

INE 5410 - Laboratório

AULA 04 – CONCORRÊNCIA COM SEMÁFOROS

1. INTRODUÇÃO

O uso de semáforos efetivamente permite efetuar o controle da concorrência, impondo restrições à execução de trechos de código de threads ou processos. As restrições impostas podem ser aplicadas a diversos cenários de execução em sistemas concorrentes, fazendo deste mecanismo um aliado importante para a programação destes.

2. OBJETIVO DA AULA

Nesta aula prática, será utilizado o mecanismo de semáforo para verificar sua contribuição em diferentes cenários de execução de programas concorrentes.

3. EXEMPLOS

3.1 Tempo real

Neste programa, é definida uma unidade de tempo denominada *tick*, que irá corresponder a 10 pulsos do clock (unidade utilizada para definir o tempo de espera de um processo num comando *sleep*). Assim, será definida uma função *dorme*, onde a unidade de espera de cada processo será o *tick*. Cada processo, ao acionar a função *dorme* vai informar a quantidade de *ticks* durante o qual vai dormir. O valor da quantidade de *ticks* é gerado no interior de cada processo, de modo aleatório, entre 1 e 15. Ao ser ativado, cada processo define o seu tempo de espera e chama a função *dorme*. Um processo servidor irá controlar o tempo de espera de cada processo, sendo que, a cada *tick*, vai verificar se algum processo deve ser acordado... se for o caso, ele acorda o processo.

```
PROGRAM ServerTicks;
CONST ntproc = 15;
VAR
    vticks : ARRAY [ 1 .. ntproc ] of INTEGER;
    vspriv : ARRAY [ 1 .. ntproc ] of SEMAPHORE;
    mutexind : SEMAPHORE;
{=====}
PROCEDURE dorme ( pid: INTEGER; nticks : INTEGER );
BEGIN
    wait ( mutexind );
    vticks [ pid ] := nticks;
    writeln(pid:3, ' dormira por: ', nticks:4 , 'Ticks');
    signal ( mutexind );
```

```

wait ( vspriv [ pid ] );
END;
{=====}
PROCEDURE acorda;
VAR ind : INTEGER;
BEGIN
  wait ( mutexind );
  FOR ind := 1 TO ntproc DO
    IF ( vticks [ ind ] > 0 )
      THEN
        BEGIN
          vticks [ ind ] := vticks [ ind ] - 1;
          IF ( vticks [ ind ] = 0 ) THEN
            BEGIN
              signal ( vspriv [ ind ] );
              writeln(' > ', ind:4, ' sendo acordado');
            END;
          END;
        signal ( mutexind );
      END;
END;
{=====}
PROCESS TYPE tpClient ( pid : INTEGER );
VAR
  nt: INTEGER;
BEGIN
  REPEAT
    nt := random ( 15 ) + 3;
    dorme ( pid, nt );
  FOREVER;
END;

PROCESS Server;
BEGIN
  REPEAT
    sleep ( 10 ); { server dormiu por 1 tick }
    acorda;
  FOREVER;
END;

VAR
  Client : ARRAY [ 1.. ntproc ] OF tpClient;
  i      : INTEGER;
BEGIN
  FOR i := 1 TO ntproc DO
    BEGIN
      vticks [ i ] := 0;
      initial ( vspriv [ i ] , 0 );
    END;
    initial ( mutexind , 1 );
  COBEGIN
    FOR i := 1 TO ntproc DO Client [ i ] ( i );
    Server;
  COEND;
END.

```

3.2 Jantar dos Filósofos

O Jantar dos Filósofos é um problema acadêmico clássico da Programação Concorrente. Conforme mostrado na figura abaixo, o problema resume-se em garantir que os cinco filósofos sentados em volta da mesa possam pensar e, quando estiverem com fome, alimentarem-se com o auxílio de dois garfos posicionados dos dois lados de seu prato. Como se pode verificar na figura, um filósofo, ao pegar um dos garfos, inviabiliza a refeição de um dos filósofos ao seu lado, de modo que, quando um filósofo está comendo, os outros dois imediatamente ao seu lado poderão alimentar-se. Pode-se ter mais de um filósofo comendo ao mesmo tempo, desde que estes não estejam lado a lado. Isto que significa que apenas dois filósofos poderão estar comendo simultaneamente. Além disso, deve-se tomar cuidado para que vários filósofos não peguem o mesmo garfo ao mesmo tempo (todos pegam o garfo à sua direita ou à sua esquerda). Se isto ocorrer, o resultado é bloqueio mortal.

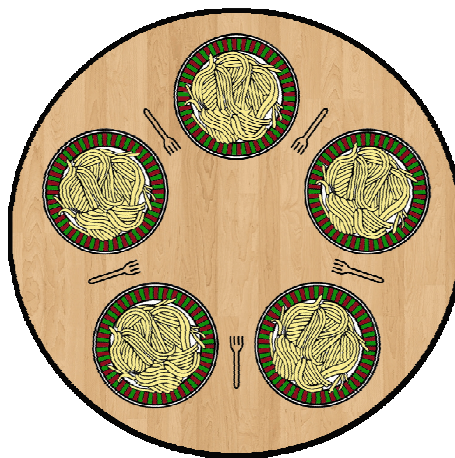


Figura 1 – Mesa do jantar dos 5 filósofos.

```

PROGRAM JANTARSENDREC;

CONST GARCOM = 6;

TYPE TPMSG = RECORD
    FONTE : INTEGER;
    PEDIDO: CHAR;
END;

VAR
    CXPOSTAL      : ARRAY [ 1..6 ] OF TPMSG;
    VSVAGA, VSMSG: ARRAY [ 1..6 ] OF SEMAPHORE;

PROCEDURE SEND( DEST: INTEGER; MSG: TPMSG );
BEGIN
    WAIT ( VSVAGA[ DEST ] );
    CXPOSTAL[ DEST ]:= MSG;
    SIGNAL( VSMSG [ DEST ] );
END;

PROCEDURE RECEIVE( PID: INTEGER; VAR MSG: TPMSG );
BEGIN

```

```

WAIT ( VSMMSG [ PID ] );
MSG:= CXPOSTAL[ PID ] ;
SIGNAL( VSVAGA[ PID ] );
END;

PROCESS TYPE TPFILO ( FID: INTEGER );
VAR
  MSG: TPMSG;
BEGIN
  REPEAT
    SLEEP( RANDOM( 15 ) + 3 );      (*PENSANDO*)

    MSG.FONTE := FID;
    MSG.PEDIDO:= 'P';

    SEND( GARCOM, MSG );
    RECEIVE( FID, MSG );

    SLEEP( RANDOM( 10 ) + 5 );      (*COMENDO*)

    MSG.FONTE := FID;
    MSG.PEDIDO:= 'L';

    SEND( GARCOM, MSG );
  FOREVER;
END;(*FILOSOFO*)

PROCESS SERVER;
VAR
  MSG : TPMSG;
  G,D,E: ARRAY [ 1..5 ] OF INTEGER;
  ST : ARRAY [ 1..5 ] OF CHAR;
  I, FID, DFID, EFID: INTEGER;
  OP: CHAR;

PROCEDURE PRINTST;
VAR
  I: INTEGER;
BEGIN
  FOR I:= 1 TO 5 DO WRITE ( ' ', ST[ I ] );
  WRITELN;
END;

BEGIN
  FOR I:= 1 TO 5 DO
    BEGIN
      G [ I ]:= 2;
      D [ I ]:= I + 1;
      E [ I ]:= I - 1;
      ST[ I ]:= 'P';
    END;
  D[ 5 ]:= 1;
  E[ 1 ]:= 5;
  PRINTST;

```

```

BEGIN
  REPEAT
    RECEIVE ( GARCOM, MSG );
    FID := MSG.FONTE;
    DFID:= D [ FID ];
    EFID:= E [ FID ];

    OP  := MSG.PEDIDO;
    CASE OP OF
      'P':
        BEGIN
          IF G[FID] = 2
            THEN
              BEGIN
                G[D[FID]] := G[D[FID]] - 1;
                G[E[FID]] := G[E[FID]] - 1;
                MSG.FONTE := GARCOM;
                SEND(FID,MSG);
                ST[FID] := 'C';
                PRINTST;
              END
            ELSE
              BEGIN
                ST[FID] := 'F';
                PRINTST;
              END;
            END;
          END;
        'L': BEGIN
          G[D[FID]] := G[D[FID]] + 1;
          G[E[FID]] := G[E[FID]] + 1;
          ST[ FID ] := 'P';
          PRINTST;
          FOR I:= 1 TO 5
            DO
              BEGIN
                IF ((ST[I] = 'F') AND (G[I] = 2 ))
                  THEN
                    BEGIN
                      G[D[I]] := G[D[I]] - 1;
                      G[E[I]] := G[E[I]] - 1;
                      MSG.FONTE := GARCOM;
                      SEND(I,MSG);
                      ST[I] := 'C';
                      PRINTST;
                    END;
                  END;
                END;
              END;
            END; (*CASE*)
          FOREVER
        END;
      END;
    VAR
      FILO : ARRAY [ 1..5 ] OF TPFIL0;
      F     : INTEGER;
    BEGIN

```

```
FOR F := 1 TO 6 DO
  BEGIN
    INITIAL ( VSVAGA[ F ], 1 );
    INITIAL ( VSMMSG [ F ], 0 );
  END;
COBEGIN
  FOR F := 1 TO 5 DO FILO [ F ]( F );
  SERVER;
COEND;
END.
```

4. EXERCÍCIO

4.1 Alocação de recursos

Considere um sistema onde dois processos competem pela utilização de dois recursos: uma impressora e um disco rígido. Os dois processos possuem comportamento idêntico, de modo que os dois precisam alocar os dois recursos para poderem realizar um dado processamento. A ordem de alocação dos recursos pode ser qualquer (impressora-disco ou disco-impressora). O processamento só é realizado quando um deles obteve os dois recursos. Após o processamento, cada processo libera os dois recursos (em qualquer ordem também) deixando-os à disposição do outro processo ou dele próprio para um processamento posterior. Realize um programa utilizando semáforos que represente a solução deste problema.

Dica: Utilize comandos de impressão para informar a alocação dos recursos e o processamento.