Lista de Exercícios 1 - Programação Concorrente - 20/03/2019

As seguintes questões foram extraídas do Livro Fundamentos de Multithread, Paralelismo e Programação Distribuída de Gregory Andrews.

1, 8, 13, 15, 18, 24 e 33

2.1 Consider the outline of the program in Figure 2.1 that prints all the lines in a file that contain **pattern**.

(a) Develop the missing code for synchronizing access to **buffer**. Use the **await** statement to program the synchronization code.

(b) Extend your program so that it reads two files and prints all the lines that contain **pattern**. Identify the independent activities and use a separate process for each. Show all synchronization code that is required.

```
string buffer;  # contains one line of input
bool done = false;  # used to signal termination

co  # process 1:  find patterns
  string line1;
  while (true) {
    wait for buffer to be full or done to be true;
    if (done) break;
    line1 = buffer;
    signal that buffer is empty;
    look for pattern in line1;
    if (pattern is in line1)
      write line1;
  }
//  # process 2:  read new lines
  string line2;
  while (true) {
    read next line of input into line2;
    if (EOF) {done = true; break; }
    wait for buffer to be empty;
    buffer = line2;
    signal that buffer is full;
  }
oc;
```

Figure 2.1 Finding patterns in a file.

2.8 A queue is often represented using a linked list. Assume that two variables, **head** and **tail**, point to the first and last elements of the list. Each element contains a data field and a link to the next element. Assume that a null link is represented by the constant **null**.

(a) Write routines to (1) search the list for the first element (if any) that contains data value **d**, (2) insert a new element at the end of the list, and (3) delete the element from the front of the list. The search and delete routines should return **null** if they cannot succeed.

(b) Now assume that several processes access the linked list. Identify the read and write sets of each routine, as defined in (2.1). Which combinations of routines can be executed in parallel? Which combinations of routines must execute one at a time (i.e., atomically)?

(c) Add synchronization code to the three routines to enforce the synchronization you identified in your answer to (b). Make your atomic actions as small as possible, and do not delay a routine unnecessarily. Use the **await** statement to program the synchronization code.

2.13 Consider the following three statements:

```
S1:  x = x + y;
S2:  y = x - y;
S3:  x = x - y;
```

Assume that **x** is initially **2** and that **y** is initially **5**. For each of the following, what are the possible final values of **x** and **y**? Explain your answers.

(a) S_1; S_2; S_3;

(b) co ⟨S_1;⟩ // ⟨S_2;⟩ // ⟨S_3;⟩ oc

(c) co ⟨await (x > y) S_1; S_2;⟩ // ⟨S_3;⟩ oc

2.15 Consider the following program:

```
int x = 0, y = 10;
co while (x != y)  x = x + 1;
// while (x != y)  y = y - 1;
oc
```

(a) Does the program meet the requirements of the At-Most-Once Property (2.2)? Explain.

(b) Will the program terminate? Always? Sometimes? Never? Explain your answer.

2.18 Consider the following program:

```
co ⟨await (x > 0)  x = x - 1;⟩
// ⟨await (x < 0)  x = x + 2;⟩
// ⟨await (x == 0)  x = x - 1;⟩
oc
```

For what initial values of `x` does the program terminate, assuming scheduling is weakly fair? What are the corresponding final values? Explain your answer.

2.24 Consider the following precondition and assignment statement.

```
{x >= 4} ⟨x = x - 4;⟩
```

For each of the following triples, show whether the above statement interferes with the triple.

(a) `{x >= 0} ⟨x = x + 5;⟩ {x >= 5}`

(b) `{x >= 0} ⟨x = x + 5;⟩ {x >= 0}`

(c) `{x >= 10} ⟨x = x + 5;⟩ {x >= 11}`

(d) `{x >= 10} ⟨x = x + 5;⟩ {x >= 12}`

(e) `{x` is odd`} ⟨x = x + 5;⟩ {x` is even`}`

(f) `{x` is odd`} ⟨y = x + 1⟩ {y` is even`}`

(g) `{y` is odd`} ⟨y = y + 1;⟩ {y` is even`}`

(h) `{x` is a multiple of `3} y = x; {y` is a multiple of `3}`

2.33 Consider the following program:

```
int x = 10, c = true;

co ⟨await x == 0⟩; c = false;
// while (c) ⟨x = x - 1⟩;
oc
```

(a) Will the program terminate if scheduling is weakly fair? Explain.

(b) Will the program terminate if scheduling is strongly fair? Explain.

(c) Add the following as a third arm of the `co` statement:

```
while (c) {if (x < 0) ⟨x = 10⟩;}
```

Repeat parts (a) and (b) for this three-process program.