalmente se usará el semáforo como condición dentro de una zona crítica, en ese caso se debe tener mucho cuidado y liberar el mutex de la zona crítica antes de ponerse a esperar ya que si no lo hacemos, ningún thread podrá entrar a la zona crítica.

```
sem_wait(&sem); // Entra zona critica
if (se_debe_esperar){
      sem_post(&sem); // Liberar el mutex
      sem_wait(&wait); // Esperar
      sem_wait(&sem); // Tomar la zona crítica si es necesario
}
sem_post(&sem); // Salir zona crítica
```

P1. Problema del baño compartido

Un estadio posee un único baño que debe ser compartido por hinchas rojos y azules. El baño es amplio y admite un número ilimitado de personas. El problema consiste en evitar que los hinchas rojos se encuentren con los hinchas azules dentro del baño.

Los hinchas rojos solicitan entrar al baño invocando `entrar(ROJO)` y notifican su salida con `salir(ROJO)`, mientras que los hinchas azules invocan `entrar(AZUL)` y `salir(AZUL)`.

Se plantea la siguiente solución incorrecta para el problema:

```
enum { ROJO = 0, AZUL = 1 };

int cantidad[2] = {0, 0};
int mutex = 0; // este mutex representa el acceso al baño.
// El equipo que lo tiene es el que está adentro.

void entrar(int color){
   if (cantidad[color] == 0){
        while (mutex)
              ;
        mutex = 1;
   }
   cantidad[color]++;
}

void salir(int color){
   cantidad[color]--;
   if (cantidad[color] == 0){
        mutex = 0;
   }
}
```

Se plantea la siguiente solución incorrecta para el problema:

```
enum { ROJO = 0, AZUL = 1 };

int cantidad[2] = {0, 0};
 // este mutex representa el acceso al baño.
 // El equipo que lo tiene es el que está adentro.
int mutex = 0;

void entrar(int color) {
    if (cantidad[color] == 0) {
        while(mutex)
            ;
        mutex = 1;
    }
    cantidad[color]++;
}

void salir(int color) {
    cantidad[color]--;
    if (cantidad[color] == 0) {
        mutex = 0;
    }
}
```

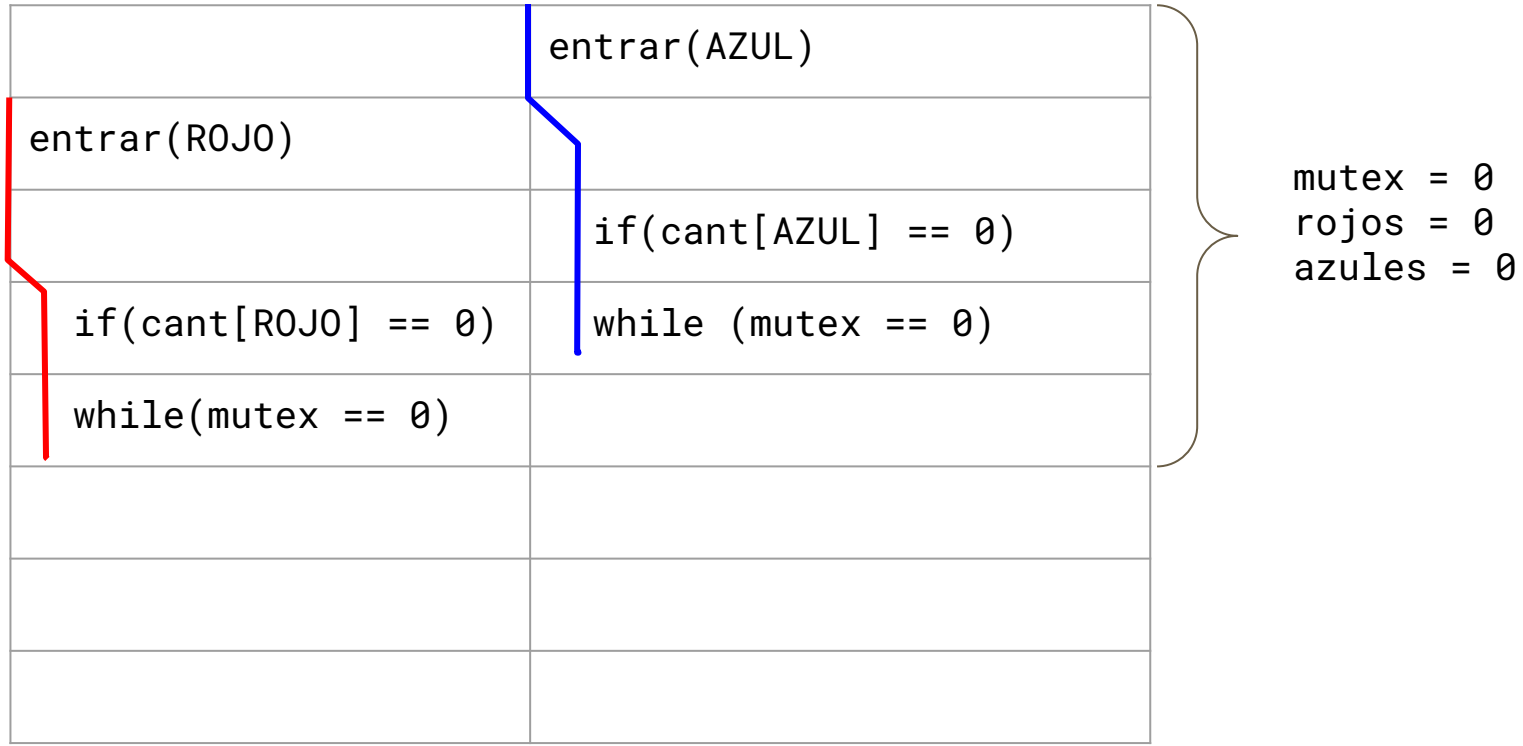
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.

P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.

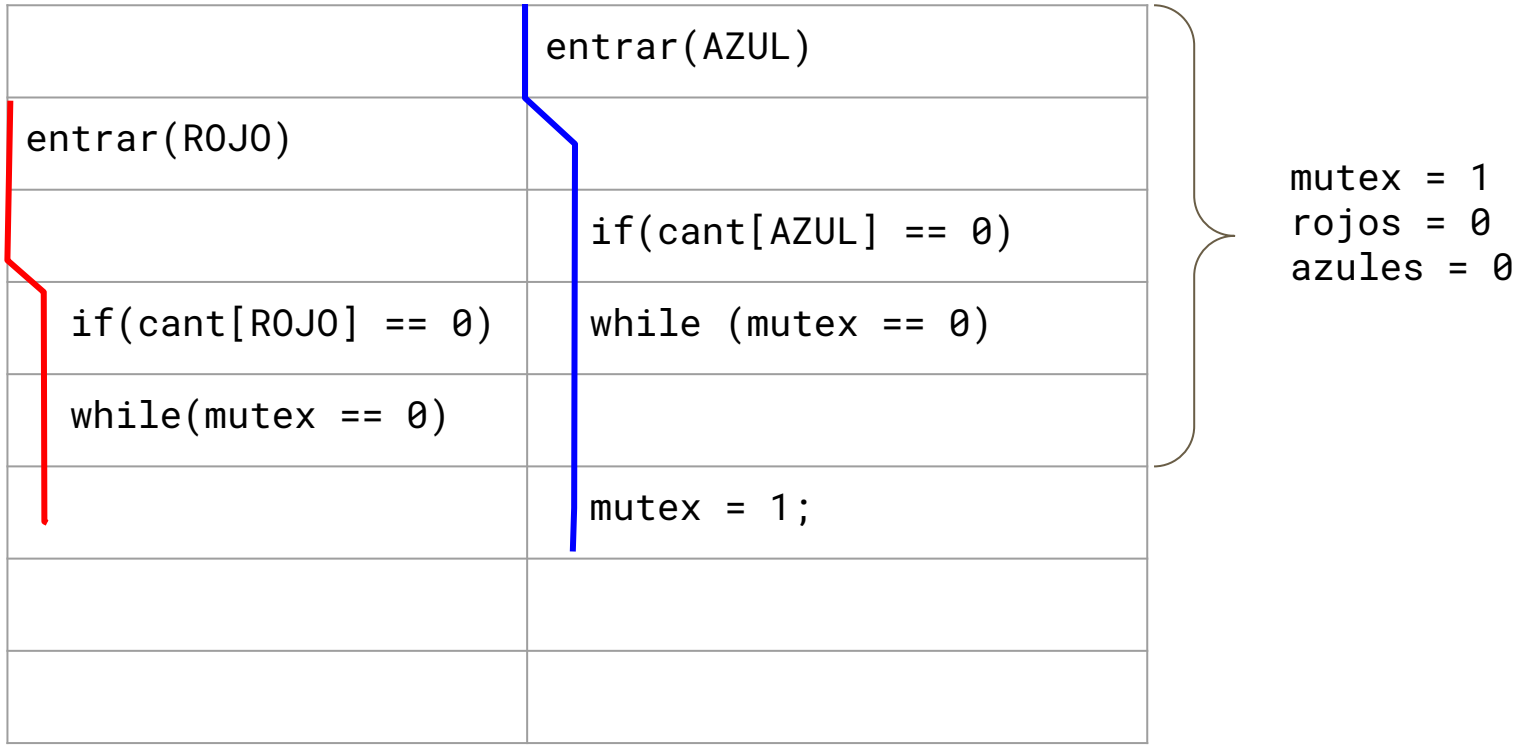
	`entrar(AZUL)`
`entrar(ROJO)`	

```
mutex = 0
rojos = 0
azules = 0
```

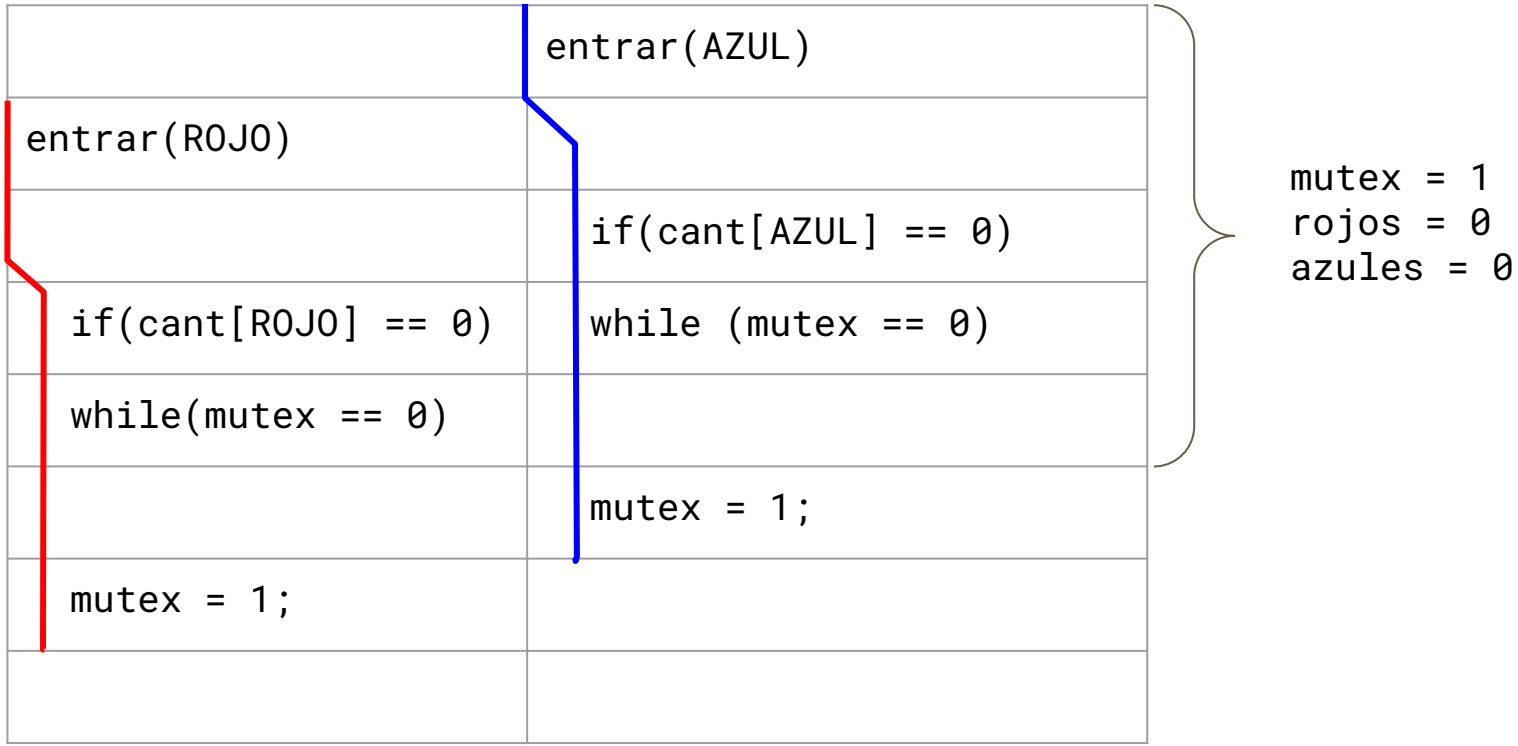
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



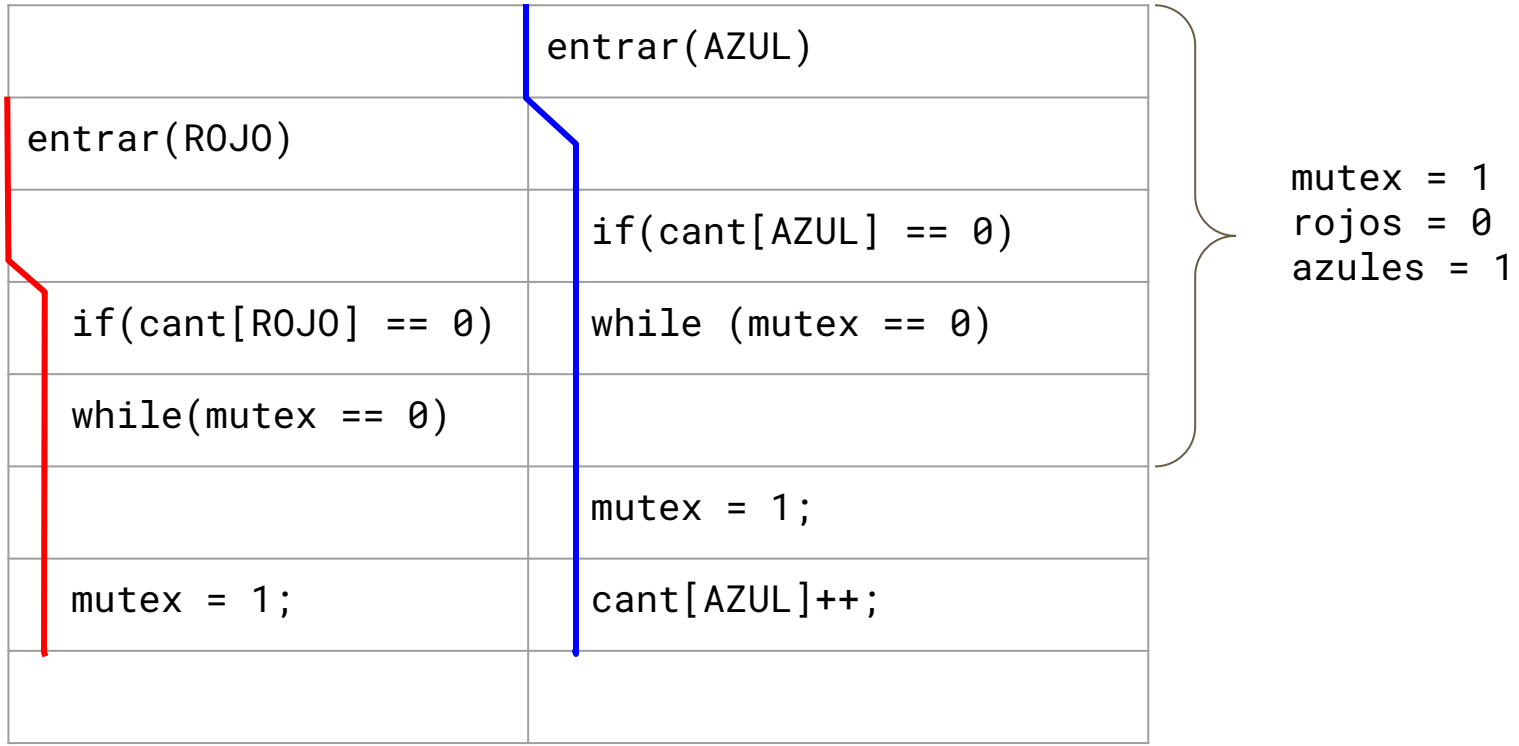
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



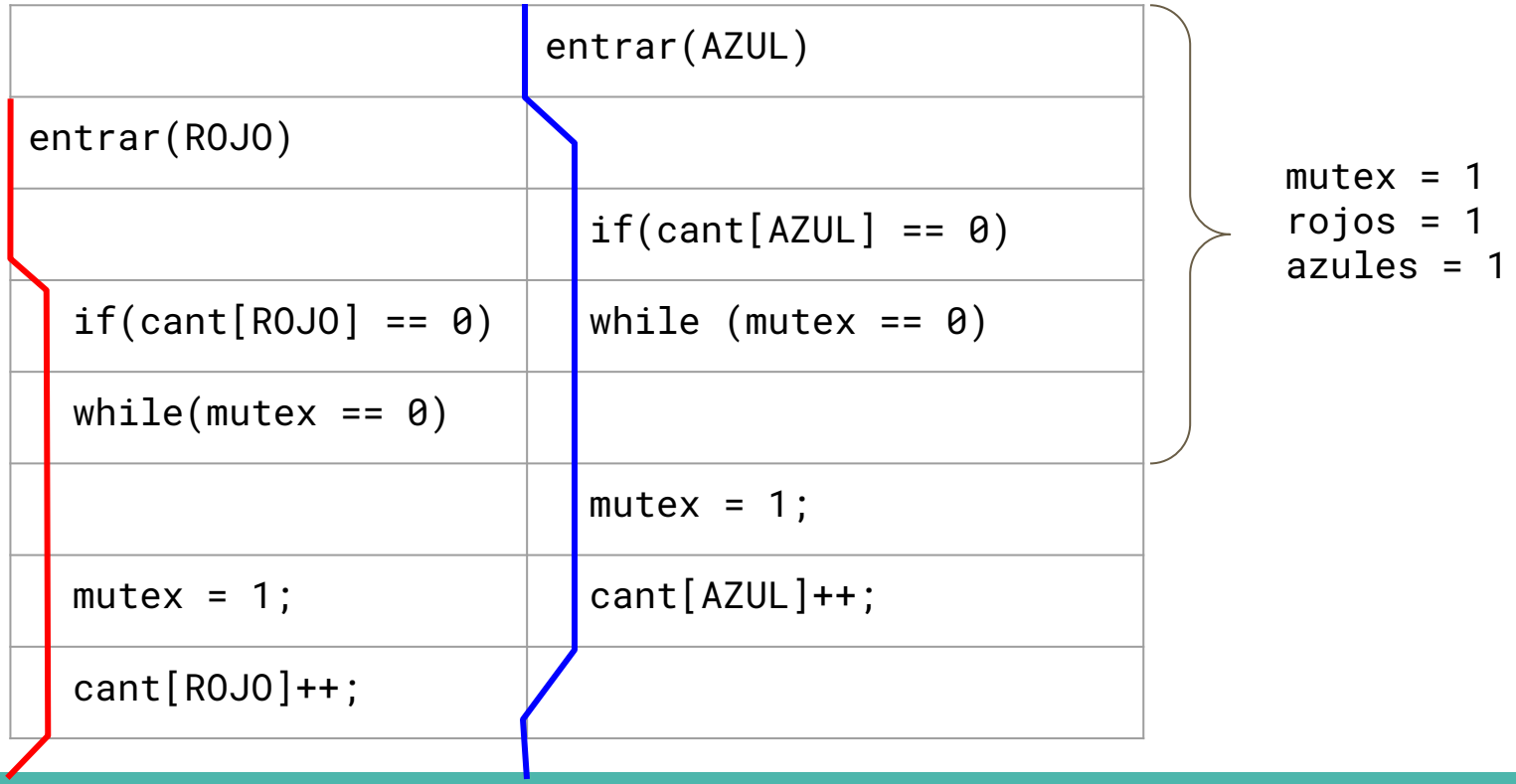
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



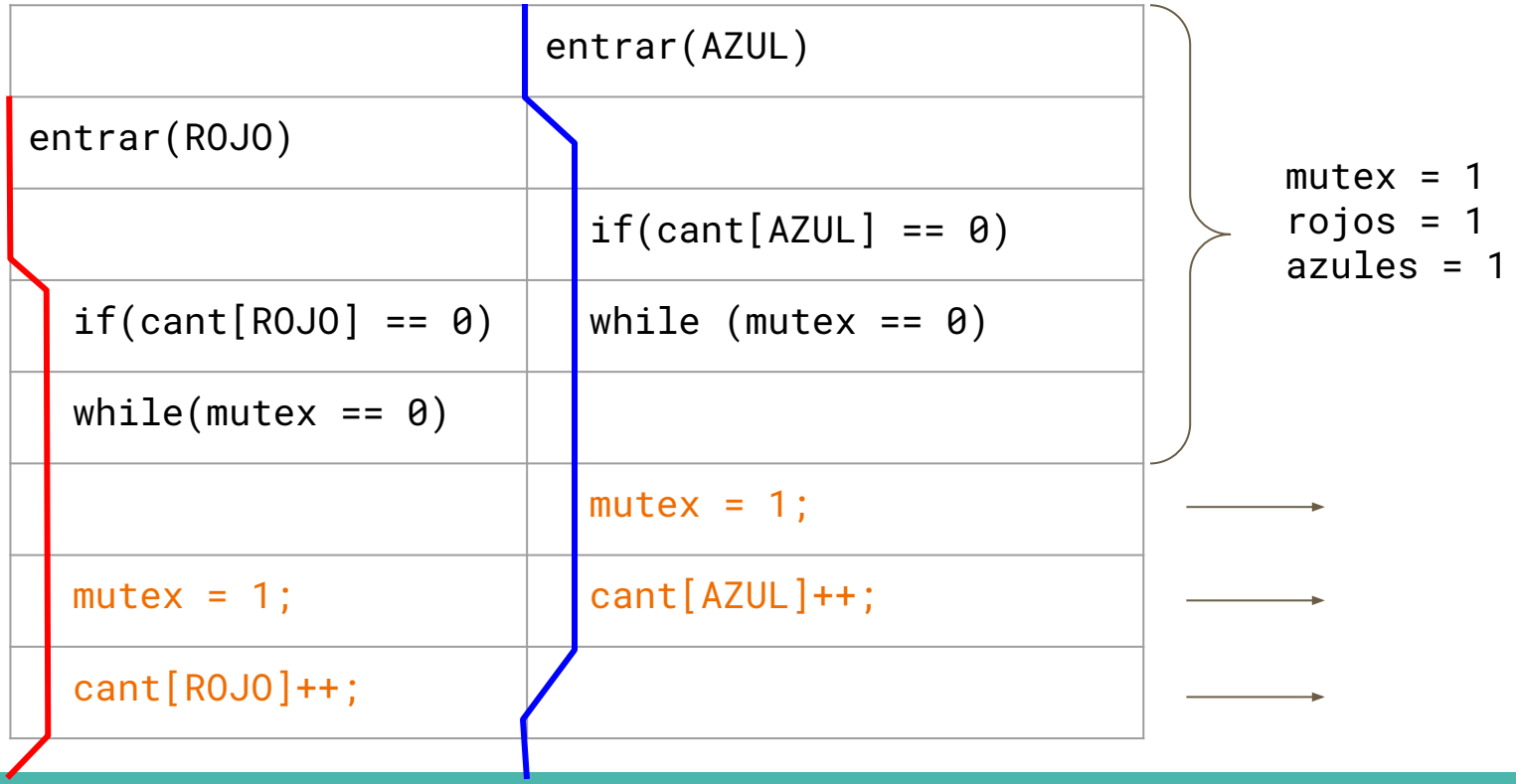
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



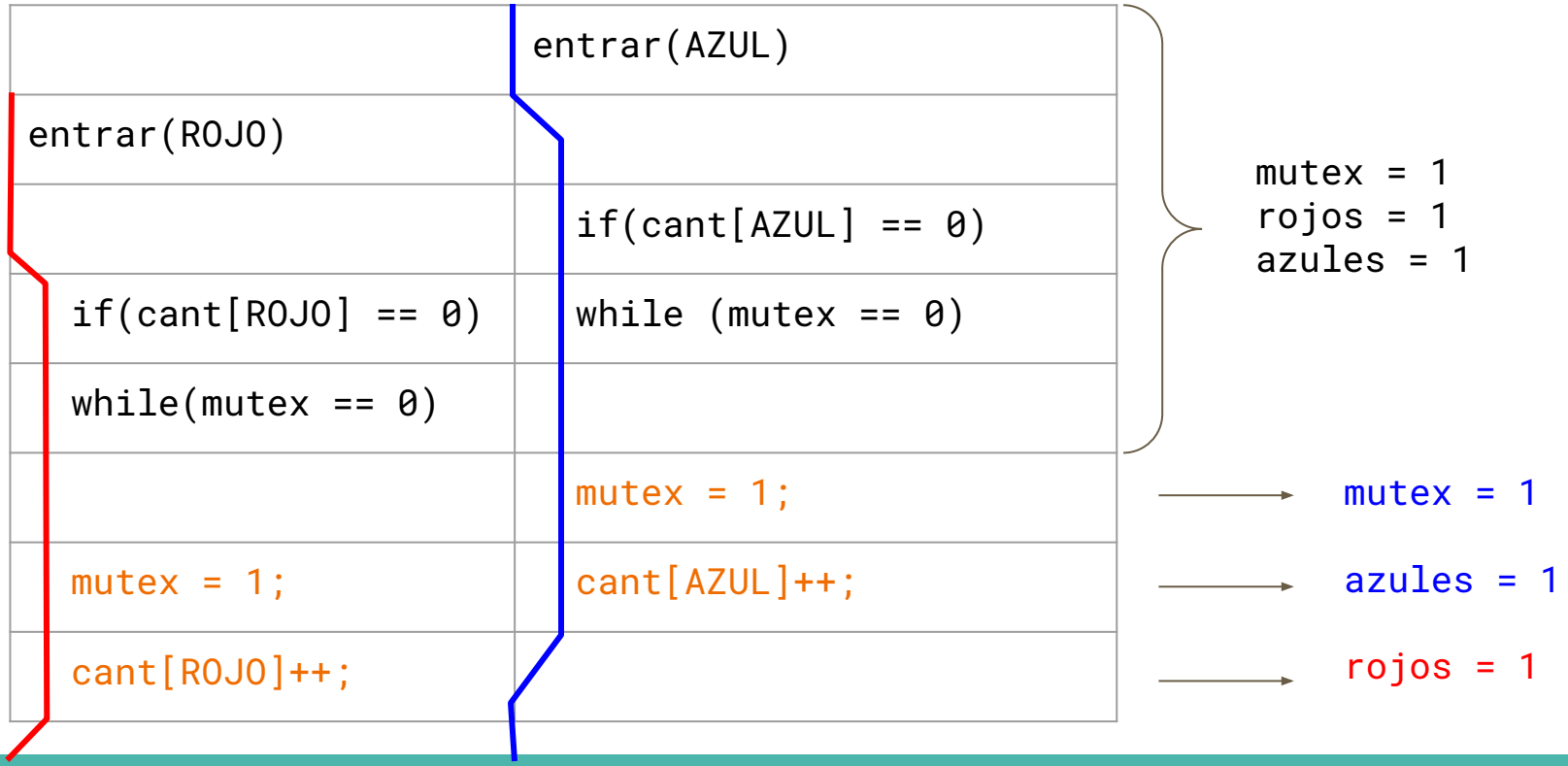
P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.



P1.a Muestre mediante un *diagrama de threads* que un hincha rojo puede entrar al baño cuando hay hinchas azules presentes.




U. de Chile
v/s
Colo Colo
lotto
lotto
CRISTAL
LA TERCERA
www.terra.cl

P1.b Corregir solución incorrecta

Escriba una solución correcta y eficiente para este problema utilizando 3 semáforos. No importa si en su solución algunos procesos sufren “hambruna”.

Hint: utilice la estructura de la solución incorrecta.

correcta: no hay data races.

eficiente: no hay *busy waiting*

P2 Baño compartido sin Hambruna

Considere ahora una solución en la que no se produzca hambruna. Para lograr esto es necesario que ningún hincha entre al baño mientras haya hinchas del otro equipo esperando. Luego, cuando sale el último hincha del baño, entran todos los hinchas del equipo contrario que estaban esperando. Por ejemplo, si hay dos hinchas del equipo rojo en el baño y un hincha azul en espera, el siguiente hincha rojo en llegar no podrá entrar hasta que haya entrado (y salido) el azul.

a. Se incluye una implementación incorrecta de esta solución (ver p3.c). Demuestra que esta solución es incorrecta confeccionando un diagrama de threads donde la exclusión mutua no se cumple.

* Esta parte fue adaptada de la tarea 1 del semestre 2021-1.

P2.a Solución Incorrecta

```
sem_t mutex;  // Un semáforo controla el acceso a la zona crítica.
sem_t sem[2]; // Un semáforo para la espera cada tipo de hincha.
int esperan[2] = {0, 0}; int adentro[2] = {0, 0};

void entrar(int color){
   // el oponente del equipo AZUL es el equipo ROJO y viceversa.
    int oponente = !color;
    sem_wait(&mutex);
   // Si hay hinchas del otro equipo en el baño o en la cola
   // se debe esperar.
    if (adentro[oponente] > 0 || esperan[oponente] > 0){
        esperan[color]++;
        sem_post(&mutex);
        sem_wait(&sem[color]); // se pone el thread en espera
        sem_wait(&mutex);
    }

    adentro[color]++; // entramos al baño
    sem_post(&mutex);
}
```

```
void salir(int color)
{
    int oponente = !color;

    sem_wait(&mutex);

    adentro[color]--; // salimos del baño

    if (adentro[color] == 0)
    {
        // Despertar a los oponentes poniendo tantos tickets
        // como son threads hay en espera
        for (int i = 0; i < esperan[oponente]; i++)
        {
            sem_post(&sem[oponente]);
        }
        esperan[oponente] = 0;
    }

    sem_wait(&mutex);
}
```

P2.a Solución Incorrecta

```c
// Esta es la solución incorrecta.

#include <semaphore.h>
enum { ROJO = 0, AZUL = 1};

sem_t mutex;     // sem_init(&mutex, 0, 1);
sem_t sem[2];    // sem_init(&sem[ROJO], 0, 1);
                 // sem_init(&sem[AZUL], 0, 1);
int esperan[2] = {0, 0};
int adentro[2] = {0, 0};

void entrar(int color) {
    // el oponente del equipo AZUL es el equipo ROJO y viceversa.
    int oponente = !color;

    sem_wait(&mutex);
    // Si hay hinchas del otro equipo en el baño o en la cola,
    // se debe esperar.
    if (adentro[oponente] > 0 || esperan[oponente] > 0) {
        esperan[color]++;
        sem_post(&mutex);
        sem_wait(&sem[color]); // se pone el thread en espera
        sem_wait(&mutex);
    }
    adentro[color]++;
    sem_post(&mutex);
}
```

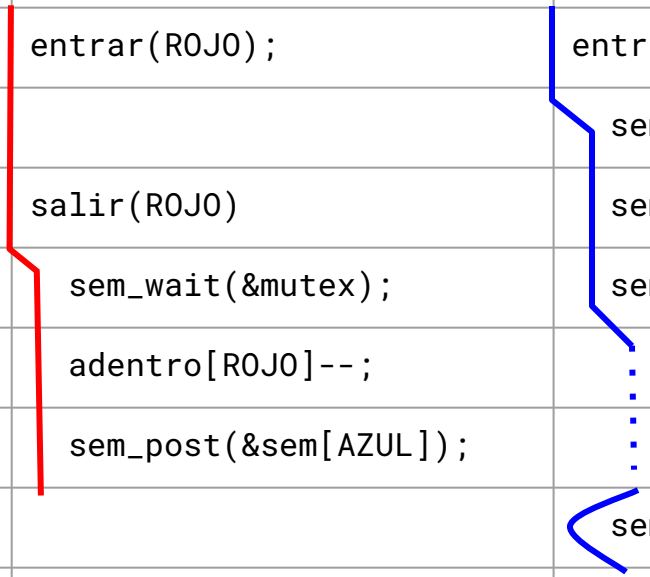
```c
void salir(int color) {
    int oponente = !color;

    sem_wait(&mutex);

    adentro[color]--;
    if (adentro[color] == 0) {
        for (int i = 0; i < esperan[oponente]; i++) {
            sem_post(&sem[oponente]);
        }
        esperan[oponente] = 0;
    }

    sem_post(&mutex);
}
```

t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);		
1			
2	salir(ROJO)		
3			
4			
5			
6			
7			
8			
9			
10			

t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3		sem_wait(&sem[AZUL]);	
4			
5			
6			
7			
8			
9			
10			

t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	
4	adentro[ROJO]--;		
5	sem_post(&sem[AZUL]);		
6			
7			
8			
9			
10			

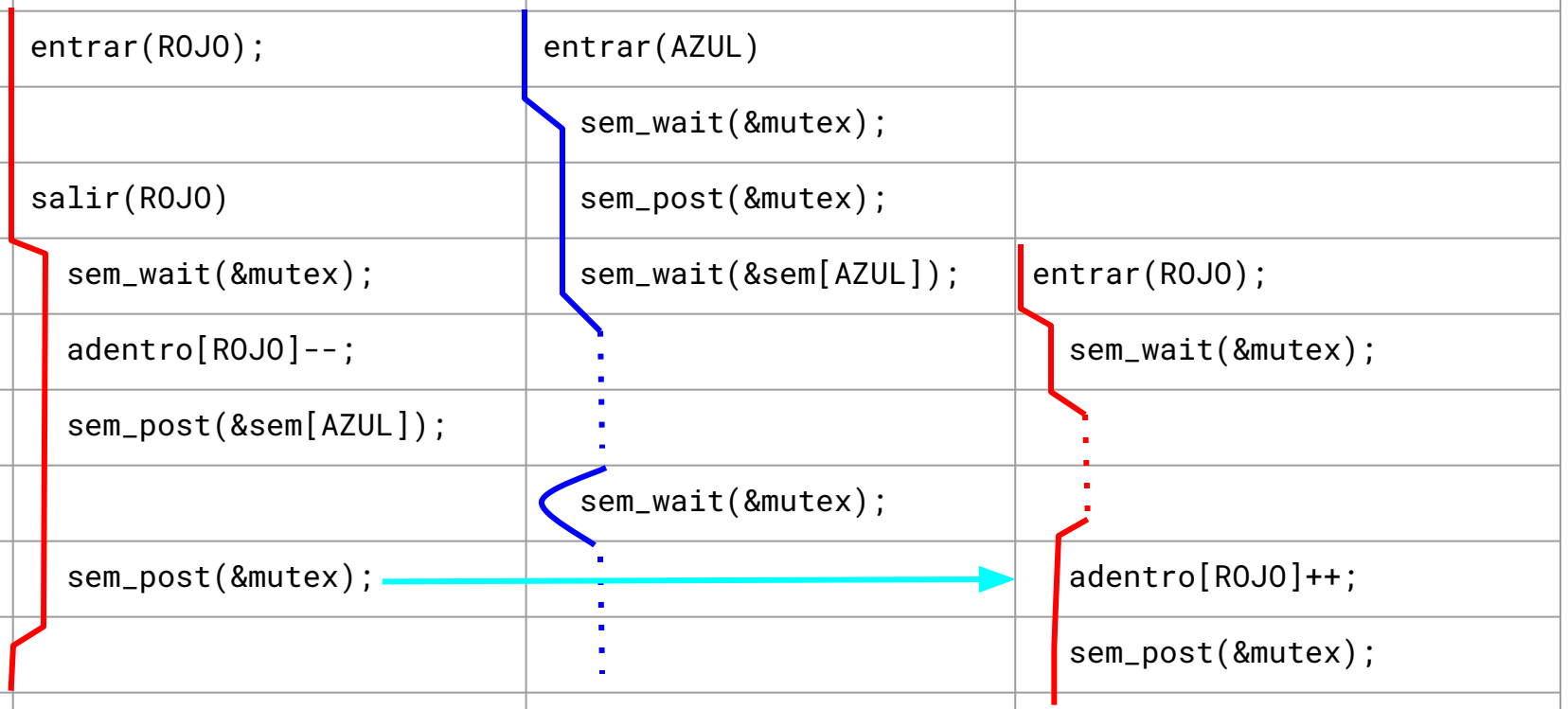
t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	
4	adentro[ROJO]--;		
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7			
8			
9			
10			



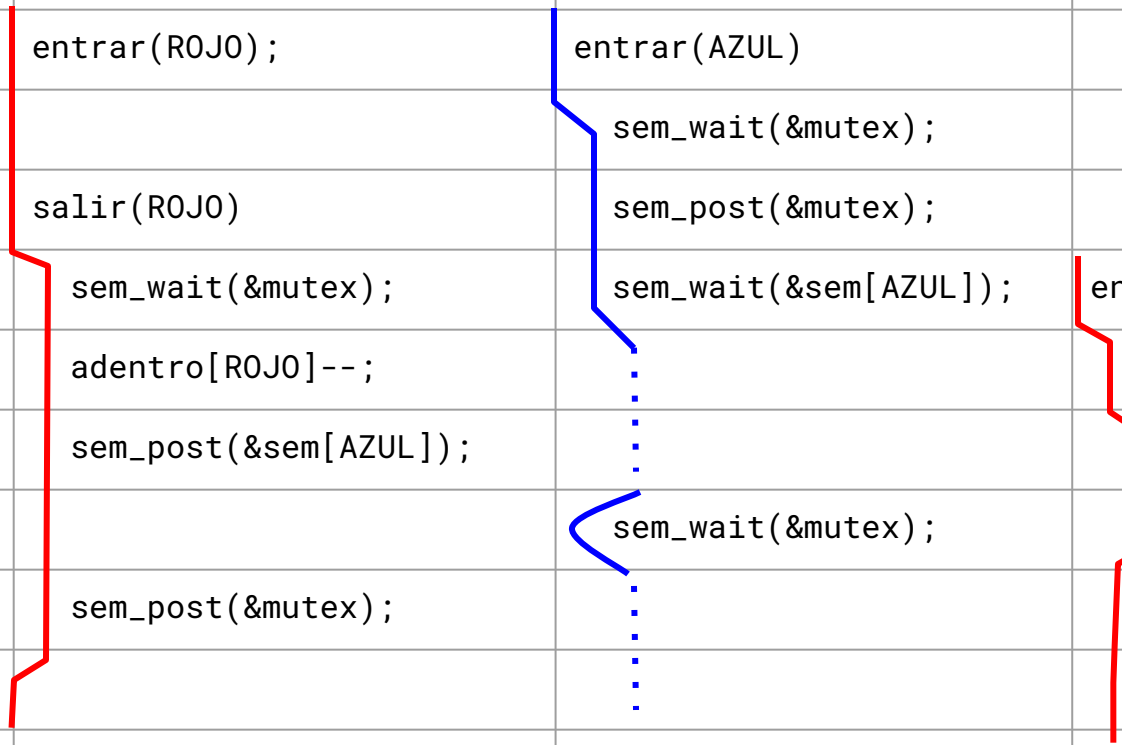
t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	entrar(ROJO);
4	adentro[ROJO]--;		sem_wait(&mutex);
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7			
8			
9			
10			

t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	entrar(ROJO);
4	adentro[ROJO]--;		sem_wait(&mutex);
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7	sem_post(&mutex);		
8			
9			
10			

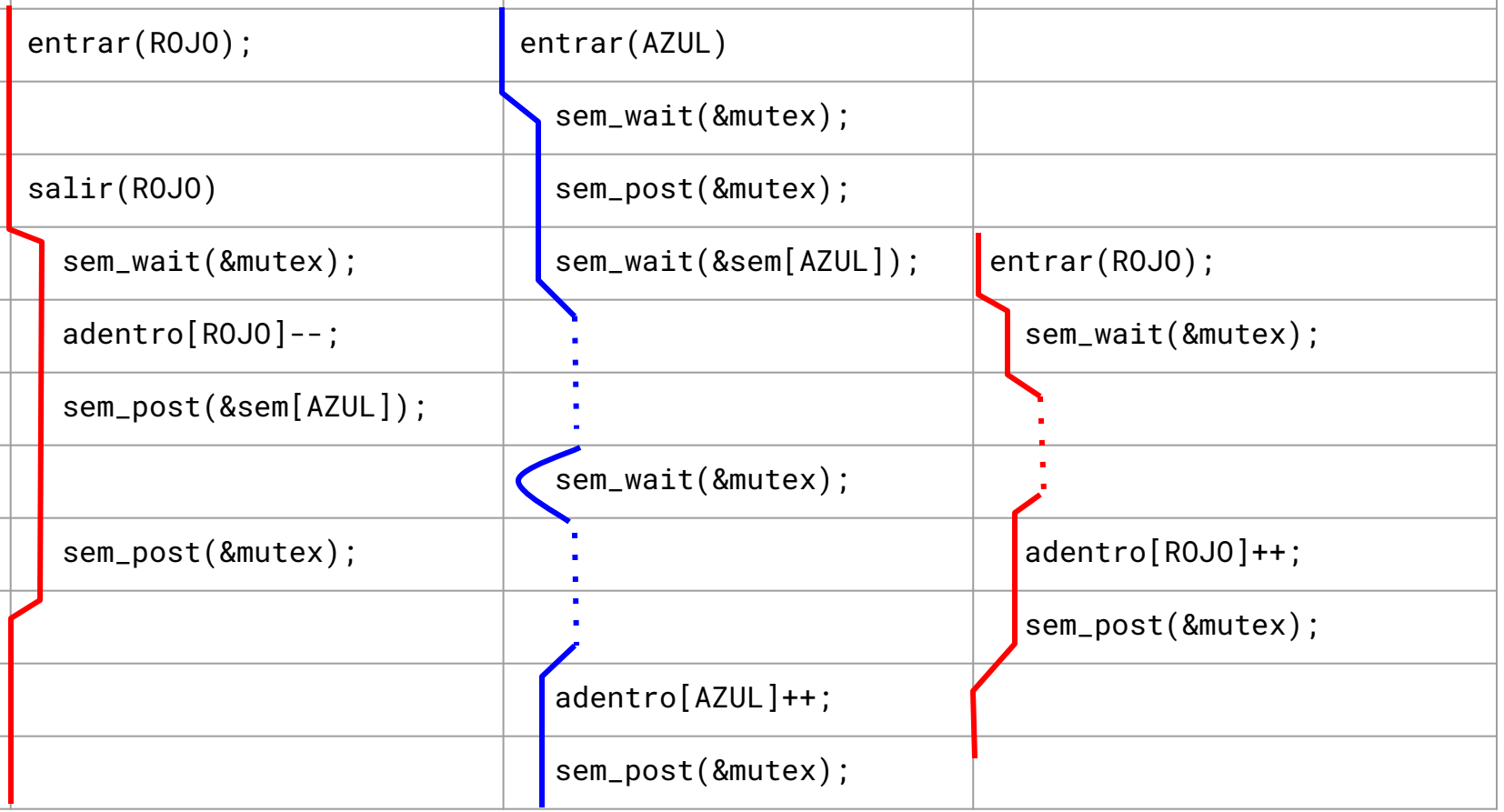
t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	entrar(ROJO);
4	adentro[ROJO]--;		sem_wait(&mutex);
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7	sem_post(&mutex);		adentro[ROJO]++;
8			sem_post(&mutex);
9			
10			



t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	entrar(ROJO);
4	adentro[ROJO]--;		sem_wait(&mutex);
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7	sem_post(&mutex);		adentro[ROJO]++;
8			sem_post(&mutex);
9			
10			



t	ROJO 0	AZUL	ROJO 1
0	entrar(ROJO);	entrar(AZUL)	
1		sem_wait(&mutex);	
2	salir(ROJO)	sem_post(&mutex);	
3	sem_wait(&mutex);	sem_wait(&sem[AZUL]);	entrar(ROJO);
4	adentro[ROJO]--;		sem_wait(&mutex);
5	sem_post(&sem[AZUL]);		
6		sem_wait(&mutex);	
7	sem_post(&mutex);		adentro[ROJO]++;
8			sem_post(&mutex);
9		adentro[AZUL]++;	
10		sem_post(&mutex);	




U. de Chile
v/s
Colo Colo
lotto
lotto
CRISTAL
LA TERCERA
www.terra.cl

- El problema es que ROJO 1 deja disponible 1 ticket para que lo tome AZUL que está en espera y fija la cola de espera en 0.

- ROJO 2 entra en ese momento, pensando que no hay nadie esperando y entra al baño. AZUL se despierta y entra al baño.

P2. Solución incorrecta

- Importante: La mayoría de estos dataraces ocurren en el momento de despertar los threads en espera, en ese instante un thread extra se puede escabullir en la zona crítica (si se cumplen las condiciones para que entre y se roba el mutex). Este thread puede cambiar las variables compartidas y alterar la correctitud del programa.
- Solución General: Debemos garantizar la correctitud del programa sin importar que un thread extra se robe el mutex, o debemos garantizar que nunca un thread extra pueda escabullirse. En general esto se soluciona cambiando todas las variables compartidas en el thread que despierta a los demás, los thread en espera se despiertan y solo deben retornar.

P2.b Corregir solución incorrecta

Reprograme la solución anterior de modo que siempre funcione correctamente. Utilice la siguiente metodología:

- Utilice 2 colas FIFO globales, una para cada equipo.

- Cuando un hincha deba esperar para entrar al baño, cree un semáforo con 0 tickets y póngalo en la cola correspondiente. Luego, suspenda el thread solicitando un ticket a este semáforo.

- Cuando salga el ultimo hincha de un equipo y haya hinchas del otro en espera, extraiga todos los semáforos de esa cola y deposite en cada uno de ellos un ticket para permitirle a los hinchas en espera entrar al baño.

- Utilice un semáforo para garantizar exclusión mutua en el acceso a las variables globales.