O Problema dos Leitores & Escritores - Solução com Semáforos[1]

Alexandre Sztajnberg

PEL - UERJ

1. Introdução

O problema dos Leitores & Escritores é uma generalização do problema básico de exclusão mútua no acesso a recursos compartilhados por processos diferentes. Este problema é introduzido em [Ben-Ari 82] e a solução apresentada é baseada na abstração de **monitores**. Os monitores apresentam como característica interessante para implementar certas propriedades de intercalamento, justiça e prioridades filas, FIFO associadas ao próprio monitor e às variáveis de condição.

A conversão da solução baseada em monitores para uma ambiente onde somente a abstração de **semáforos** estejam disponíveis é factível, mas requer bastante atenção. Através de **semáforos** binários (que só assumem valores os 0 e 1), obviamente, podemos gerenciar a exclusão mútua a recursos compartilhados (que seria equivalente a *entrar no monitor*). Para simular as **primitivas de condição** dos monitores utilizam-se, para cada condição ***cond***, por exemplo uma variável de contagem, ***cond-count*** para contabilizar quantos processos estão bloqueados a espera da condição ser satisfeita e um semáforo binário ***cond-sem*** para efetivamente bloquear e liberar estes processos. Estes elementos adequadamente arranjados podem simular os monitores. Na realidade arranjo semelhante a este é normalmente apresentado na literatura antes da técnica de monitores na solução do problema clássico do **Barbeiro Dorminhoco**.

A única característica com a qual os semáforos clássicos não são apresentados normalmente, são as filas FIFO associadas aos monitores. Tanto no acesso ao monitor, quanto no acesso aos recursos compartilhados liberados por uma condição, os processos são atendidos na ordem de chegada. Entretanto, no conceito clássico de semáforos apresentado em [Ben-Ari 76] os processos suspensos em um semáforo podem ser liberados em uma ordem arbitraria escolhida na sorte. Por outro lado em [Leão 96] é sugerida a implementação de semáforos já contemplando as filas FIFO para disciplinar a liberação de processos suspensos.

Observa-se finalmente que na implementação de semáforos apresentados na biblioteca IPC - Interprocess Comunication do System V do sistema operacional Solaris da Sun é seguida a orientação clássica de que um dos processos suspensos é escolhido na sorte para ser liberado, dado que seja possível (o valor do semáforo foi incrementado tornando-se > 0).

1.1 Formulação Geral do Problema

Existe um recurso comum, por exemplo (uma conta bancária, por exemplo) e vários processos podem tentar acessar este recurso para escrita/atualização (seria equivalente a depósitos e saques na conta-corrente) ou leitura (o que corresponderia a consulta de saldo ou extrato bancário da conta corrente). Para esta finalidade existem dois tipos de processos: os **escritores**, que podem escrever no recursos e os **leitores**, que apenas consultam os recursos. Processos Escritores e Leitores, escrevem e lêem em um tempo finito e podem repetir estas ações em seguida, guardado um intervalo de tempo finito que pode ser nulo.

---

[1] Trabalho originalmente desenvolvido na disciplina de Programação de Tempo Real, prof. Jorge Leão, COPPE, em 1996.

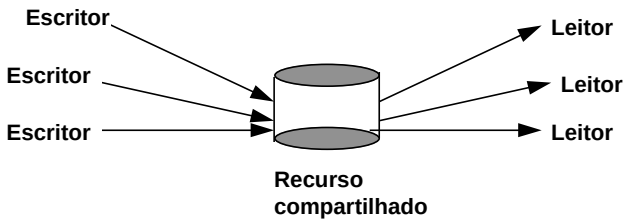

Figura 1 - cenário genérico dos Leitores & Escritores

Existe uma série de propriedades que deve ser associada a este sistema e será descrita a seguir. Ressalta-se ainda que o esquema de prioridade ou justiça que será também apresentado é específico para o sistema alvo, não havendo uma regra geral para a aplicabilidade ou implementação deste esquema. Por exemplo, dentre as propriedades exigidas no presente problema, vários leitores podem consultar o recursos compartilhado ao mesmo tempo, mas nada impediria que em um outro problema fosse exigida a exclusão mútua entre leitores.

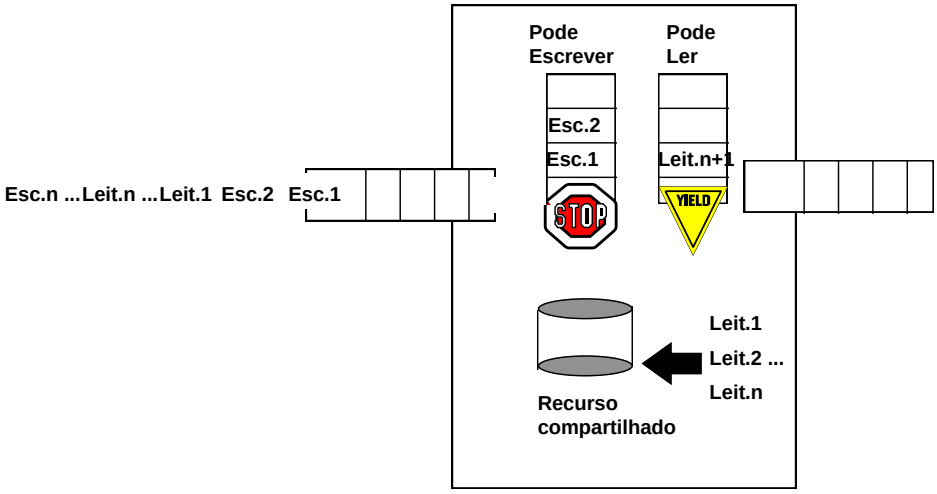

Figura 2 - Solução com monitores

2. Propriedades do Problema em Linguagem Natural

F.1) Deve haver exclusão mútua para os escritores, tanto em relação a leitores quanto a outros escritores.

F.2) Exclusão mútua entre leitores não é necessária.

F.3) Sejam as variáveis Escrevendo e NumeroLeitoresLendo, a propriedade de exclusão mútua pode ser indicada por:

if NumeroLeitoresLendo > 0 then Escrevendo = FALSE and

if Escrevendo = TRUE then NumeroLeitoresLendo = 0

F.4) Havendo leitores em ação, e chegando um escritor para esperar para começar a escrever, todos os novos leitores que chegarem para começar a ler ficarão suspensos, e quando o último leitor em ação terminar, será dada a vez a um escritor (ao primeiro) do conjunto (da fila) de escritores suspensos.

F.5) Se algum processo está utilizando o recurso comum e chega um escritor, este também ficará suspenso numa fila de escritores.

F.6) Havendo um escritor em ação (Escrevendo = TRUE), e existem leitores e escritores esperando, ao término do processo de escrita a vez será dada a todos os leitores que estiverem suspensos e o

próximo escritor a escrever só o fará depois que todos os leitores que estiverem esperando terminarem de ler.

F.7) Não havendo situação de competição, qualquer processo poderá acessar o recurso comum.

3. pseudocódigo dos Programas

Serão apresentados apenas os pseudocódigo dos leitores e escritores. Este pseudocódigo foi extraído do programa PV02.PAS apresentado em [Leão]. O código para a criação e liberação dos recursos compartilhados como semáforos e memória, dependentes do sistema operacional serão apresentados apenas na listagem dos programas no Anexo A.

Observações a respeito do pseudocódigo:

na solução do problema com semáforos são utilizados 3 semáforos:

ExM	semáforo binário, que garante exclusão mútua para cada tentativa de acesso aos recursos, tanto para a posse quanto para a liberação do mesmo. Faz o papel da entrada no monitor.
PodeEscrever	semáforo binário que, juntamente com o restante das variáveis permite que um escritor, por vez tenha acesso exclusivo ao recursos comum
PodeLer	semáforo binário que, juntamente com o restante das variáveis permite que vários leitores possam ler, excluindo qualquer escritor do acesso ao recurso comum neste momento

O protocolo entre leitores e escritores é apoiado por 4 variáveis que trabalham em conjunto com os semáforos:

Leitores	indica o número de leitores lendo efetivamente (o que só ocorre quando não existe nenhum escritor escrevendo)
Escrevendo	assumo o valor TRUE quando existe um escritor efetivamente escrevendo e FALSE em caso contrário
SuspensosPodeLer	contabiliza quantos leitores estão suspensos no semáforo PodeLer e é utilizado para o esquema de prioridade
SuspensosPodeEscrever	contabiliza quantos escritores estão suspensos no semáforo PodeEscrever e é utilizado para o esquema de prioridade

Todos as variáveis e semáforos são iniciadas com o valor 0 (ou FALSE) com exceção do semáforo ExM.

As linhas 49-50 e 54-57 funcionam com uma cadeia de operações de *signal* no semáforo *PodeLer*, de forma que quando um dos Leitores consegue começar a ler, este sinaliza para outro leitor que esteja aguardando no semáforo *PodeLer* que também faz a mesma seqüência. Assim, todos os leitores que estejam suspensos começam a ler quase juntos.

As linhas contendo **if** implementam o protocolo de prioridades P.3, P.4 e P.5. Por exemplo, a linha 13 do procedimento *PodeEscrever* indica que quando um escritor deseja começar a escrever, ele primeiro verifica se existe um outro escritor escrevendo (*Escrevendo == TRUE)* ou se existem um ou mais leitores lendo (*Leitores > 0*). Assim sendo, o próprio escritor se suspende esperado permissão para escrever (*PodeEscrever*).

Foram destacados por legendas do tipo <<**TX**>> o início das transições em cada tipo de processo. Estas transições podem ser também observadas na seção onde são apresentadas as redes de Petri Condição/Ação.

```
{*******************************************************}
{* Pseudocódigo extraído do programa PV02.PAS          *}
{*******************************************************}
1:  program LeitoresEscritores;

2:  Var
3:  ExM,
4:  PodeEscrever,
5:  PodeLer                : semaphore;

6:  Escrevendo             : boolean;

7:  Leitores, Escritores,
8:  SuspensosPodeEscrever,
9:  SuspensosPodeLer       : integer;

     {-----------------------------------------------------}
10:  procedure ComecarEscrever;
11:  <<T1>> begin
12:    wait(ExM);
13:  <<T2>>  if (Escrevendo or (Leitores>0)) then
14:    begin
15:      inc(SuspensosPodeEscrever);
16:      signal(ExM);
17:  <<T3>>    wait(PodeEscrever);
18:      dec(SuspensosPodeEscrever);
19:      Escrevendo:=true; inc (Escritores);
20:      signal(ExM);
21:    end
22:    else
23:  <<T4>>  begin
24:      Escrevendo:=true; inc (Escritores);
25:      signal(ExM);
26:    end;
27:  end;

     {-----------------------------------------------------}
28:  procedure AcabarEscrever;
29:  <<T5>> begin
30:    wait(ExM);
31:    Escrevendo:=false;
32:  <<T6>>  if (SuspensosPodeLer>0) then
33:      signal(PodeLer)
34:  <<T7>>  else if (SuspensosPodeEscrever>0) then
35:      signal(PodeEscrever)
36:  <<T8>>  else
37:      signal(ExM);
38:  end;
     {-----------------------------------------------------}
39:  procedure ComecarLer;
40:  <<T9>> begin
41:    wait(ExM);
42:  <<T10>>  if (Escrevendo or (SuspensosPodeEscrever > 0)) then
43:    begin
44:      inc(SuspensosPodeLer);
45:      signal(ExM);
46:  <<T11>>    wait(PodeLer);
47:      dec(SuspensosPodeLer);
48:      inc(Leitores);
49:  <<T12>>    if (SuspensosPodeLer > 0) then
50:        signal(PodeLer)
51:  <<T13>>    else
52:        signal(ExM);
53:    end
```

```
54:    else
55:  <<T14>>  begin
56:       inc(Leitores);
57:       signal(ExM);
58:    end;
59:  end;

     {-----------------------------------------------------}
60:  procedure AcabarLer;
61:  <<T15>> begin
62:    wait(ExM);
63:    dec(Leitores);
64:  <<T16>>  if ((Leitores=0) and (SuspensosPodeEscrever>0)) then
65:       signal(PodeEscrever)
66:  <<T17>>  else
67:       signal(ExM);
68:  end;

     {-----------------------------------------------------}
69:  procedure Escritor;
70:  begin
71:    repeat
72:      ComecarEscrever;
73:      Escrever;
74:      AcabarEscrever;
75:    until false;
76:  end;

     {-----------------------------------------------------}
77:  procedure Leitor;
78:  begin
79:    repeat
80:      ComecarLer;
81:      Ler;
82:      AcabarLer;
83:    until false;
84:  end;

     {----------- Corpo do programa principal -------------}
85:  begin
86:     InitSem (ExM, 1);
87:     InitSem (PodeLer, 0);
88:     InitSem (PodeEscrever, 0);
89:     Leitores:=0;
90:     Escrevendo:=false;
91:     SuspensosPodeLer:=0;
92:     SuspensosPodeEscrever:=0;
93:     Cobegin
94:        Escritor1; ... EscritorN;
95:        Leitor1;... LeitorN;
96:    Coend;
97:  end.
```

4. Tradução das principais propriedades em lógica temporal:

Com o pseudocódigo entendido e com a formulação natural do problema, podemos transformar as propriedades em fórmulas de lógica temporal:

F.1) $(\text{Escrevendo} = \text{TRUE}) \supset ((\text{Leitores} = 0) \wedge (\text{Escritores} = 1))$

para exclusão mútua

F.2) $((\text{Leitores} > 0) \wedge (\text{SuspensosPodeEscrever} = 0)) \supset (\text{SuspensosPodeLer} = 0)$

se existe algum leitor ativo e não existem escritores suspensos aguardando oportunidade para escrever, então não existe motivo para outros leitores estarem esperando para ler, já que leitores não precisam de exclusão mútua entre si.

F.3) $((Leitores > 0) \wedge (SuspensosPodeEscrever > 0)) \supset (\neg\ Escrevendo = TRUE\ until\ (Leitores = 0))$

na situação em que existem leitores ativos e escritores esperando oportunidade para escrever, um escritor não será liberado até que todos os leitores ativos terminem suas atividades.

F.4) $((Escrevendo = TRUE) \wedge (SuspensosPodeLer > 0) \wedge (SuspensosPode\ Escrever > 0)) \supset ((\Diamond (Escrevendo = FALSE)) \wedge ((\neg\ SuspensosPodeLer = 0)\ until\ (Escrevendo = FALSE)))$

aqui temos parte do esquema de prioridades formulado. Se existe um escritor ativo e temos leitores e escritores suspensos, aguardando permissão para agir, então a prioridade será dada aos leitores que só poderão agir quando o escritor terminar.

Podemos capturar, ainda, grande parte das propriedades desejadas no problema na seguinte fórmula:

F.5) $((Leitores > 0) \wedge (SuspensosPodeEscrever > 0) \wedge (SuspensosPodeLer > 0)) \supset ((\neg\ SuspensosPodeLer = 0)\ until\ (Escrevendo = TRUE))$

um outro aspecto do esquema de prioridades. Neste caso, se existem leitores em atividade e escritores e leitores suspensos. Isso significa que os leitores suspensos chegaram em uma hora em que já haviam escritores suspensos. Então os leitores somente terão oportunidade de ler depois que um escritor suspenso conseguir escrever.

Prova informal:

De forma semelhante ao que é apresentado em [Ben-Ari 96] as propriedades traduzidas em lógica temporal serão analisadas de forma informal. Para facilitar a análise e o entendimento das fórmulas estas serão reinterpretadas em linguagem natural quando necessário.

4.1) Exclusão mútua dos escritores com leitores e outros escritores:

Na formula em lógica temporal, temos afirmado que, se tem alguém escrevendo (Escrevendo = TRUE), então não há ninguém lendo (Leitores = 0) e só há uma tarefa escrevendo (Escritores = 1), embora esta variável não esteja modelada é fácil inferir a sua utilidade.

Se há um processo escrevendo (Escrevendo = TRUE), isto significa que o procedimento ComecaEscrever foi completado com sucesso, pois este é o único procedimento que faz Escrevendo = TRUE. Sabendo que inicialmente, a variável Escritores = 0, ao fim de ComecaEscrever teremos Escritores = 1. Na segunda parte da implicação, temos a conjunção:

(a) Leitores = 0; é verdade, pois, caso contrário, ComecaEscrever não seria completada com sucesso, devido ao teste na linha 13 (a tarefa ficaria suspensa em PodeEscrever - linha 15). Por outro lado, após ComecaEscrever ter sido completada, o valor de Leitores não poderia ser incrementado enquanto Escrevendo = TRUE, pois o procedimento ComecaLer faz o teste na variável Escrevendo (linha 42), e caso não tivéssemos TRUE, a tarefa ficaria suspensa em PodeLer (linha 44).

(b) Escritores = 1; enquanto Escrevendo = TRUE, o valor de Escritores, setado para 1 na execução de ComecaEscrever, não poderá ser incrementado (linha 24), devido ao teste na linha 13. Quando Escritores for decrementado (AcabaEscrever, linha 31) já não teremos mais Escrevendo = TRUE .

4.2) Exclusão mútua entre leitores:

O que a expressão em lógica temporal afirma é que, se há alguma tarefa lendo, e não há nenhuma esperando para escrever, então não há também nenhuma esperando para ler. A expressão só é falsa caso tenhamos, nessas condições, alguém na fila de leitores (SuspensoPodeLer > 0).

Isso só ocorreria por conta do esquema de prioridades em que um leitor ao terminar o procedimento ComecaLer, verifica se existem outros leitores suspensos (linhas 49 e 50). Caso positivo, o leitor ativo libera o próximo leitor suspenso (linhas 50 e 57). Isso se dá até que um leitor ativo não encontre mais nenhum leitor suspenso e neste caso apenas sai do *monitor* (signal ExM).

Nesta hora chegamos a situação mais básica em que podem não haver nem escritores suspensos ou nenhum leitor ou escritor chegou.

4.3) Comportamento dos leitores quando da chegada de um escritor durante leitura:

Segundo a expressão F.3, se há leitores em ação e escritores esperando, nenhum leitor vai começar a ler enquanto o primeiro escritor não terminar de escrever. Quando o leitor começa sua execução e encontra SuspensosPodeEscrever > 0, ele próprio se bloqueia no semáforo PodeLer (linhas 42, 43 e 46). Os leitores são liberados pelo escritor ativo de forma prioritária (linha 32). Nesse caso, já teríamos Escrevendo = FALSE (linha 31). Assim, a segunda parte da implicação é satisfeita.

4.4) Comportamento dos leitores e escritores suspensos após o término de um processo de escrita:

Segundo a fórmula F.4, se existem leitores e escritores esperando enquanto uma tarefa esta escrevendo, ao término dessa tarefa não haverá escrita, e assim será até que a última tarefa de leitura termine sua rotina de leitura e não existam mais leitores lendo.

Quando a tarefa escritor que estiver ativa terminar, vai executar AcabaEscrever, onde Escrevendo será levada para FALSE. Neste momento, como existem leitores esperando na condição PodeLer (linha 32), estes serão liberados (linha 33) antes dos escritores na condição PodeEscrever (teste na linha 32). Nesse caso, ComecaEscrever não será executada, e Escrevendo continuará FALSE. Essa situação vai perdurar de forma semelhante ao item anterior, até que o último leitor ativo termine. Assim, a transição de AcabaLer vai levar a Leitores = 0 (68 e o teste da linha 69), e conseqüentemente a uma sinalização para o semáforo PodeEscrever (linha 70), liberando novamente um escritor que por ventura esteja suspenso.

4.5) Comportamento dos leitores e escritores suspensos após a término da atividade de leitores ativos:

O caso semelhante ao item 4.4 se dá aqui havendo somente algumas alterações no esquema de prioridade. A fórmula F.5 nos revela que se existem leitores ativos, os leitores suspensos no semáforo PodeLer não serão liberados (¬SuspensosPodeLer = 0) até que Escrevendo = TRUE.

Escrevendo = TRUE só ocorre quando a tarefa escritor executa ComecarEscrever com sucesso (linhas 19 e 24). Para que isto ocorra, é necessário que algum leitor execute um signal no semáforo PodeEscrever, liberando um escritor suspenso. Como visto em 4.4 e 4.3 isso é feito pelo último leitor ativo que está terminando sua atividade (linha 70) e assim sendo os leitores suspensos ainda permanecerão nesta condição até que o escritor termine, recaindo no cenário do item 4.4..

5. Verificação da Validade das fórmulas pelo método da Verificação do Modelo (*Model Checking*)

5.1 Rede Condição-Ação

Verifica-se que por simplicidade foram omitidos os passos de inicio dos recursos e no modelo do escritor e do leitor não são consideradas as fases de escrita e leitura por não ser relevante na verificação do modelo. Seria simples incluir mais um lugar e uma transição para representar a escrita no recurso e a consulta ao mesmo, mas o resultado seria apenas o aumento na árvore de marcações.

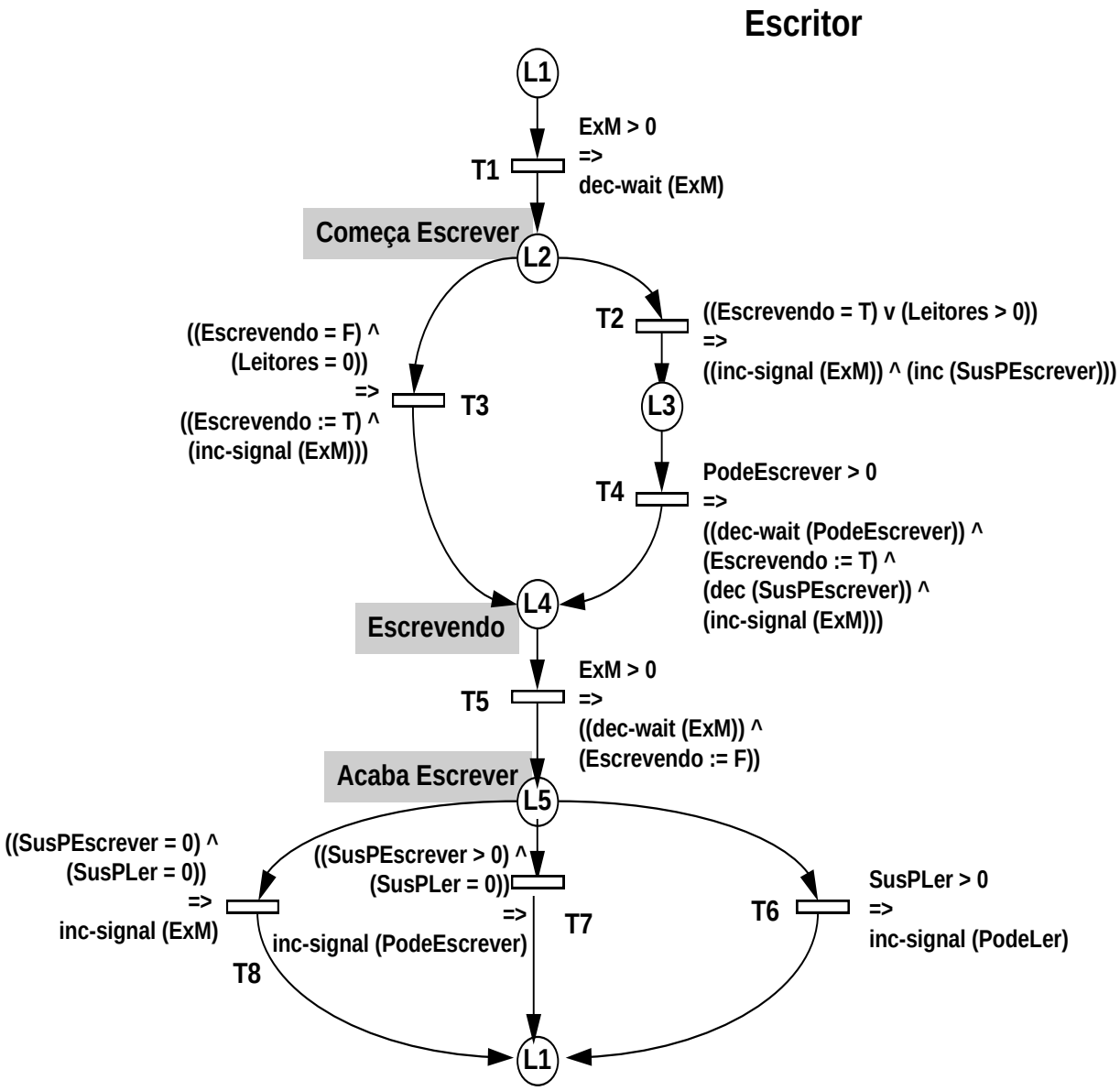

Figura 3 - Rede de Petri Condição/Ação para um Escritor

Observações com relação às redes apresentadas:

- os pares condição/ação do tipo semáforo > 0 => dec-wait(semáforo) indicam uma operação de wait no semáforo bem sucedida, ou seja, o semáforo estava com valor 0, foi incrementado por um signal (inc-signal) e em seguida o wait decrementou novamente o seu valor. Esta é a condição para o disparo da transição.

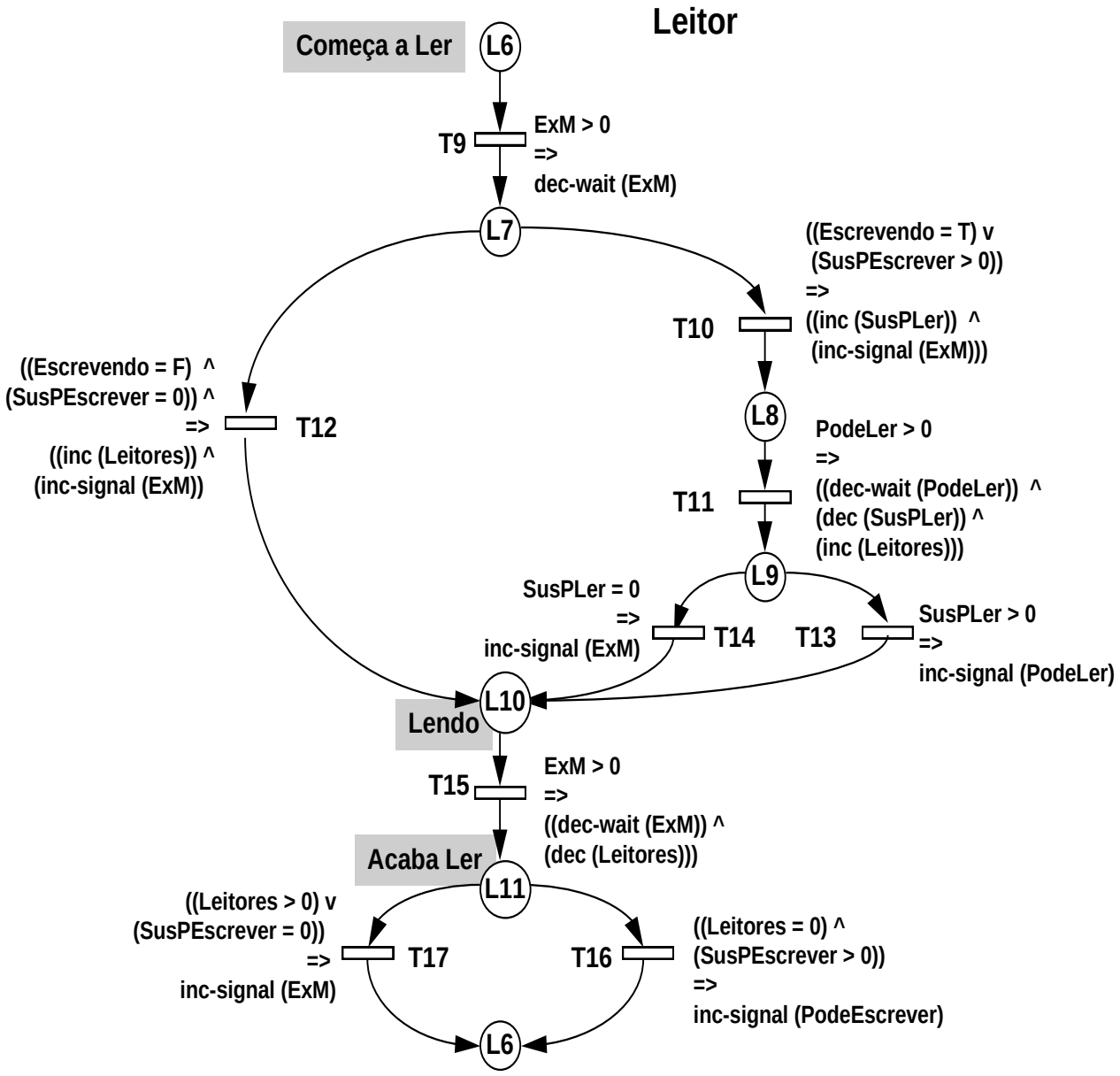

Figura 4 - Rede de Petri Condição/Ação para um Leitor

5.2 Árvore de Alcançabilidade

Para a árvore de alcançabilidade e verificação do modelo seria considerado inicialmente o caso de um escritor e dois leitores. Portanto um estado da árvore será representado pela seguinte tupla:

< LE, LL1, LL2, ExM, PodeEscrever, PodeLer, Escrevendo, Leitores, SusPEscrever, SusPLer>

onde:

LE: representa o lugar na rede onde se encontra o escritor em dada marcação;

LL1: representa o lugar na rede onde se encontra o leitor número 1 em dada marcação;

LL2: representa o lugar na rede onde se encontra o leitor número 2 em dada marcação.

o restante dos elementos da tupla tem o mesmo significado descrito anteriormente. Ressalta-se ainda que embora não tenha sido considerado até aqui a implementação de semáforos com fila FIFO, os elementos

SusPEscrever e SusPLer serão representados pela fila de escritores e leitores, respectivamente suspensos nos semáforos e não somente o valor da variável. Desta forma pode ser avaliado o esquema de prioridade com justiça.

A geração da árvore de alcançabilidade para esta situação é bastante trabalhosa se feita manualmente e a possibilidade de erros ou enganos é alta. Este cenário poderia alcançar a marca de 50 nós na árvore. Assim sendo, um cenário com apenas um escritor e um leitor foi considerado para a geração manual da árvore de marcações.

Marcação	LE	LL1	LL2	ExM	PodeEsc	PodeLer	Escrevendo	Leitores	SupPEsc	SusPLer	Próximo
M0	L1	L1	L1	1	0	0	F (0)	0	-	--	M1, M2, M3
M1	L2	L1	L1	0	0	0	F (0)	0	-	-	M4
M2	L1	L2	L1	0	0	0	F (0)	0	-	-	M6
M2	L1	L1	L1	0	0	0	F (0)	0	-	-	
M4	L4	L1	L1	1	0	0	T (1)	0	-	-	M5, M8
M5	L5	L1	L1	0	0	0	F (0)	0	-	-	M0
M6	L1	L5	L1	1	0	0	F (0)	1	-	-	M7
M7	L1	L6	L1	0	0	0	F (0)	0	-	-	M0
M8	L4	L2	L1	0	0	0	T (1)	0	-	-	M9
M9	L4	L3	L1	1	0	0	T (1)	0	-	L1	M10
M10	L5	L3	L1	0	0	0	F (0)	0	-	L1	M11
M11	L1	L3	L1	0	0	1	F (0)	0	-	L1	M12
M12	L1	L4	L1	0	0	0	F (0)	1	-	-	M13
M13	L1	L5	L1	1	0	0	F (0)	1	-	-	M7, M15
M14 = M7	L1	L6	L1	0	0	0	F (0)	0	-	-	M0
M15	L2	L5	L1	0	0	0	F (0)	1	-	-	M16
M16	L3	L5	L1	1	0	0	F (0)	1	E	-	M17
M17	L3	L6	L1	0	0	0	F (0)	0	E	-	M18
M18	L3	L1	L1	0	1	0	F (0)	0	E	-	M4
M19 = M4	L4	L1	L1	1	0	0	T (1)	0	-	-	
										-	

Tabela 1 - Marcações obtidas manualmente

Observa-se que mesmo na situação simplificada de um processo leitor e um escritor número de estados cresce rapidamente. Na tabela acima os estados representam uma execução em que o escritor executa sozinho, em seguida o leitor executa sozinho e na seqüência existe uma situação em que a exclusão mútua é necessária. A arvore de estados com estas seqüências é apresentada a seguir.

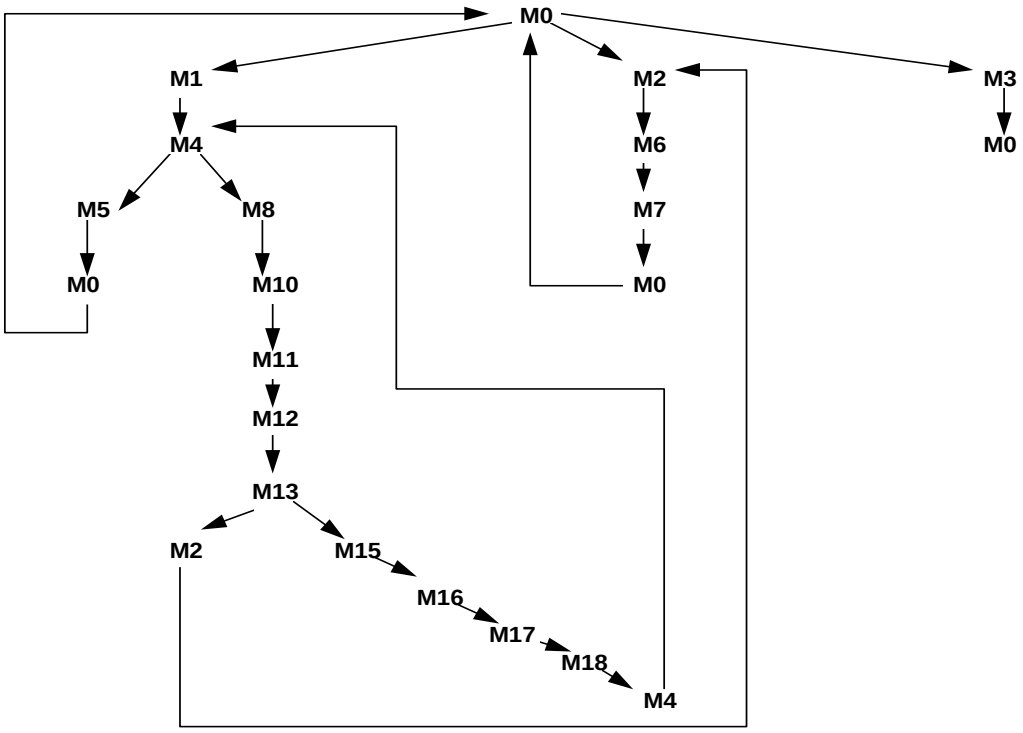

Figura 5 - Árvore parcial de marcações para um escritor e um leitor obtida manualmente

5.3 Grafo de Estados

Para que o método da Verificação do Modelo possa ser utilizado é necessário que o grafo esteja completo. Para que esta condição fosse satisfeita foi fornecida recentemente uma ferramenta que permite gerar a árvore de marcações de forma automática. A seguir são apresentados os resultados obtidos pela ferramenta modelando-se um escritor e dois leitores:

Resultado 1: apresentado na próxima página. O semáforo não é modelado com filas FIFO

Resultado 2: obtido com um escritor e dois leitores. Semáforos com filas FIFO. O grafo resultante possui 370 estados. (Ficando bastante evidente a complexidade de se obter manualmente o grafo e até mesmo a verificação do modelo, ainda que o grafo esteja disponível).

```
 Nome do arquivo : lesc1-2.dat
 2 leitores e 1 escritor
 13 seg num 486 66MHz
 378 estados
 736 nos alocados


Obs.: amostra de 2 estados.

 1.[(Pi[1]=1)(Pi[2]=6)(Pi[3]=6)][(V[1]=1)(V[2]=0)(V[3]=0)(V[4]=0)(V |
0)(V[6]=0)(V[7]=0)][(Fila[1]=<0 >)(Fila[2]=<0 >)(Fila[3]=<0
>)][(T:3,P:1,S:662)(T:17,P:2,S:18)(T:17,P:3,S:4)]

 4.[(Pi[1]=1)(Pi[2]=6)(Pi[3]=7)][(V[1]=0)(V[2]=0)(V[3]=0)(V[4]=0)(V |
0)(V[6]=0)(V[7]=0)][(Fila[1]=<0 >)(Fila[2]=<0 >)(Fila[3]=<0
>)][(T:1,P:1,S:60)(T:15,P:2,S:34)(T:20,P:3,S:7)]
```

O problema da explosão de estados na verificação de sistemas concorrentes fica ainda mais evidente com a apresentação dos resultados acima. Por exemplo, seria conveniente que as variáveis importantes em cada estado fossem representadas diretamente em cada estado e não apenas o seu identificador. Mesmo que se refinasse visualmente a posição de cada nó e a rota dos arcos fosse otimizada para *despoluir* o desenho, este ainda seria complicado.

Para efeitos da verificação do modelo será utilizado o exemplo de um escritor e dois leitores onde os semáforos não são modelados com filas FIFO. O grafo de estados deste modelo é apresentado a seguir.

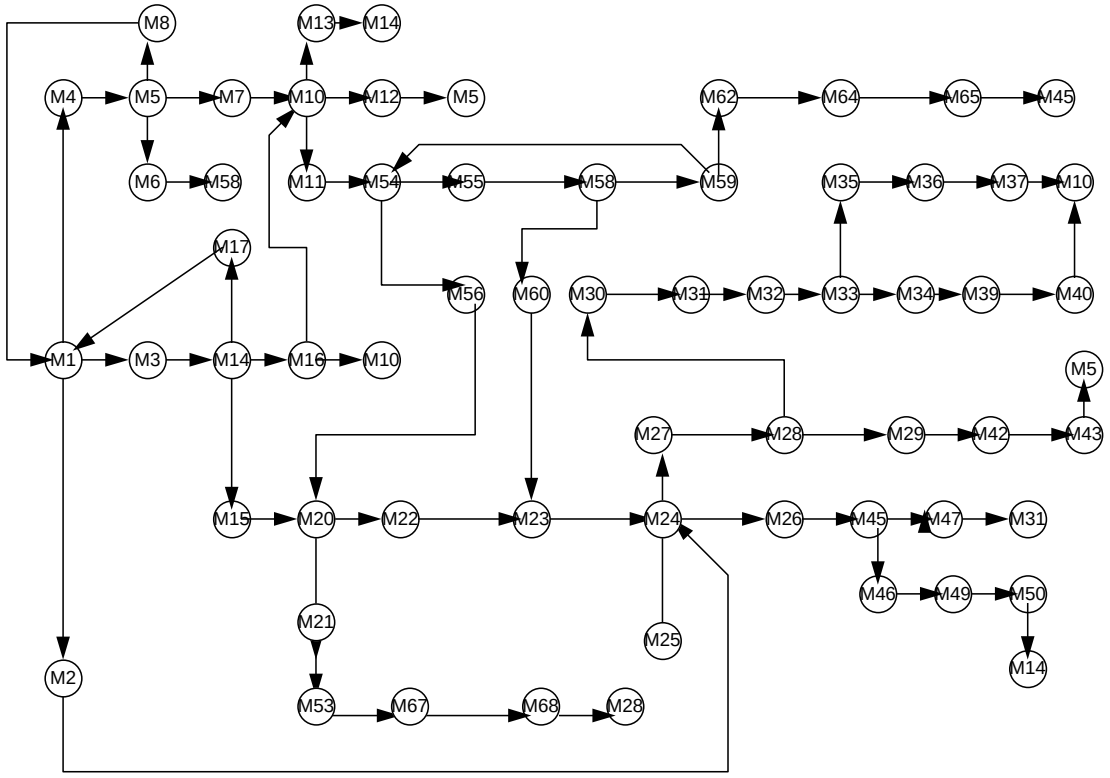

Figura 6 - Grafo de estados para um escritor e um leitor modelados com semáforos com filas FIFO.

5.4. Verificação da Validade das Fórmulas no Conjunto de Computações do Sistema

Pela tabela de marcações algumas propriedades podem ser facilmente verificadas, embora a utilização do método da Verificação do Modelo não seja simples neste caso. Observa-se claramente que a exclusão mútua entre escritores e leitores e, entre os próprios escritores é garantida. Basta *varrer* a tabela de marcações e verificar que em nenhuma marcação existe um escritor ativo (V[4] = 1) e um ou mais leitores ativos (V[5] > 0). Igualmente, não existe nenhuma marcação cujo número de escritores seja maior do que 1 (V[4] > 1), o que é trivial no caso de um leitor e um escritor, mas que é confirmado para os casos mais gerias. Além disso, as outras propriedades estabelecidas nas fórmulas em lógica temporal também podem ser intuitivamente verificadas analisando a tabela de marcações.

Para que a verificação do modelo fosse feita formalmente as fórmulas de lógica temporal apresentadas anteriormente perderam a generalidade e foram adaptadas para o caso específico de um escritor (Escr.) e dois leitores (Leit1 e Leit2). A fórmula F.5 foi escolhida como esta finalidade e assume a seguinte forma:

((Leit 1 = Lendo) $\wedge$ (Esc. = Suspenso) $\wedge$ (Leit 2 = Suspenso)) $\supset$ ((¬ Leit 2 = Lendo) UNTIL (Esc. = AcabarEscrever))

Esta fórmula está direcionada para o modelo alvo e as condições da mesma não se restringem as variáveis do pseudocódigo (como por exemplo, SuspensosEmPodeLer > 0 ou Lendo = 0, etc.), e utiliza lugares específicos da rede condição-ação para este caso. A fórmula pode ser representada como uma árvore de subfórmulas.

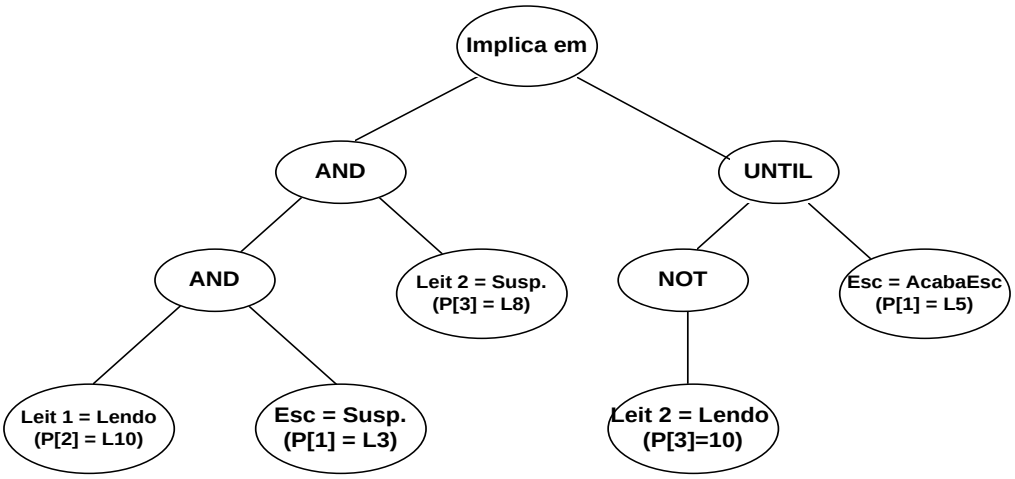

Figura 7 - Fórmula de lógica temporal verificada

Em seguida é apresentada a tabela com as várias fases da solução manual da verificação do modelo. Cada coluna tem marcadas as células cujo estado correspondente é válido Observe que a última coluna da tabela está toda marcada, indicando que a fórmula F.5 é válida para todos os estados do grafo (o Universo) e, portanto a fórmula é válida no modelo.

6. Bibliografia:

[Leão 96] Leão, J.L.S, "Programação e Verificação de Sistemas Multitarefa, COPPE/UFRJ - PEE, V96.1, 1996

[Ben-Ari 82] Ben-Ari, M., "Principles of Concurrent Programming", Prentice-Hall Int., EUA, 1982

Anexo - Listagem dos Programas Fonte em Linguagem C para o sistema Solaris

A implementação consta basicamente de 4 programas em C.

O programa **cria** aloca os recursos compartilhados usando o IPC do System V (semáforos e memória compartilhada). Inicialmente é uma chave default é utilizada e armazenada em dois arquivos (**sema.key** e **memo.key**) para consulta por outros programas. No caso da chave já estar sendo usada o usuário é avisado e uma chave deve ser indicada pelo teclado.

O programa **mata** simplesmente libera os recursos compartilhados.

O programa **leitor** e **escrito**r implementam respectivamente os processos de leitores e escritores. Ao iniciar cada programa destes o usuário pode optar por uma execução **manual**, em que existem vários pontos de parada do programa a espera de um comando do usuário para dar continuidade; ou de forma **automática** em que 50 transações são feitas em seqüência**.**

Observa-se que foi dada grande ênfase à parte de controle do sistema. As transações bancárias foram programadas de forma bastante simples havendo uma transação de deposito na conta corrente (**conta_corrente.dbf**) por conta dos escritores e a consulta contendo um número variável de transações feitas pelos leitores.

Várias mensagens são exibidas durante a execução e adicionalmente é criado um arquivo de log (**escritor.log** e **leitor.log**) que registra em arquivo o **status** dos recursos importantes, como um nó de marcação, em pontos de maior relevância. Desta forma podemos rastrear exatamente o que ocorreu durante uma rodada de execução dos programas.

Uma interface gráfica começou a ser desenvolvida de forma a monitorar em tempo de execução os recursos compartilhados. Este programa ainda não está totalmente operacional mas pode ser ativado (**gstatus**).

A seguir, são listados os arquivos contendo a fonte de cada programa em linguagem C. Adicionalmente estão listados os arquivos **Makefile** com o processo de compilação de todos os arquivos e o **README** e o arquivo de **log** do escritor.

Makefile

```
#      Makefile para os sistema escritores e leitores com semaforo e shmem

CFLAGS= -g
CC = gcc

.KEEP_STATE:

batch: cria mata mata_ant escritor leitor produtor consumidor

cria:   cria.c le.h
       $(CC) -g -o cria  cria.c -lsocket -lm

mata:   mata.c le.h
       $(CC) -g -o mata  mata.c -lsocket -lm

mata_ant:   mata_ant.c le.h
       $(CC) -g -o mata_ant  mata_ant.c -lsocket -lm

leitor: leitor.c le.h
       $(CC) -g -o leitor leitor.c -lsocket -lm

escritor :escritor.c le.h
       $(CC) -g -o escritor  escritor.c -lsocket -lm

produtor: produtor.c le.h
       $(CC) -g -o produtor produtor.c -lsocket -lm

consumidor: consumidor.c le.h
       $(CC) -g -o consumidor  consumidor.c -lsocket -lm
```

README

```
/*
 Autor: Alexandre Sztajnberg
 Data: 28/08/96 (início)

 O arquivo le.h tem as constantes necessárias para o programa

 cria.c - cria semáforo e shared memory
 mata.c - libera os recursos
 escritor.c
 leitor.c

Modo de usar:

1 - execute o CRIA.
2 - chame o ESCRITOR
3 - idem para o LEITOR
4 - ao termino use o MATA para liberar os recursos
```

le.h

```
/*=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-


  =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-*/
/* ============= Bateria de includes ============ */

/*#include <sys/types.h>
#include <sys/socket.h>
#include <sys/syscall.h>
*/
/*#include <sys/ddi.h>
*/
/*
#include <sys/file.h>
#include <sys/errno.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>
*/
/* IPC */

#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

/*
#include <netinet/in.h>
*/
/*
#include <malloc.h>
*/
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <fcntl.h>

#include <errno.h>
#include <signal.h>


/* ============= Constantes ===================*/

#define OK          0
#define ERRO        -1
```

```
#define TRUE        1
#define FALSE       0

#define USED        1
#define UNUSED      0

#define MANUAL      1
#define AUTO        0


#define MAX_ITEM  10 /* define o tamanho do buffer */
#define NUM_SEM       4  /* numero de semáforos usados */
#define STRING_TAM 70 /* tamanho máximo de cada linha do estrato */


/* ======= Semaforos que implementam o monitor de Hoare ======= */
#define ExM               1   /* o 0 fica para o futuro... */
#define PodeEscrever      2
#define PodeLer           3

#define WAIT -1
#define SIGN +1


typedef struct shm_type
{
   int Escrevendo;         /* TRUE se houver alguem escrevendo, FALSE se nao houver */
   int Leitores;           /* #leitores lendo neste momento */
   int SuspensosEmPodeLer; /* #leitores suspensos no semaforo PodeLer */
   int SuspensosEmPodeEscrever; /* #escritores suspensos no semaforo PodeEscrever */
   int count;                   /* contador de linear de transacoes bancarias */
} T_SHM;


/* ========== Arquivos abertos ================= */
#define SHARED_FILE "conta_corrente.dbf"
#define SKEY_FILE "sema_key.key"
#define MKEY_FILE "smem_key.key"
#define ELOG "escritor.log"
#define LLOG "leitor.log"
#define SALDO "saldo.dbf"


/* ============ Variaveis globais =============== */
struct shmid_ds shmid_ds; /* para conter os resultados de smclt */

union semun
{
   int val;
   struct semid_ds *buf;
   ushort * array;
} ;


/*-------------------------------------------------------------
   Funcoes primitivas implementadas

  -------------------------------------------------------------*/
```

cria.c

```
/*-------------------------------------------------------

 Autor: Alexandre Sztajnberg
 Data: 11/12/1992

 Processo principal cria shared memory e semaforo

-------------------------------------------------------*/
```

```
#include "le.h" /* include do sistema, com varios outros includes */

void main ()
{
      int i, b;

      key_t chave_memo = 6622;
      key_t chave_sema = 6621;

      int memo_comp,  /* id da shared memory */
          sema;       /* id da estrutura de semaforos */
      struct sembuf sops; /* estrutura para operacao */
      union semun arg;  /* estrutura para o controle do semaforo */
      int semval;
      struct shmid_ds shmid_ds; /* para conter os resultados de smclt */
      FILE *semf, *smemf; /* arquivos com as chaves de IPC */
      char line [256];

      /* estamos criando novos recursos, portanto: deletar os antigos */

      printf ("CRIA: Alexandre Sztajnberg - Setembro/96\n");
      printf ("CRIA: Cria recursos compartilhados IPC / System V\n");
      printf ("CRIA: Semaforos e Memoria Compartilhada\n");
      printf ("================================================\n");

      sprintf ( line, "rm %s\n", SKEY_FILE );
      system ( line );
      sprintf ( line, "rm %s\n", MKEY_FILE );
      system ( line );

       /* cria a memoria compartilhada (que sera vista pelo pai e filho */

        for (i = 0;
            ((memo_comp = shmget (chave_memo, sizeof (T_SHM), (0666 | IPC_CREAT |
                                                               IPC_EXCL))) == -1);
             i++)
       {
          printf ("CRIA: Erro na criacao de memoria comprtda (key = %d)\n", chave_memo);
          printf ("CRIA: Escolha uma chave alternativa: \n");
          scanf  ("%li", &chave_memo);

          if (i == 9) exit (1);
       }
       printf("CRIA: ID shared mem devolvido %d\n",memo_comp);

        /* salva a chave em um arquivo */
        smemf = fopen (MKEY_FILE, "w");
        fprintf (smemf, "%li", chave_memo);

       /* cria os semaforo */
        for (i = 0;
            ((sema = semget (chave_sema, NUM_SEM, (0666 | IPC_CREAT | IPC_EXCL))) == -1);
             i++)
       {
          printf ("CRIA: Erro na criacao de semaforo (key = %d)\n", chave_sema);
          printf ("CRIA: Escolha uma chave alternativa: \n");
          scanf  ("%li", &chave_sema);

       /* se deu errado, libera a shared memory e sai */

          if (i == 9)
          {
                    if (( shmctl (memo_comp, IPC_RMID , &shmid_ds)) == -1)
                printf ("CRIA: Erro, nao consegui liberar o shm\n");

               exit (2);
          }
        }

       printf("CRIA: ID semaforo devolvido %d\n",sema);

        /* salva a chave em um arquivo */
```

```
        semf = fopen (SKEY_FILE, "w");

        fprintf (semf, "%li", chave_sema);

      /* inicia os semaforos com os valores apropriados */

        /* simula a marcacao inicial da Rede Condicao/Acao */

           arg.val = 1;
           printf ("CRIA: Iniciando semaforo ExM com 1\n");
           semctl (sema, ExM, SETVAL, arg); /* inicializa a exclusao mutua com 1 */

           arg.val = 0;
           printf ("CRIA: Iniciando semaforo PodeEscrever com 0\n");
           semctl (sema, PodeEscrever, SETVAL, arg ); /* inic. semaf de Escrs com 1 */

           printf ("CRIA: Iniciando semaforo PodeLer com 0\n");
           semctl (sema, PodeLer, SETVAL, arg  ); /* inic semaf de Leitores  1 */

}
```

mata.c

```
/*

 Autor: Alexandre Sztajnberg
 Data: 7/9/1996

 */

#include "le.h"

void main ()

{
   key_t chave_memo;
   key_t chave_sema;

   struct shmid_ds shmid_ds; /* para conter os resultados de smclt */

   int memo_comp, sema;

   int j,i,k,t;
   struct sembuf sops;

   union semun arg;  /* estrutura para o controle do semaforo */

   int semval;

   T_SHM *shmem;

   FILE *semf, *smemf; /* arquivos com as chaves de IPC */

      printf ("MATA: Alexandre Sztajnberg - Setembro/96\n");
      printf ("MATA: Programa implementando Escritor\n");
      printf ("MATA: Semaforos e Memoria Compartilhada\n");
      printf ("=======================================\n");


   /* pega as chaves para a memoria e semaforos */

/*   printf ("MATA: chave da memoria compartilha: \n");
   scanf  ("%li", &chave_memo);

   printf ("MATA: chave do semaforo: \n");
   scanf  ("%li", &chave_sema);
*/

   /* le as chaves nos arquivos */

    smemf = fopen (MKEY_FILE, "r");

    fscanf (smemf, "%li", &chave_memo);

    semf = fopen (SKEY_FILE, "r");

    fscanf (semf, "%li", &chave_sema);

   /* pede acesso a memoria compartilhada */

   if ((memo_comp = shmget (chave_memo, sizeof (T_SHM), 0666)) == ERRO)
       printf ("MATA: Erro no acesso da memoria compartilhada (key = %d)\n", chave_memo);
   else
       printf("MATA: ID shared mem devolvido %d\n",memo_comp);

   /* pede acesso aos semaforo */

   if ((sema = semget (chave_sema, NUM_SEM, 0666)) == ERRO)
       printf ("MATA: Erro no acesso aos semaforos (key = %d)\n", chave_sema);
   else
       printf("MATA: ID semaforo devolvido %d\n",sema);
```

```
   /* nao estou tratando se alguma coisa deu errado */

   /* mata os recursos de shared memory */

   if (( shmctl (memo_comp, IPC_RMID , &shmid_ds)) == ERRO)
      printf ("MATA: nao consegui liberar o shm\n");
   else
      printf("MATA: ID shared memory liberada %d\n",memo_comp);

   /* mata os semaforos */

   arg.val = 0;

   if ((semctl (sema, 0, IPC_RMID, arg )) == ERRO)
      printf ("MATA: nao consegui liberar o sem\n");
   else
      printf("MATA: semaforos liberados  %d\n",sema);

   exit (0);
}
```

escritores.c

```
/*

 Autor: Alexandre Sztajnberg
 Data: 28/08/96

*/

#include "le.h"

void print_status (int sema, T_SHM *shmem);

void log (FILE *log, int sema, T_SHM *shmem);

void main ()

{
  int j,i,k,t;

  key_t chave_memo;
  key_t chave_sema;

  int  memo_comp,
       sema;

  struct sembuf sops;
  int semval;

  T_SHM *shmem;     /* variavel que aponta para a area de memoria compartilhada */

  char path[30];   /* possivel string com o path do arquivo - nao deve ser necessario */
  char buf[STRING_TAM + 1]; /* buffer onde sera gravada a trans. banc. antes de gravar */
  int nbyte;       /* numero de bytes a serem gravados */
  int wbyte;       /* numero de bytes gravado */
  FILE *fd;          /* descritor do arquivo comum a ser escrito/lido */

  FILE *semf, *smemf; /* arquivos com as chaves de IPC */

  union semun arg;  /* estrutura para o controle do semaforo */

  char modo, pross;   /* inica o modo manual ou automatico de execucao */
  int bmode;

  char fim;           /* inica o modo manual ou automatico de execucao */
  int bfim = FALSE;

  struct timeval  temp;    /* hora instantanea local */

  unsigned long   tnow;

  FILE *esclog; /* arquivo de log do escritor */

      printf ("ESCR: COE717 - Programacao Tempo Real\n");
      printf ("ESCR: Alexandre Sztajnberg - Setembro/96\n");
      printf ("ESCR: Programa implementando Escritor\n");
      printf ("ESCR: Semaforos e Memoria Compartilhada\n");
      printf ("=======================================\n");

/*
  printf ("ESCR: chave da memoria compartilha: \n");
  scanf  ("%li", &chave_memo);

  printf ("ESCR: chave do semaforo: \n");
  scanf  ("%li", &chave_sema);
*/

    printf ("ESCR: Execucao manual ou automatica: [m/a] \n");
    scanf  ("%c", &modo);

    if (modo == 'm') bmode = MANUAL;
```

```
     else bmode = AUTO;

    /* le as chaves nos arquivos */

     smemf = fopen (MKEY_FILE, "r");

     fscanf (smemf, "%li", &chave_memo);

     semf = fopen (SKEY_FILE, "r");

     fscanf (semf, "%li", &chave_sema);

     esclog = fopen (ELOG, "w");

     fprintf (esclog, "Escritor =====> Inicio da execucao \n");

    /* pede acesso a memoria compartilhada */

    if ((memo_comp = shmget (chave_memo, sizeof (T_SHM), 0666)) == ERRO)
        printf ("ESCR: Erro no acesso da memoria compartilhada (key = %d)\n", chave_memo);
    else
        printf("ESCR: ID shared mem devolvido %d\n",memo_comp);

    /* pede acesso aos semaforo */

    if ((sema = semget (chave_sema, NUM_SEM, 0666)) == ERRO)
        printf ("ESCR: Erro no acesso aos semaforos (key = %d)\n", chave_sema);
    else
        printf("ESCR: ID semaforo devolvido %d\n",sema);

    /* nao estou tratando se alguma coisa deu errado */

   printf ("ESCR: Iniciando escritor... \n");

   printf("ESCR: ID do semaforo do ESCRITOR  %d\n",sema);
   printf("ESCR: ID shared mem devolvido ESCRITOR %d\n",memo_comp);

   shmem = shmat (memo_comp, NULL, 0666);
   printf ("ESCR: Ponteiro para semaforo ESCRITOR %d\n", shmem);

   for (j = 0; ((j < 50) && (bfim == FALSE)); j++)
   {

/* =============== Comeca a Escrever ========================== */

      printf ("ESCR: !!!! Comeca a Escrever !!!!\n");

      if (bmode == MANUAL)
      {
         printf ("ESCR: ATENCAO ! Prossegue [s/n] ? \n");
         scanf ("%c", &pross);
      }

      /* wait (ExM) */

      sops.sem_num = ExM;
      sops.sem_op = WAIT;

      if (( semop (sema, &sops, 1)) == ERRO)
        printf ("ESCR: Erro na operacao wait (ExM)\n");
      else
        printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);


       t=rand (); for (i = 0; i < (t * (j % 10)); i++);

      if ((shmem -> Escrevendo == TRUE) || (shmem -> Leitores > 0))
      {
         shmem -> SuspensosEmPodeEscrever++;
```

```
print_status (sema, shmem);
log (esclog, sema, shmem);

        /* signal (ExM) */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (ExM)\n");
        else
          printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

       /* wait (PodeEscrever) */

        sops.sem_num = PodeEscrever;
        sops.sem_op = WAIT;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (PodeEscrever)\n");
        else
          printf ("ESCR: Operacao wait (PodeEscrever) no semaforo OK\n");

        shmem -> SuspensosEmPodeEscrever--;

        shmem ->  Escrevendo = TRUE;

print_status (sema, shmem);
log (esclog, sema, shmem);

        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (ExM)\n");
        else
          printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

     }
     else
     {
        shmem ->  Escrevendo = TRUE;

        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (ExM)\n");
        else
          printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");
     }

print_status (sema, shmem);
log (esclog, sema, shmem);

     printf ("ESCR: !!!! Fim: Comeca a Escrever !!!!\n");

   /* fim Comeca Escrever */

/* ================ Escrevendo ====================== */
```

```
       printf ("ESCR: !!!! escrevendo ... !!!!\n");

       /* !!!!!!!!!! transcerver o lancamento para o buf, sabendo quantos bytes */

       /* memset (buf, '\n', STRING_TAM + 1); */
       memset (buf, ' ', STRING_TAM + 1);

        /* pega data e hora local para imprimir tambem */

       gettimeofday(&temp, (struct timezone *)0);
       tnow = (unsigned)(temp.tv_sec*1000000)+(unsigned)temp.tv_usec;
       printf("ESCR: iniciando transacao em %u\n", tnow);
       printf("ESCR  DATA %s\n", ctime(&(temp.tv_sec)));


       sprintf (buf, "Transacao numero (%d): deposito de 10,00 ", j);
       strcat (buf,  ctime(&(temp.tv_sec)), 23);


       /* printf ("ESCR: parametros path=%s, nbyte=%d\n", path, nbyte); */

       /* if ((fd = open (SHARED_FILE, O_WRONLY | O_APPEND)) < 0) */
       if ((fd = fopen (SHARED_FILE, "a")) < 0)
       {
          printf ("ESCR_debug: nao consegui fazero o OPEN...\n");
       }
       else
       /* if ((wbyte = write (fd, buf, STRING_TAM + 1)) < 0) */
        if ((wbyte = fprintf (fd,"Transacao numero (%d): deposito de 10,00 em %s", j,
                                                        ctime(&(temp.tv_sec))) ) < 0)
       {
          printf ("ESCR: nao consegui fazero o WRITE...\n      %s\n", buf);
          fclose (fd);
       }
       else
       {
          /* !!!!!!!!!!! mostra na tela os lancamentos */
          printf ("ESCR: lancamento (%d) feito WRITE...\n", j);

          /* incrementa o contador de memoria compartilhada */
          shmem -> count++;

          fprintf (stderr,"Escritor saida do processo contador = %d\n", shmem ->count);
          fclose (fd);
       }

       printf ("ESC: transacao terminada. Status = %d\n", wbyte);
       printf ("ESCR: !!!! Fim: escrevendo ... !!!!\n");

       if (bmode == MANUAL)
       {
           printf ("ESCR: ATENCAO ! Prossegue [s/n] ? \n");
           scanf ("%c", &pross);
       }


/* fim Escrevendo */

/* ========== Acaba Escrever ========================*/

     printf ("ESCR: !!!! Acaba Escrever !!!!\n");

print_status (sema, shmem);
log (esclog, sema, shmem);


   /* wait (ExM) */

     sops.sem_num = ExM;
     sops.sem_op = WAIT;

     if (( semop (sema, &sops, 1)) == ERRO)
```

```
       printf ("ESCR: Erro na operacao wait (ExM)\n");
      else
       printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

     t=rand (); for (i = 0; i < (t * (j % 10)); i++);

     shmem ->  Escrevendo = FALSE;

     if (shmem -> SuspensosEmPodeLer > 0)
     {

print_status (sema, shmem);
log (esclog, sema, shmem);

        /* signal (PodeLer) */ /* !!!!!!!!! */

        sops.sem_num = PodeLer;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (PodeLer)\n");
        else
          printf ("ESCR: Operacao wait (PodeLer) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

     }
     else if (shmem -> SuspensosEmPodeEscrever > 0)
     {
        /* signal (PodeEscrever) */ /* !!!!!!!!! */

        sops.sem_num = PodeEscrever;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (PodeEscrever)\n");
        else
          printf ("ESCR: Operacao wait (PodeEscrever) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

     }
     else
     {
        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("ESCR: Erro na operacao wait (ExM)\n");
        else
          printf ("ESCR: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (esclog, sema, shmem);

     } /* fim Acaba Escrever */

     printf ("ESCR: !!!! Fim: Acaba Escrever !!!!\n");

     if (bmode == MANUAL)
     {
       printf ("ESCR: Atencao: mais uma transacao ?: [s/n] \n");
       scanf  ("%c", &fim);

       if (fim == 'n') bfim = TRUE;
```

```
        else bfim = FALSE;
      }

      if (bmode == MANUAL)
      {
        printf ("ESCR: Chavear execucao manual ou automatica: [m/a] \n");
        scanf  ("%c", &modo);
        if (modo == 'm') bmode = MANUAL;
        else bmode = AUTO;
      }

  } /* fim for */

   printf ("ESCR: !!!! Liberando recursos IPC !!!!\n");

    fprintf (esclog, "Escritor =====> Fim da execucao \n");

   fclose (esclog);

   shmdt ((void *)shmem);

   exit (0);
}

void print_status (int sema, T_SHM *shmem)
{

     printf (" Escrevendo %s ||", shmem -> Escrevendo == TRUE ? "TRUE ": "FALSE");
     printf (" Leitores %d ||", shmem -> Leitores);
     printf (" SuspensosEmPodeLer %d ||",shmem -> SuspensosEmPodeLer);
     printf (" SuspensosEmPodeEscrever %d ", shmem -> SuspensosEmPodeEscrever);

     /* semctl (sema, ExM, GETVAL );*/ /* pega os valores */
     printf (" ExM %d ||", /*arg.val*/semctl (sema, ExM, GETVAL ));
     /* semctl (sema, PodeEscrever, GETVAL );*/ /* pega os valores */
     printf (" PodeEscrever %d ||", semctl (sema, PodeEscrever, GETVAL ));
     /* semctl (sema, PodeLer, GETVAL  ); *//* pega os valores */
     printf (" PodeLer %d \n", semctl (sema, PodeLer, GETVAL ));
}

void log (FILE *log, int sema, T_SHM *shmem)
{
   struct timeval  temp;   /* hora instantanea local */
   unsigned long   tnow;

   gettimeofday(&temp, (struct timezone *)0);
   tnow = (unsigned)(temp.tv_sec*1000000)+(unsigned)temp.tv_usec;

   fprintf (log, " Escrevendo %s || Leitores %d || SuspensosEmPodeLer %d ||
      SuspensosEmPodeEscrever %d || ExM %d || PodeEscrever %d || PodeLer %d  => %s",

   shmem -> Escrevendo == TRUE ? "TRUE ": "FALSE",
   shmem -> Leitores,
   shmem -> SuspensosEmPodeLer,
   shmem -> SuspensosEmPodeEscrever,

   semctl (sema, ExM, GETVAL ),

   semctl (sema, PodeEscrever, GETVAL ),

   semctl (sema, PodeLer, GETVAL ),
   ctime(&(temp.tv_sec)));
}
```

leitor.c

```
/*

 Autor: Alexandre Sztajnberg
 Data: 11/12/1992

*/

#include "le.h"

void print_status (int sema, T_SHM *shmem);

void log (FILE *log, int sema, T_SHM *shmem);

void main ()

{
   int   j,i,k,t;

   key_t chave_memo;
   key_t chave_sema;

   int memo_comp, sema;

   struct sembuf sops;

   int semval;

   T_SHM *shmem;    /* variavel que aponta para a area de memoria compartilhada */

   char path[30];   /* possivel string com o path do arquivo - nao deve ser necessario */
   char buf[STRING_TAM + 1];    /* buffer onde sera trans. bancaria ao ler */
   int nbyte;       /* numero de bytes a ser lido */
   int rbyte;       /* numero de bytes lidos */
   /*int fd;*/          /* descritor do arquivo comum a ser escrito/lido */
   FILE *fd;

   char b [12][20];

   FILE *semf, *smemf; /* arquivos com as chaves de IPC */

  char modo, pross;       /* inica o modo manual ou automatico de execucao */
  int bmode;

  char fim;           /* inica o modo manual ou automatico de execucao */
  int bfim = FALSE;
  int n;

  FILE *leitlog;

      printf ("LEIT: Alexandre Sztajnberg - Setembro/96\n");
      printf ("LEIT: Programa implementando Leitor\n");
      printf ("LEIT: Semaforos e Memoria Compartilhada\n");
      printf ("=======================================\n");

/*
   printf ("LEIT: chave da memoria compartilha: \n");
   scanf  ("%li", &chave_memo);

   printf ("LEIT: chave do semaforo: \n");
   scanf  ("%li", &chave_sema);

*/

    printf ("ESCR: Execucao manual ou automatica: [m/a] \n");
    scanf  ("%c", &modo);

    if (modo == 'm') bmode = MANUAL;
    else bmode = AUTO;
```

```
   /* le as chaves nos arquivos */
    smemf = fopen (MKEY_FILE, "r");
    fscanf (smemf, "%li", &chave_memo);

    semf = fopen (SKEY_FILE, "r");
    fscanf (semf, "%li", &chave_sema);

    leitlog = fopen (LLOG, "w");
    fprintf (leitlog, "Leitor =====> Inicio da execucao \n");

   /* pede acesso a memoria compartilhada */
   if ((memo_comp = shmget (chave_memo, 2 * sizeof (int), IPC_EXCL)) == ERRO)
      printf ("LEIT: Erro no acesso da memoria compartilhada (key = %d)\n", chave_memo);
   else
      printf("LEIT: ID shared mem devolvido %d\n",memo_comp);

   /* pede acesso aos semaforo */

   if ((sema = semget (chave_sema, NUM_SEM, IPC_EXCL)) == ERRO)
      printf ("LEIT: Erro no acesso aos semaforos (key = %d)\n", chave_sema);
   else
      printf("LEIT: ID semaforo devolvido %d\n",sema);

   /* nao estou tratando se alguma coisa deu errado */

   printf ("LEIT: Inicio do Leitor\n");

   printf("LEIT: ID do semaforo do LEITOR  %d\n",sema);
   printf("LEIT: ID shared mem devolvido LEITOR %d\n",memo_comp);

   shmem = shmat (memo_comp, NULL, 0666);
   printf ("LEIT: Ponteiro para memoria  LEITOR %d\n", shmem);

   for (j = 0;  ((j < 50) && (bfim == FALSE)); j++)
   {
/* =============== Comeca a Ler ========================== */

     printf ("LEIT: !!!! Comeca a Ler !!!!\n");

     if (bmode == MANUAL)
     {
        printf ("LEIT: ATENCAO ! Prossegue [s/n] ? \n");
        scanf ("%c", &pross);
     }
print_status (sema, shmem);
log (leitlog, sema, shmem);

   /* wait (ExM) */

     sops.sem_num = ExM;
     sops.sem_op = WAIT;

     if (( semop (sema, &sops, 1)) == ERRO)
       printf ("LEIT: Erro na operacao wait (ExM)\n");
     else
       printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

      t=rand (); for (i = 0; i < (t *  (j % 10)); i++);

     if ((shmem -> Escrevendo == TRUE) || (shmem -> SuspensosEmPodeEscrever > 0))
     {
        shmem -> SuspensosEmPodeLer++;

print_status (sema, shmem);
log (leitlog, sema, shmem);

        /* signal (ExM) */
```

```
        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (ExM)\n");
        else
          printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

       /* wait (PodeLer) */

        sops.sem_num = PodeLer;
        sops.sem_op = WAIT;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (PodeLer)\n");
        else
          printf ("LEIT: Operacao wait (PodeLer) no semaforo OK\n");

        shmem -> SuspensosEmPodeLer--;

        shmem ->  Leitores++;

print_status (sema, shmem);
log (leitlog, sema, shmem);

        if (shmem -> SuspensosEmPodeLer > 0)
       {

print_status (sema, shmem);
log (leitlog, sema, shmem);

          /* signal (PodeLer) */

          sops.sem_num = PodeLer;
          sops.sem_op = SIGN;

          if (( semop (sema, &sops, 1)) == ERRO)
            printf ("LEIT: Erro na operacao wait (PodeLer)\n");
          else
            printf ("LEIT: Operacao wait (PodeLer) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

       }
        else
       {
           /* signal (ExM) */ /* !!!!!!!!! */

           sops.sem_num = ExM;
           sops.sem_op = SIGN;

           if (( semop (sema, &sops, 1)) == ERRO)
             printf ("LEIT: Erro na operacao wait (ExM)\n");
           else
             printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

        }
     }
     else if (shmem -> SuspensosEmPodeLer > 0)
     {
        shmem ->  Leitores++;

print_status (sema, shmem);
log (leitlog, sema, shmem);
```

```
        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = PodeLer;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (PodeLer)\n");
        else
          printf ("LEIT: Operacao wait (PodeLer) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

     }
     else
     {
        shmem ->  Leitores++;

print_status (sema, shmem);
log (leitlog, sema, shmem);

        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (ExM)\n");
        else
          printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

     }

    /* fim Comeca Ler */

     printf ("LEIT: !!!! Fim: Comeca a Ler !!!!\n");

/* ================= Lendo ====================== */

        printf ("LEIT: !!!! Lendo !!!!\n");

       printf ("LEIT: iniciando o servico\n");

       /* printf ("LEIT: parametros path=%s, nbyte=%d\n", path, nbyte); */

       /* if ((fd = open (SHARED_FILE, O_RDONLY)) < 0)*/
        if ((fd = fopen (SHARED_FILE, "r")) < 0)
       {
              printf ("LEIT_debug: nao consegui fazer o OPEN...\n");

       }
       else

         for (i = 0; i < j; i++)

       /* if ((rbyte = read (fd, buf, STRING_TAM + 1 nbyte)) < 0) */
         if ((rbyte = fscanf (fd, "%s %s %s %s %s %s %s %s %s %s %s %s\n", b[0],
b[1],b[2],b[3],b[4],b[5],b[6],b[7],b[8],b[9],b[10],b[11])) < 0)
        /* if ((rbyte = fread (buf, STRING_TAM + 1, j, fd)) < 0) */
       {
              printf ("LEIT_debug: nao consegui fazero o READ...\n");

              /* close (fd); */
                 fclose (fd);
       }
       else
       {
                /* !!!!!! mostra o arquivo lido */
              printf ("LEIT_out: ");
```

```
                for (n=0; n<12; n++) printf ("%s ", b [n]);
                printf ("\n");
        }

        /* close (fd); */ fclose (fd);

        printf ("LEIT: Fim: Lendo !!!. Status = %d\n", rbyte);

/* fim Ler */

/* ============== Acaba Ler ======================*/

     printf ("LEIT: !!!! Acaba Ler !!!!\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

   /* wait (ExM) */

     sops.sem_num = ExM;
     sops.sem_op = WAIT;

     if (( semop (sema, &sops, 1)) == ERRO)
       printf ("LEIT: Erro na operacao wait (ExM)\n");
     else
       printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

     t=rand (); for (i = 0; i < (t *  (j % 10)); i++);

     shmem ->  Leitores--;

print_status (sema, shmem);
log (leitlog, sema, shmem);

     if ((shmem -> Leitores == 0) && (shmem -> SuspensosEmPodeEscrever > 0))
     {

print_status (sema, shmem);
log (leitlog, sema, shmem);

        /* signal (PodeEscrever) */ /* !!!!!!!!! */

        sops.sem_num = PodeEscrever;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (PodeEscrever)\n");
        else
          printf ("LEIT: Operacao wait (PodeEscrever) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);

     }
     else
     {
        /* signal (ExM) */ /* !!!!!!!!! */

        sops.sem_num = ExM;
        sops.sem_op = SIGN;

        if (( semop (sema, &sops, 1)) == ERRO)
          printf ("LEIT: Erro na operacao wait (ExM)\n");
        else
          printf ("LEIT: Operacao wait (ExM) no semaforo OK\n");

print_status (sema, shmem);
log (leitlog, sema, shmem);
```

```
      } /* fim Acaba Ler */

      printf ("LEIT: !!!! Fim: Acaba Ler !!!!\n");

      if (bmode == MANUAL)
      {
        printf ("LEIT: Atencao: mais uma transacao ?: [s/n] \n");
        scanf  ("%c", &fim);

        if (fim == 'n') bfim = TRUE;
        else bfim = FALSE;
      }
      if (bmode == MANUAL)
      {
        printf ("LEIT: Chavear execucao manual ou automatica: [m/a] \n");
        scanf  ("%c", &modo);
        if (modo == 'm') bmode = MANUAL;
        else bmode = AUTO;
      }
    } /* fim for */

    printf ("LEIT: !!!! Liberando recursos IPC !!!!\n");

     fprintf (leitlog, "Leitor =====> Fim da execucao \n");
     fclose (leitlog);

    shmdt ((void *) shmem);

    exit (0);
}

void print_status (int sema, T_SHM *shmem)
{

      printf (" Escrevendo %s ||", shmem -> Escrevendo == TRUE ? "TRUE ": "FALSE");
      printf (" Leitores %d ||", shmem -> Leitores);
      printf (" SuspensosEmPodeLer %d ||",shmem -> SuspensosEmPodeLer);
      printf (" SuspensosEmPodeEscrever %d ", shmem -> SuspensosEmPodeEscrever);

      /* semctl (sema, ExM, GETVAL );*/ /* pega os valores */
      printf (" ExM %d ||", /*arg.val*/semctl (sema, ExM, GETVAL ));
      /* semctl (sema, PodeEscrever, GETVAL );*/ /* pega os valores */
      printf (" PodeEscrever %d ||", semctl (sema, PodeEscrever, GETVAL ));
      /* semctl (sema, PodeLer, GETVAL  ); *//* pega os valores */
      printf (" PodeLer %d \n", semctl (sema, PodeLer, GETVAL ));

}

void log (FILE *log, int sema, T_SHM *shmem)
{
  struct timeval  temp;    /* hora instantanea local */
  unsigned long   tnow;

       gettimeofday(&temp, (struct timezone *)0);
       tnow = (unsigned)(temp.tv_sec*1000000)+(unsigned)temp.tv_usec;

       fprintf (log, " Escrevendo %s || Leitores %d || SuspensosEmPodeLer %d ||
SuspensosEmPodeEscrever %d || ExM %d || PodeEscrever %d || PodeLer %d  => %s",

       shmem -> Escrevendo == TRUE ? "TRUE ": "FALSE",
       shmem -> Leitores,
       shmem -> SuspensosEmPodeLer,
       shmem -> SuspensosEmPodeEscrever,

       semctl (sema, ExM, GETVAL ),
       semctl (sema, PodeEscrever, GETVAL ),
       semctl (sema, PodeLer, GETVAL ),
       ctime(&(temp.tv_sec)));
}
```

escritor.log

```
Escritor =====> Inicio da execucao

 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 1>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 2 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 2 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo FALSE | Leits 2 | SusPLer 0 | SusPEsc 1 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:06
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo TRUE  | Leits 0 | SusPLer 2 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo FALSE | Leits 0 | SusPLer 2 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo FALSE | Leits 1 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 1>  Sep 11 11:41:07
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:07
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 1 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 0 | SusPLer 1 | SusPEsc 1 | ExM 0 | PdEsc 1 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 1 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo TRUE  | Leits 0 | SusPLer 2 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:08
 Escvdo FALSE | Leits 0 | SusPLer 2 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
 Escvdo FALSE | Leits 1 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
 Escvdo TRUE  | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 0 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
 Escvdo FALSE | Leits 0 | SusPLer 0 | SusPEsc 0 | ExM 1 | PdEsc 0 | PdLer 0>  Sep 11 11:41:09
......

Escritor =====> Fim da execucao
```