PROVA 2

MAC0438 - PROGRAMAÇÃO CONCORRENTE
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA - USP

1º SEMESTRE DE 2016 - PROF. MARCEL P. JACKOWSKI

RENATO LUI GEH
NUSP: 8536030

Questão 1

A Figura 1 ilustra as dependências $D = \{P_1 \to P_2, P_1 \to P_3, P_2 \to P_4, P_3 \to P_5, P_4 \to P_5\}$. Uma aresta direcionada indica uma dependência em D. Uma aresta não-direcionada indica a necessidade de um semáforo S_i. A altura de um nó indica se precisamos terminar algum processo antes. A altura do nó raíz P_1 é $h_1 = 0$, já que não temos nenhuma dependência para P_1. Se um nó i tem altura $h_i > 0$, então precisamos completar todos os nós que tenham altura menor que h_i. Por exemplo, para começarmos o processo P_2, precisamos antes satisfazer a dependência P_1; para começarmos o processo P_5, precisamos completar os nós P_4, P_3, P_2 e P_1 antes, já que estão em alturas menores que P_5.

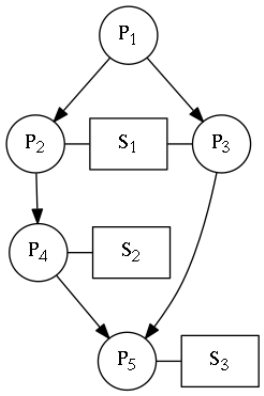

Figura 1

Portanto, sabemos do grafo quem devemos rodar antes de começarmos algum processo. O algoritmo abaixo mostra o código de cada processo e a inicialização dos semáforos.

Algoritmo 1

```
1:  sem S_1 := 0
2:  sem S_2 := 0
3:  sem S_3 := −1
4:  function P_1:
5:      código do processo 1
6:      V(S_1)
7:      V(S_1)
8:  function P_2:
9:      P(S_1)
10:     código do processo 2
11:     P(S_2)
12: function P_3:
13:     P(S_1)
14:     código do processo 3
15:     P(S_3)
16: function P_4:
17:     P(S_2)
18:     código do processo 4
19:     P(S_3)
20: function P_5:
21:     P(S_3)
22:     P(S_3)
23:     código do processo 5
```

Questão 2

```
1   type s[1..M];  // Buffer
2   int c = 0;
3   sem write = 1;
4   sem ready = 1;
5   sem full = 0;
6
7   produtor [i=1..N] { // Processos produtores
8     P(write);
9     P(ready);
10    push(s[c]); // Deposita
11    ++c;
12    if (c == M) V(full);
13    else V(ready);
14    V(write);
15  }
16
17  consumidor { // Processo consumidor
18    P(full);
```

```
19      for (i ← 1..M)
20        consume(S[i]); // Consome
21      V(ready);
22    }
```

Questão 3

(a) Se por acesso exclusivo quer-se dizer que somente um processo poderá acessar a base de dados por vez, então não, o acesso não é exclusivo, já que os processos leitores podem ler a base de dados simultaneamente. No entanto, se considerarmos acesso exclusivo como a restrição de não haver leitura e escrita simultânea, então sim, há exclusão neste caso devido ao semáforo `escrita`. Como leitura simultânea não gera conflitos, não seria necessário sequencializar a leitura.

(b)
- `sem a = 1`
  Tem o propósito de controlar o acesso a variável global `nr`, que é modificada e lida em vários processos, e portanto requer exclusão mútua.
- `sem b = 1`
  Garante a exclusão mútua da variável `nw`. Análogo ao semáforo `a`.
- `sem c = 1`
  Semáforo que dá preferência a processos escritores (vide item c).
- `sem leitura = 1`
  Previne que processos leitores leiam a base de dados quando um processo escritos está modificando-a.
- `sem escrita = 1`
  Previne que processos escritores escrevam enquanto leitores lêem. Além disso, garante exclusão mútua entre múltiplos escritores.
- `int nr = 0`
  Contador de leitores ativos. Apenas o primeiro e último leitores travam e destravam os processos escritores.
- `int nw = 0`
  Análogo a `nr` para os escritores.

(c) Esta solução dá preferência para os escritores devido ao semáforo `c`, já que este gasta o timeslice dos leitores com um semáforo para que os escritores possam ter mais tempo para escrita.

Questão 4

```
1   class semaphore {
2     int i=0;
3     cond c, k;
4
5     void P() {
6       wait(k);
7       if (i == 0) wait(c);
8       else ++i;
9       signal(k);
```

```
10      }
11
12      void V() {
13        wait(k);
14        if (empty(c)) --i;
15        else signal(c);
16        signal(k);
17      }
18    }
```