Sistemas Operacionais – Prof. Rafael R. Obelheiro

Exercícios Complementares — Comunicação Interprocessos — Respostas

1.

```
semaphore mutex = 1;

void aplicacao() {
    while (TRUE) {
        faz_algo();
        down(&mutex);
        come_memoria();
        up(&mutex);
    }
}
```

2. Como os recursos precisam ser alocados e usados simultaneamente, a solução é idêntica à do exercício anterior. Para tornar o algoritmo mais claro, seria melhor trocar `come_memoria()` por `come_cpu_memoria()`, mas isso não altera a estrutura da solução.

3. Na modelagem da solução, cada tipo de aplicação é associada a um valor inteiro entre 0 e 2, dependendo dos recursos usados (linhas 1–3). A função `escolhe_recursos()` (linha 7) retorna um inteiro entre 0 e 2 seguindo a mesma convenção. Um vetor de semáforos `s` (linha 5) é usado para sinalizar a aplicação correta (linhas 8 e 13), e um semáforo `pronto` (linha 4) é usado para sinalizar ao gerente a liberação dos recursos (linhas 15 e 6).

```
1  #define CPU_MEM 0
2  #define CPU_DSK 1
3  #define DSK_MEM 2
4  semaphore pronto = 1;
5  semaphore s[3] = 0;
```

```
3  void gerenciador() {
4      int i;
5      while (TRUE) {
6          down(&pronto);
7          i = escolhe_recursos();
8          up(&s[i]);
9      }
10 }
```

```
11 void aplicacao(int i) {
12     while (TRUE) {
13         down(&s[i]);
14         usa_recursos();
15         up(&pronto);
16     }
17 }
```

4.

```
int n = 1;                  /* variável compartilhada entre X, Y e Z */
semaphore S1 = 0, S2 = 0;   /* semáforos compartilhados */

void X() {
   n = n * 16;
   up(&S1);
}
```

```
void Y() {
   down(&S2);
   n = n / 7;
}
```

```
void Z() {
   down(&S1);
   n = n + 40;
   up(&S2);
}
```

5. Na solução, o vetor `lugares` (linha 4) representa o mapa de alocação dos lugares. O semáforo `livres` (linha 3) conta o número de lugares livres; o `down()` na linha 9 faz com que os processos fiquem bloqueados quando todos os lugares estiverem ocupados, enquanto o `up()` na linha 21 libera um lugar caso o pagamento não seja autorizado. As regiões críticas neste código são os trechos em que o mapa de alocação é modificado (RC 1: linhas 11–14, RC 2: linha 19), e são guardadas pelo semáforo `mutex` (definido na linha 4, e usado nas linhas 10, 15, 18 e 20). Como a autorização de pagamento (linhas 16 e 17) é muito lenta e sujeita a falhas, ela deve estar fora da região crítica.

```
1  enum { LIVRE, OCUP };
2  semaphore mutex = 1;
3  semaphore livres = MAX;
4  int lugares[MAX] = LIVRE;
5
6  void quiosque() {
7      int lug, pg_ok;
8      while (TRUE) {
9          down(&livres);
10         down(&mutex);
11         do {
12             lug = escolhe_lugar();
13         } while (lugares[lug] == OCUP);
14         lugares[lug] = OCUP;
15         up(&mutex);
16         pg_ok = autoriza_pagamento();
17         if (!pg_ok) {
18             down(&mutex);
19             lugares[lug] = LIVRE;
20             up(&mutex);
21             up(&livres);
22         } else {
23             imprime_bilhete(lug);
24         }
25     }
26 }
```