LEIC
LERCI

Sistemas Operativos

Colectânea de Exercícios de Sincronização

Soluções

Outubro 2004

Exercícios de Sincronização

A. Competição por um recurso

---

1. [Banco]

a.

```
void levantar(int valor) {
  esperar(mutex);
  if (saldo >= valor) {
    saldo = saldo - valor;
  }
  assinalar(mutex);
}
```

```
void depositar(int valor) {
  esperar(mutex);
  saldo = saldo + valor;
  assinalar(mutex);

}
```

b. Existem três situações de concorrência.:

- Dois processos executam "levantar" em simultâneo:
  - Se ambos os valores a levantar forem inferiores ao saldo e a sua soma for superior ao saldo só um dos levantamentos se deveria realizar, mas se não existir o mutex então é possível que ambos sejam efectuados levando o saldo para valores negativos. Para tal basta que o primeiro processo perca o processador logo após ter verificado que o saldo era superior ao valor a levantar mas antes da subtracção. O segundo processo verifica o saldo (ainda não actualizado) e efectua a subtracção. Por fim o primeiro processo retoma o processador efectuando a subtracção.
- Dois processos que executam depositar em simultâneo:
  - Neste caso é necessário salientar que a operação "saldo = saldo + valor" não é normalmente atómica. Isto é primeiro é calculada a soma e colocada numa variável temporária (ou num registo) e só depois é actualizado o valor do saldo. Se um processo perder o processador no meio destas duas operações(ex.: tmp1 = 5 + 4 (tmp1 = saldo + valor1)), pode acontecer que outro processo calcule um novo saldo com base no valor de saldo antigo (ex.: saldo = 5 + 3 (tmp2 = saldo + valor2, saldo = tmp2)) e quando o processo um recomeça coloca em saldo o valor da primeira operação perdendo o segundo depósito (ex.: saldo = 9 (saldo = tmp1)). Note-se que o mesmo se pode passar com a subtracção das rotinas levantar.
- Dois processos executam em simultâneo as operações depositar e levantar

A situação é semelhante à anterior uma vez que a subtracção também é usualmente efectuada em dois passos.

---

2. [Somas concorrentes]

```
semaforo_t mutex;
lista_t lista;
int numsomas = 0;

void Soma()
{
    while(1)
    {
        Esperar(mutex);
        if (numsomas == N-1)
        {
            Assinalar(mutex);
            return;
        }
        else
        {
            int n1 = TirarNumero(lista);
            int n2 = TirarNumero(lista);
            PorNumero(lista,n1+n2);
            numsomas++;
            Assinalar(mutex);
        }
    }
}
```

---

3. [Leitores-Escritores*]

a. A base de dados contém campos que devem estar coerentes entre si. Por exemplo se a base de dados contém informação de contas bancárias, quando se faz uma transferência de uma conta para a outra, ao valor levantado de uma conta deve corresponder um valor depositado noutra. Por isso para garantir que todas as operações são efectuadas sobre estados coerentes da base de dados é necessário garantir que todas as operações de actualização acabam antes de começar outra.

b.

```
typedef int semaforo;
semaforo mutex = 1;     // controls access to 'rc'
semaforo db = 1;        // controls access to the database
int cl = 0;             // # of processes reading or wanting to

void Escreve()
{
    Esperar(&db);
    escreverDB( );
    Assinalar(&db);
}
```

```
void Le()
{
    Esperar (&mutex);
    cl = cl + 1;
    if (cl == 1)
    {
        Esperar (&db);
    }
    Assinalar (&mutex);
    lerDB( );
    Esperar (&mutex);
    cl = cl -1;
    if (cl == 0)
    {
        Assinalar(&db);
    }
    Assinalar (&mutex);
}
```

---

4. [Ponte estreita]

a.

```
semaforo_t mutexOeste = 1;
semaforo_t mutexEste = 1;
semaforo_t ponte = 1;
int CarrosDeOesteNaPonte = 0;
int CarrosDeEsteNaPonte = 0;

void CarroDeOeste()
{
    Esperar (&mutexOeste);
    CarrosDeOesteNaPonte++;
    if (CarrosDeOesteNaPonte == 1)
    {
        Esperar (&ponte);
    }
    Assinalar (&mutexOeste);
}

void CarrosSaiDeOeste()
{
    Esperar (&mutexOeste);
    CarrosDeOesteNaPonte--;
    if (CarrosDeOesteNaPonte == 0)
    {
        Assinalar (&ponte);
    }
    Assinalar (&mutexOeste);
}
```

```
void carroDeEste()
{
    Esperar (&mutexEste);
    CarrosDeEsteNaPonte++;
    if (CarrosDeEsteNaPonte == 1)
    {
        Esperar (&ponte);
    }
    Assinalar (&mutexEste);
}

void carrosSaiEste()
{
    Esperar (&mutexEste);
    CarrosDeEsteNaPonte--;
    if (CarrosDeEsteNaPonte == 0)
    {
        Assinalar (&ponte);
    }
    Assinalar (&mutexEste);
}
```

b.

```
semaforo mutexOeste = 1;
semaforo mutexEste = 1;
semaforo ponte = 1;
semaforo ponteComEspacoOeste = 5;
semaforo ponteComEspacoEste = 5;
int CarrosDeOesteNaPonte = 0;
int CarrosDeEsteNaPonte = 0;

void CarroDeOeste()
{
    Esperar(&ponteComEspacoOeste);
    Esperar (&mutexOeste);
    CarrosDeOesteNaPonte++;
    if (CarrosDeOesteNaPonte == 1)
    {
        Esperar (&ponte);
    }
    Assinalar (&mutexOeste);
}

void CarrosSaiDeOeste()
{
    Esperar (&mutexOeste);
    CarrosDeOesteNaPonte--;
    if (CarrosDeOesteNaPonte == 0)
    {
        Assinalar (&ponte);
    }
    Assinalar (&mutexOeste);
    Assinalar(ponteComEspacoOeste);
}
```

```
void CarroDeEste()
{
    Esperar(ponteComEspacoEste);
    Esperar (&mutexEste);
    CarrosDeEsteNaPonte++;
    if (CarrosDeEsteNaPonte == 1)
    {
        Esperar (&ponte);
    }
    Assinalar (&mutexEste);
}

void CarrosSaiEste()
{
    Esperar (&mutexEste);
    CarrosDeEsteNaPonte--;
    if (CarrosDeEsteNaPonte == 0)
    {
        Assinalar (&ponte);
    }
    Assinalar (&mutexEste);
    Assinalar(ponteComEspacoEste);
}
```

5. [Searchers, inserters, deleters]

```
semaforo mutexOeste = 1;
semaforo mutexEste = 1;
semaforo ponte = 1;
semaforo ponteComEspacoOeste = 5;
semaforo ponteComEspacoEste = 5;
int CarrosDeOesteNaPonte = 0;
int CarrosDeEsteNaPonte = 0;

void SearcherIn()
{
    Esperar(semNotDeleter);
    notDeleterNum++;
    if (notDeleterNum == 1)
    {
        Esperar (semDB);
    }
    Assinalar (semNotDeleter);
}

void InserterIn()
{
    Esperar(semInserter);
    Esperar(semNotDeleter);
    notDeleterNum++;
    if (notDeleterNum == 1)
    {
        Esperar (semDB);
    }
    Assinalar (semNotDeleter);
}

void DeleterIn()
{
    Esperar(semDB);
}

void SearcherOut()
{
    Esperar(semNotDeleter);
    notDeleterNum--;
    if (notDeleterNum == 0)
    {
        Assinalar (semDB);
    }
    Assinalar (semNotDeleter);
}

void InserterOut()
{
    Assinalar(semInserter);
    Esperar(semNotDeleter);
    notDeleterNum--;
    if (notDeleterNum == 0)
    {
        Assinalar (semDB);
    }
    Assinalar (semNotDeleter);
}

void DeleterOut()
{
    Assinalar(semDB);
}
```

---

6. [Parque de estacionamento]

Solução A: estado do parque modificado por quem sai.

```
#define TIPOS 3
#define DOC 0
#define FUN 1
#define ALU 2
#define MAX 100

semaforo semExMut, semExTipo[TIPOS]; // 0
int contador;
int fila[TIPOS]; // = {0,0,0};

void EntraViatura(tipo v)
{
    Esperar (semExMut);
    if (contador < MAX)
    {
        contador++;
        Assinalar (semExMut);
    }
    else
    {

void SaiViatura()
{
    tipo v;
    Esperar (semExMut);
    if (contador < MAX)
    {
        contador--;
        Assinalar (semExMut);
    }
    else
```

```
        fila[v]++;
        Assinalar (semExMut);
        Esperar (semExTipo[v]);
    }
}
```

```
    {
        v = -1;
        if (fila[DOC] > 0)
            v= DOC;
        else if (fila[FUN] > 0)
            v= FUN;
        else if (fila[ALU] > 0)
            v = ALU;
        if (v >= 0)
        {
            Assinalar (semExTipo[v]);
            fila[v]--;
        }
        else
            contador --;
        Assinalar (semExMut);
    }
}
```

Solução B: estado do parque modificado por quem entra.

```
#define TIPOS 3
#define DOC 0
#define FUN 1
#define ALU 2
#define MAX 100

semaforo_t semExMut; //(1),
semaforo_t semExTipo[TIPOS]; // 0
int contador; // = 0,
int fila[TIPOS]; // = {0,0,0};
int Emespera; // = 0;

EntraViatura(tipo v)
{
    Esperar (semExMut);
    if (Emespera>0 || contador==MAX)
    {
        fila[v]++;
        Emespera++;
        Assinalar (semExMut);
        Esperar (semExTipo[v]);
        Esperar (semExMut);
        Fila[v]--;
        Emespera --;
    }
    contador ++;
    Assinalar (semExMut);
}
```

```
SaiViatura()
{
    Esperar (semExMut);
    if (Emespera>0)
    {
        if (fila[DOC] > 0)
        {
            Assinalar(semExTipo[DOC]);
        }
        else if (fila[FUN] > 0)
        {
            Assinalar(semExTipo[FUN]);
        }
        else if (fila[ALU] > 0)
        {
            Assinalar(semExTipo[ALU]);
        }
    }
    contador --;
    Assinalar (semExMut);
}
```

---

7. [Passeios turísticos]

```
semaforo_t havaganocarro = CriarSemaforo(C);
semaforo_t carroestacheio;
semaforo_t carroaindanaocheio;
int numpnocarro = 0;
```

```
Passageiro()
{
    while(1)
    {
        Esperar(semPoderEntrar);
        EntrarNoCarro();
        Esperar(mutex);
        numpnocarro++;
        if (numpnocarro == C)
        {
             Assinalar(semEstarCheio);
        }
        Assinalar(mutex)
        Esperar(semPartida);

        Esperar(semChegada);
        SairDoCarro();
        Esperar(mutex);
        numpnocarro--;
        if (numpnocarro == 0) {
            for (int i=0; i<C;i++)
            {
            Assinalar(semPoderEntrar);
            }
        }
        Assinalar(mutex);
    }
}
```

```
Carro()
{
    while(1)
    {
        Esperar(semEstarCheio);
        for (int i = 0; i < C; i++)
        {
             Assinalar(semPartida);
        }

        Partir();

        Chegar();
        for (int i=0; i<C; i++)
        {
            Assinalar(semChegada);
        }
    }
}
```

B. Competição por múltiplos recursos

8. [Múltiplos recursos]

```
#define NPROCS 4
#define NRECURSOS 6

semaforo_t semP[NPROCS];          // inicializar a 0
semaforo_t semRecurso[NRECURSOS]; // inicializar a 0
semaforo_t semFimTarefa;          // inicializar a 0
semaforo_t semMutex;              // inicializar a 1
int contprocs=0, acumulador=0;

void Agente()
{
    while(1) {
        // escolher configuração aleatória
        // execução de 4 operações Assinalar nos recursos correspondentes
        Esperar(semFimTarefa);
    }
}

void P(int i)
{
    while(1) {
        Esperar(semP[i]);
        // código que executa a tarefa
        Assinlar(semFimTarefa);
    }
}

void EsperaRecurso(int n)
{
```

```
    while(1) {
        Esperar(semRecurso[n]);
        Esperar(semMutex);
        acumulador += power(2,5-n);
        contprocs++;
        if(cont == 4) {
             switch(acumulador) {
                  case 58: Assinalar(semP[1]); break;
                  case 54: Assinalar(semP[2]); break;
                  case 23: Assinalar(semP[3]); break;
                  case 29: Assinalar(semP[4]); break;
             }
             acumulador = 0;
             contprocs = 0;
        }
        Assinalar(semMutex);
    }
}
```

Notas:

- A ideia é lançar uma tarefa EsperaRecurso associada a cada recurso. Quando o agente assinala os semáforos de quatro recursos, é avaliada qual a configuração de recursos disponível com base no valor constante no acumulador. Em função desta configuração, é desbloqueado o processo P respectivo que já tem à sua disposição todos os recursos que necessita para prosseguir.
- O acumulador funciona como uma máscara. Cada bit do acumulador está associado a um determinado recurso: o recurso 0 corresponde ao bit 5 (i.e. $2^5 = 32$), o recurso 5 corresponde ao bit 0 (i.e. $2^0 = 1$).
- Alternativamente ao acumulador, poderiamos ter uma matriz que representasse a tabela especificada e testar as configurações varendo as linhas da matriz.

---

9. [Jantar dos Filósofos*]

```
#define N 5
#define ESQUERDA (i+N-1)%N
#define DIREITA (i+1)%N
#define PENSAR 0
#define FOME 1
#define COMER 2

semaforo_t semFilosofo[N];  // inicializar a 0
semaforo_t semMutex;        // inicializar a 1
int estado[N];

void Filosofo(int i)
{
    while(1) {
        Pensar();
        ObterPauzinhos(i);
        Comer();
        LargarPauzinhos(i);
    }
}

void ObterPauzinhos(int i)
{
    Esperar(mutex);
    estado[i] = FOME;
    Teste(i);
    Assinalar(mutex);
```

```
    Esperar(semFilosofo[i]);
}

void LargarPauzinhos(int i)
{
    Esperar(mutex);
    estado[i] = PENSAR;
    Teste(ESQUERDA);
    Teste(DIREITA);
    Assinalar(mutex);
}

void Teste(int i)
{
    if(estado[i] == FOME &&
        estado[ESQUERDA] != COMER &&
        estado[DIREITA] != COMER)
    {
        estado[i] = COMER;
        Assinalar(semFilosofo[i]);
    }
}
```

---

10. [Cruzamento]

Solução A: Matrizes para representar o estado do sistema

```
#define N 4
#define EMPTY 0
#define REQUEST 1
#define RESERVED 2

int route[N][N];
typedef int semaforo;
semaforo_t mutex = 1;
semaforo_t s[N][N];

void request(int sl, int el)
{
    Esperar(&mutex);
    route[sl][el] = REQUEST;
    test(sl,el);
    Assinalar(&mutex);
    Esperar(&s[sl][el]);
}

void leave(int sl, int el)
{
    int i,j;

    Esperar(&mutex);
    route[sl][el] = EMPTY;
    for (i = 0; i < NLANES; i++)
    {
        for (j = 0; j < NLANES; j++)
        {
            if (useCommonRegion(sl,el,i,j))
            {
                test(i,j);
            }
        }
    }
    Assinalar(&mutex);
}
```

```
void test(int sl, int el)
{
    int i,j,wait = 0;
    if (route[sl][el] != REQUEST)
    {
        return;
    }
    for (i = 0; i < NLANES; i++)
    {
        for (j = 0; j < NLANES; j++)
        {
            if (useCommonRegion(sl,el,i,j) && (route[i][j] == RESERVED))
            {
                wait = 1;
                break;
            }
        }
    }
    if (!wait)
    {
        route[sl][el] = RESERVED;
        Assinalar(&s[sl][el]);
    }
}
```

Solução B: Estrutura representando o carro em vez de matrizes

```
#define N 4
#define EMPTY 0
#define REQUEST 1
#define RESERVED 2

// esta estrutura representa o carro na entrada do cruzamento
typedef struct
{
    int destino; // saida do cruzamento para onde se dirige o carro
    int estado; // estado do carro: EMPTY, REQUEST, RESERVED
    semaforo_t s; // semaforo para fazer o carro esperar
}
carro;

carro entradas[N];
semaforo_t mutex = 1;

void request(int sl, int el)
{
    Esperar(&mutex);
    entradas[sl].destino = el;
    entradas[sl].estado = REQUEST;
    test(sl,el);
    Assinalar(&mutex);
    Esperar(&entradas[sl].s);
}

void leave(int sl, int el)
{
    int i;

    Esperar(&mutex);
    entradas[sl].estado = EMPTY;
    for (i = 0; i < N ; i++)
    {
        if ((i!=sl) && useCommonRegion(sl,el,i,entradas[i].destino) &&
                (entradas[i].estado = REQUEST))
        {
            test(i,entradas[i].destino);
        }
```

```
    }
    Assinalar(&mutex);
}

void test(int sl, int el)
{
    int i,wait = 0;

    for (i = 0; i < N ; i++)
    {
        if ((i!=sl) && useCommonRegion(sl,el,i,entradas[i].destino) &&
                (entradas[i].destino == RESERVED))
        {
            wait = 1;
            break;
        }
    }
    if (!wait)
    {
        entradas[sl].estado = RESERVED;
        Assinalar(&entradas[sl].s);
    }
}
```

C. Cooperação entre processos

11. [Produtores-Consumidores*]

a.

```
#define N ..
#define NUMPROCSPRODUTORES ..
#define NUMPROCSCONSUMIDORES ..

ItemInformacao buffer[N];
int indiceProxLeitura = 0;
int indiceProxEscrita = 0;
semaforo_t posicoesSemInfo = CriarSemaforo(N);
semaforo_t posicoesComInfo = CriarSemaforo(0);
semaforo_t semExMut = CriarSemaforo(1);

void DepositaItem(ItemInformacao item)
{
    Esperar(posicoesSemInfo);
    Esperar(semExMut);

    buffer[iProxEscrita++] = item;
    if (iProxEscrita >= N) {
        iProxEscrita = 0;
    }

    Assinalar(semExMut);
    Assinalar(posicoesComInfo);
}
```

```
void RetiraItem(ItemInformacao item) {
    Esperar(posicoesComInfo);
    Esperar(semExMut);

    item =  buffer[iProxLeitura++];
    if (iProxLeitura >= N ) {
        iProxLeitura = 0;
    }

    Assinalar(semExMut);
    Assinalar(posicoesSemInfo);
}
```

b.

```
struct ItemInformacao Buffer[N];
int IProxLeitura = 0;
int IProxEscrita = 0;
int ocupados = 0;
semáforo_t sema_livres = CriaSemaforo(0);
trinco tr_buffer = CriaTrinco();

void DepositaItem(ItemInformacao item)
{
    Esperar(sema_livres);
    Buffer[IProxEscrita] = item;
    IProxEscrita++;
    if (IProxEscrita == N)
    {
        IProxEscrita = 0;
    }
    Fechar(tr_buffer);
    ocupados++;
    Abrir(tr_buffer);
}
```

```
void RetiraItem(ItemInformacao item)
{
    Fechar(tr_buffer);
    if (ocupados > 0)
    {
        item = Buffer[IProxLeitura];
        IProxLeitura++;
        if (IProxLeitura == N)
        {
            IProxLeitura = 0;
        }
        ocupados--;
        Abrir(tr_buffer);
        Assinalar(sema_livres);
    }
    else
    {
        Abrir(tr_buffer);
    }
}
```

---

12. [Pipe Unix]

```
#define N ..
#define NUMPROCSPRODUTORES ..
#define NUMPROCSCONSUMIDORES ..

Item buffer[N];
Int poslivres = N, pespera = 0; cespera = FALSE;
semaforo_t semExMut = CriarSemaforo(1);
semaforo_t semExMutCons = CriarSemaforo(1);
semaforo_t semEsperaProds = CriarSemaforo(0);
semaforo_t semEsperaCons = CriarSemaforo(0);

void DepositaItem(Item item, int n) {
    Esperar(semExMut);
    while (n > poslivres) {
        pespera++;
        Assinalar(semExMut);
        Esperar(semEsperaProds);
        Esperar(semExMut);
    }

    // Depositar n itens no buffer

    poslivres -= n;
    if (cespera == TRUE)
    {
        Assinalar(semEsperaCons);
    }
    Assinalar(semExMut);
}
```

```
void RetiraItem(ItemInformacao item) {
    Esperar(semExMutCons);
    Esperar(semExMut);
    if (poslivres == N) {
        cespera = TRUE;
        Assinalar(semExMut);
        Esperar(semEsperaCons);
        Esperar(semExMut);
        cespera = FALSE;
    }

    // Retirar min(n,N-poslivres)

    while (pespera > 0) {
        Assinalar(semEsperaProds);
        pespera--;
    }
    Assinalar(semExMut);
    Assinalar(semExMutCons);
}
```

---

13. [Passagem de testemunho]

```
semaforo_t semtest[N];  // Inicializar a 0
semaforo_t mutex        // não é necessário...
int testemunho = 0;

void Test(int mypid)
{
    while(1)
    {
        Esperar(semtest[mypid]);
        Esperar(mutex);
        if (mypid == 0)
        {
            printf("%d\n",testemunho);
        }
        testemunho = (testemunho + 1) % N;
        Assinalar(mutex);
        Assinalar(semtest[testemunho]);
    }
}

Assinalar(semtest[0]);
```

---

14. [Barbeiro*]

Solução A

```
trinco mutex;
semaforo cadeira = 0;
semaforo barbeiro = 0;

void CustomerThread()
{
    Fechar(mutex);
    if (cliespera < N)
    {
        cliespera++;
        Signal(barbeiro);
        Abrir(mutex);

        Esperar(cadeira);

        // Corta cabelo
    }
    else
    {
        Abrir(mutex);
    }
}
```

```
void BarberThread()
{
    while(1)
    {
        Esperar(barbeiro);
        Fechar(mutex);
        cliespera--;
        Abrir(mutex);

        Assinalar(cadeira);

        // Corta cabelo
    }
}
```

Solução B

```
trinco mutex;
semaforo cadeira = 0;
semaforo barbeiro = 0;
semaforo clipronto = 0;
```

```
void CustomerThread()
{
    Fechar(mutex);
    if (cliespera < N)
    {
        cliespera++;
        Signal(barbeiro);
        Abrir(mutex);

        Esperar(cadeira);

        Fechar(mutex);
        cliespera--;
        Abrir(mutex);

        Assinalar(clipronto);

        // Corta cabelo
    }
    else
    {
        Abrir(mutex);
    }
}
```

```
void BarberThread()
{
    while(1)
    {
        Esperar(barbeiro);

        Assinalar(cadeira);

        Esperar(clipronto);

        // Corta cabelo
    }
}
```

Notas:

- As soluções são praticamente iguais. A única diferença é quem fica responsável por decrementar o número de clientes em espera, o barbeiro (A) ou o cliente que se sentou na cadeira (B). Penso que não é necessário o semáforo clipronto na solução B, mas assim garantimos que ambas as soluções são equivalentes.
- Seria também possível acrescentar mais um semáforo para o cliente ficar bloqueado até que o barbeiro termine o corte de cabelo.

---

15. [Hilzer's Barbershop]

```
trinco mutex;
semaforo_t semSofa = 4;
semaforo_t semCadeiras = 3;
semaforo_t semBarbeiro = 0;
semaforo_t semFimCorte = 0;
semaforo_t semPagar = 0;
int clijoja = 0;
```

```
void Cliente()
{
    Fechar(mutex);
    if (cliloja == 20 )
    {
        Assinalar(mutex);
        return;
    }
    cliloja++;
    Abrir(mutex);

    EntrarNaLoja();

    Esperar(semSofa);

    SentarNoSofa();

    Esperar(semCadeiras);
    Assinalar(semSofa);
```

```
void Barbeiro()
{
    while(1)
    {
        Esperar(semBarbeiro);

        CortarCabelo();

        Assinalar(semFimCorte);
        Esperar(semPagar);

        AceitarPagamento();

        Assinalar(semFim);
    }
}
```

```
    SentarNaCadeira();

    Assinalar(semBarbeiro);
    Esperar(semFimCorte);
    Assinalar(semCadeira);

    Pagar();

    Assinalar(semPagar);
    Esperar(semFim);

    Sair();

    Esperar(mutex);
    cliloja--;
    Assinalar(mutex);
}
```

---

16. [Fumadores]

```
#define TOBACCO 0
#define PAPER   1
#define MATCH   2

semaforo_t smokers[3]; // Init a 0
semaforo_t table;      // Init a 0
int lacki = -1;

void Smoker(int i)
{
    Wait(smokers[i]);
    lacki = -1;
    Signal(table);
    Smoke();
}
```

```
void Agent()
{
    lacki = Rand() % 3;
    Signal(smokers[lacki]);
    Wait(table);
}
```

---

17. [Jantar de Gauleses]

```
semaforo_t hajavali = CriarSemaforo(0);
semaforo_t naohajavali = criarSemaforo(0);
semaforo_t mutex = CriarSemaforo(1);

javali_t RetiraJavali()
{
    Esperar(mutex);
    if (numjavalis == 0)
    {
        Assinalar(naohajavali);
        Esperar(hajavali);
    }

    // retirar javali

    numjavalis--;
    Assinalar(mutex);
}
```

```
void ColocaJavalis(int n)
{
        Esperar(naohajavali);
        numjavalis = n;
        Assinalar(hajavali);
}
```

---

18. [Pai Natal]

```
trinco_t mutex;
trinco_t mutexElfos;
semaforo_t semPaiNatal = 0;
semaforo_t semRenas = 0;
semaforo_t semElfos = 0;
semaforo_t semEntrar = 0;
semaforo_t semGrupoElfos = 1;
int numRenas = 0, numElfos = 0;

void PaiNatal()
{
    while(1)
    {
        Esperar(semPaiNatal);
        Fechar(mutex);
        if(numRenas == 9)
        {
             PrepararTreno();
             while(numRenas > 0)
             {
                 Assinalar(semRenas);
                 numRenas--;
             }
             Abrir(mutex);
        }
        else
        {
             Abrir(mutex);
             Assinalar(semElfos);
             AjudarElfos();
        }
    }
}
```

```
void Rena()
{
    while(1)
    {
        Fechar(mutex);
        numRenas++;
        if (numRenas == 9)
        {
            Assinalar(semPaiNatal);
        }
        Abrir(mutex);
        Esperar(semRenas);
        AtrelarATreno();
    }
}

void Elfo()
{
    while(1)
    {
         Esperar(semGrupoElfos);
         Fechar(mutexElfos);
         numElfos++;
         if(numElfos == 3)
         {
             Assinalar(semPaiNatal);
             Esperar(semElfos);
             Assinalar(semEntrar);
         }
         else
         {
             Assinalar(semGrupoElfos);
         }
         Abrir(mutexElfos);
         Esperar(semEntar);
         Assinalar(semEntrar);

         SerAjudado();

         Fechar(mutexElfos);
         numElfos--;
         if(numElfos == 0)
         {
             Assinalar(semGrupoElfos);
         }
         Abrir(mutexElfos);
    }
}
```

D. Objectos avançados de sincronização

19. [Barreira]

```
typedef struct
{
    semaforo_t sem;
    semaforo_t mutex;
    int numassociar;
    int numchegar;
}
barreira_t;

void InicializarBarreira(barreira_t* b)
{
    b->sem = CriarSemaforo(0);
    b->mutex = CriarSemaforo(1);
    b->numassociar = 0;
    b->numchegar = 0;
}

void Associar(barreira_t* b)
{
    Esperar(b->mutex);
    b->numassociar++;
    Assinalar(b->mutex);
}
```

```
void Chegar(barreira_t* b)
{
    Esperar(b->mutex);
    b->numchegar++;
    if (b->numchegar < b->numassociar)
    {
        Assinalar(b->mutex);
        Esperar(b->sem);
    }
    else
    {
        while(b->numchegar > 0)
        {
            Assinalar(b->sem);
            b->numchegar--;
        }
    }
    Assinalar(b->mutex);
}
```

20. [Monitor]

a.

```
// Monitor

typedef struct {
    trinco_t trinco;
}* monitor_t;


monitor_t CriaMonitor()
{
    monitor_t m = AlocarMemoria();
    return m;
}

Entrar(monitor_t m)
    Fechar(m->mutex);
```

```
// Variáveis de Condição

typedef struct {
    monitor_t monitor;
    semaforo_t aespera;
    int cont;
}* varcond_t;

varcond_t CriaVarCondicao(monitor_t m)
    varcond_t vc = AlocarMemoria();
    vc->aespera = CriarSemaforo(0);
    vc->count = 0;
    vc->monitor = m;
    return vc;
}
```

```
}

Sair(monitor_t m)
    Abrir(m->mutex);
}
```

```
Aguardar(varcond_t vc)
{
    vc->cond++;
    Assinalar(vc->trinco);
    Esperar(vc->aespera);
    Esperar(vc->trinco);
}

Notificar(varcond_t vc)
    if (vc->count > 0)
    {
        Assinalar(vc->aespera);
        vc->cond--;
    }
}
```

b.

```
int esteNaPonte = 0;
int oesteNaPonte = 0;
int esteEspera = 0;
int oesteEspera = 0;

monitor_t m = CriarMonitor();
varcond_t este = CriarVarCond(m);
varcond_t oeste = CriarVarCond(m);

void carroDeOeste()
{
    Entrar(m);
    while(esteNaPonte > 0) {
        oesteEspera++;
        Aguardar(oeste);
        oesteEspera--;
    }
    oesteNaPonte++;
    if(oesteEspera > 0) {
        Notificar(oeste);
    }
    Sair(m);
}

void carrosSaiDeOeste()
{
    Entrar(m);
    oesteNaPonte--;
    if(oesteNaPonte == 0) {
        Notificar(este);
    }
    Sair(m);
}
```

```
void carroDeEste()
{
    Entrar(m);
    while(oesteNaPonte > 0) {
        esteEspera++;
        Aguardar(este);
        esteEspera--;
    }
    esteNaPonte++;
    if(esteEspera > 0) {
        Notificar(este);
    }
    Sair(m);
}

void carrosSaiEste()
{
    Entrar(m);
    esteNaPonte--;
    if(esteNaPonte == 0) {
        Notificar(oeste);
    }
    Sair(m);
}
```