

Programación Concurrente

Práctica 3: Acciones Atómicas

1. Considere el siguiente fragmento de código (donde la variable `x` comienza inicializada en cero):

```
thread T1: {          thread T2: {          thread T3: {
  while(entrar());    while(entrar());    while(entrar());
  x = x + 1;          x = x + 2;          x = x + 3;
  salir();            salir();          salir();
}                    }                    }
```

Responda:

- a) Dada implementaciones triviales de las funciones `entrar` y `salir` determinar todos los valores posibles de la variable `x` al final de la ejecución de los threads.

```
entrar() {            salir() { }
  return false;
}
```

- b) Considere ahora las siguientes implementaciones de las funciones `entrar` y `salir` que utilizan la variable booleana global `ocupado` (inicializada en `false`) e indique si al usarlas se delimita correctamente una sección crítica (en caso contrario indicar al menos una propiedad que no se cumpla y mostrar una traza que ejemplifique el problema).

```
entrar() {            salir() {
  if (ocupado)        ocupado = true;
    return true;      }
  ocupado = true;
  return false;
}
```

- c) Analice el problema considerando que las funciones `entrar` y `salir` son atómicas ¿Se resuelve en este caso el problema de la exclusión mutua? Justifique su respuesta.

2. Considere la operación atómica fetch-and-add definida de la siguiente manera:

```
atomic int fetch-and-add(ref, x) {
  local = ref.shared;
  ref.shared = ref.shared + x;
  return local;
}
```

y el siguiente algoritmo para resolver el problema de la exclusión mutua que utiliza las variables compartidas `ticket` y `turno`, ambas inicializadas en 0.

```
// non-critical section
int miTurno = fetch-and-add(ticket, 1);
while (turno != miturno) {}
// critical section
fetch-and-add(ticket, -1);
// non-critical section
```

Indique si el algoritmo resuelve el problema de la exclusión mutua o no. Justifique (en caso negativo indicar que condición es violada y mostrar una traza).

3. Considere la siguiente operación

```
tomarFlag(mia, otro) {
  flags[mia] = !flags[otro];
}
```

Se propone el siguiente algoritmo para resolver el problema de la exclusión mutua entre dos procesos que utilizan el array compartido

```
global boolean[] flags = {false, false};

thread { // threadId=0      thread { // threadId=1
  while (!flags[0])        while (!flags[1])
    tomarFlag(0,1);        tomarFlag(1,0);
  // critical section      // critical section
  flags[0] = false;        flag[1] = false;
}                          }
```

Se solicita:

- Asumiendo que la operación **tomarFlag** no es atómica ¿Resuelve la propuesta anterior el problema de la exclusión mutua? Justifique su respuesta.
- En el caso en que **tomarFlag** es una acción atómica, el algoritmo es una solución al problema de la exclusión mutua. Justifique.

4. Considere el siguiente fragmento de código:

```
global int ticket = 0;
global int turn = 0;

thread T1: {
  // non-critical section
  int myTurn = getTurn();
  while (myTurn != turn);
  // critical section
  releaseTurn();
  // non-critical section
}

thread T2: {
  // non-critical section
  int myTurn = getTurn();
  while (myTurn != turn);
  // critical section
  releaseTurn();
  // non-critical section
}
```

Dadas las siguientes implementaciones de las funciones **getTurn** y **releaseTurn**.

```
getTurn() {
  return ticket++;
}

releaseTurn() {
  turn++;
}
```

Responda:

- Determine si al utilizar las funciones **getTurn** y **releaseTurn** se delimita correctamente una sección crítica (en caso contrario indicar al menos una propiedad que no se cumpla y mostrar una traza que ejemplifique el problema).
- Ahora considere como atómicas las funciones **getTurn** y **releaseTurn** e indique si en este caso se resuelve el problema de la exclusión mutua.
- Si esta solución se ejecuta en un entorno donde los enteros son representados con un byte (8 bits), es decir, el número sin signo más significativo representable es el 512 ¿Afecta esto su respuesta anterior?