Programación Concurrente

Práctica 3: Atomicidad

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

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

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

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

a) Dadas las siguientes 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() {
  return false;
}
```

```
salir() { }
```

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

```
salir() {
  ocupado = 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.

Ejercicio 2. 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
  while (!flags[0])
    tomarFlag(0,1);
  // seccion critica
  flags[0] = false;
}
```

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

Se solicita:

a) Asumiendo que la operación `tomarFlag` no es atómica ¿Resuelve la propuesta anterior el problema de la exclusión mutua? Justifique su respuesta.

b) En el caso en que `tomarFlag` es una acción atómica, el algoritmo es una solución al problema de la exclusión mutua. Justifique por qué es así.

Ejercicio 3. Sea `Ref` una clase con una variable pública `value` de tipo booleano. Considere la operación atómica test-and-set definida de la siguiente manera:

```
boolean test-and-set(Ref ref) {
  boolean result = ref.value; // lee el valor antes de cambiarlo
  ref.value = true;   // cambia el valor por true
  return result;      // retorna el valor leido anterior
}
```

Más el siguiente algoritmo para resolver el problema de la exclusión mutua que utiliza la variable compartida `flag`.

```
global Ref flag = new Ref();
flag.value = false;

thread {
  // seccion no critica
  while( test-and-set(flag) );
  // SECCION CRITICA
  flag.value = false;
  // seccion no critica
}
```

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

Ejercicio 4. Sea `Ref` una clase con una variable pública `value` de tipo entero. Considere la operación atómica fetch-and-add definida de la siguiente manera:

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

Más el siguiente algoritmo para resolver el problema de la exclusión mutua que utiliza las variables compartidas `ticket` y `turn`.

```
gobal Ref ticket = new Ref();
gobal Ref turn   = new Ref();
```

```
ticket.value = 0;
turn.value   = 1;

thread {
  // seccion no critica
  int myTurn = fetch-and-add(ticket, 1);
  while (!turn.value.equals(myTurn));
  // SECCION CRITICA
  fetch-and-add(turn, 1);
  // seccion no critica
}
```

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

b) 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 255, ¿Afecta esto su respuesta anterior?

Ejercicio 5. Considere el siguiente fragmento de código:

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


thread T1: {                  thread T2: {
  // non-critical section       // non-critical section
  int myTurn = getTurn();       int myTurn = getTurn();
  while (myTurn != turn);       while (myTurn != turn);
  // critical section           // critical section
  releaseTurn();                releaseTurn();
  // non-critical section       // non-critical section
}                             }
```

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

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

a) 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).

b) Ahora considere como atómicas las funciones `getTurn` y `releaseTurn` e indique si en este caso se resuelve el problema de la exclusión mutua.

c) 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 255, ¿Afecta esto su respuesta anterior?