SISTEMAS OPERACIONAIS

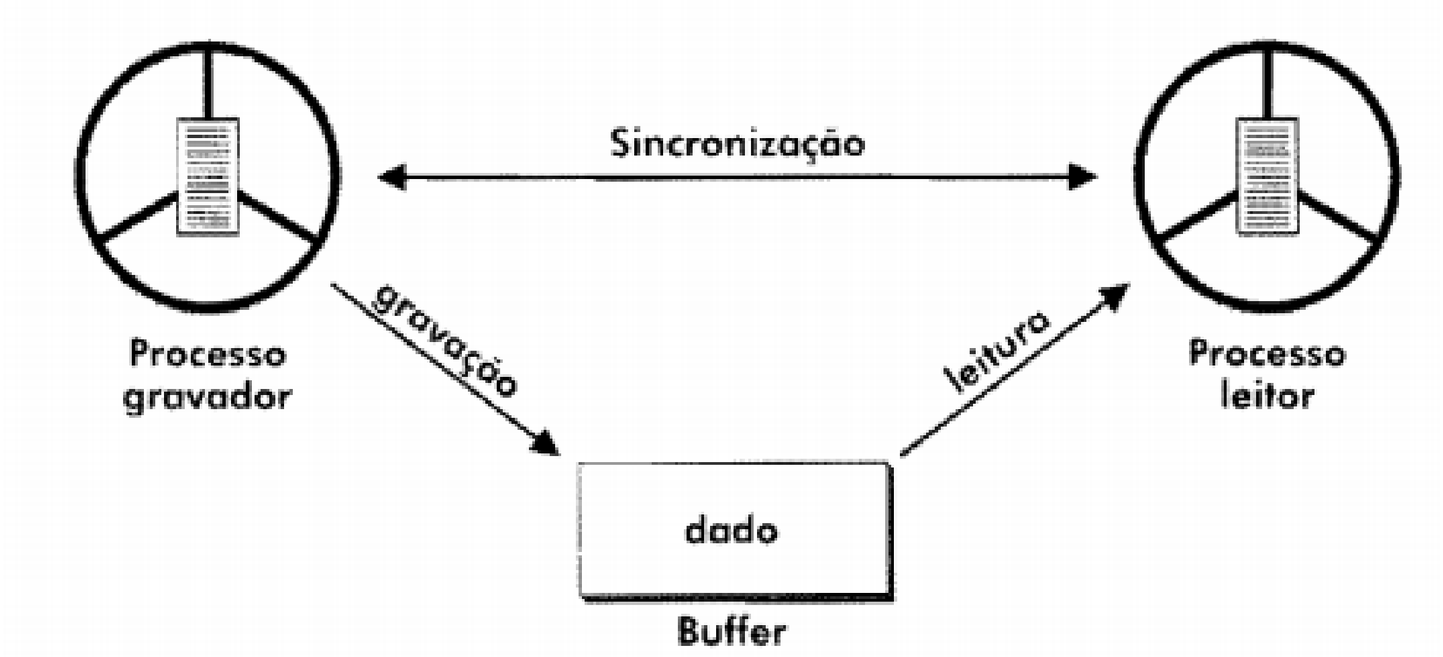
SINCRONIZAÇÃO E COMUNICAÇÃO DE PROCESSOS

Capítulo 7

Professor Fábio Angelo

Conceitos Iniciais

MECANISMOS DE SINCRONIZAÇÃO



Premissas:

1) A comunicação não é direta, usando um buffer para intermediar;

2) O processo gravador escreve apenas se o buffer caso não estiver cheio;

3) O processo leitor consulta o buffer para saber se tem informações a consumir.

Concorrência em programas

```
PARBEGIN
    Comando_1;
    Comando_2;
      .
      .
    Comando_n;
PAREND
```

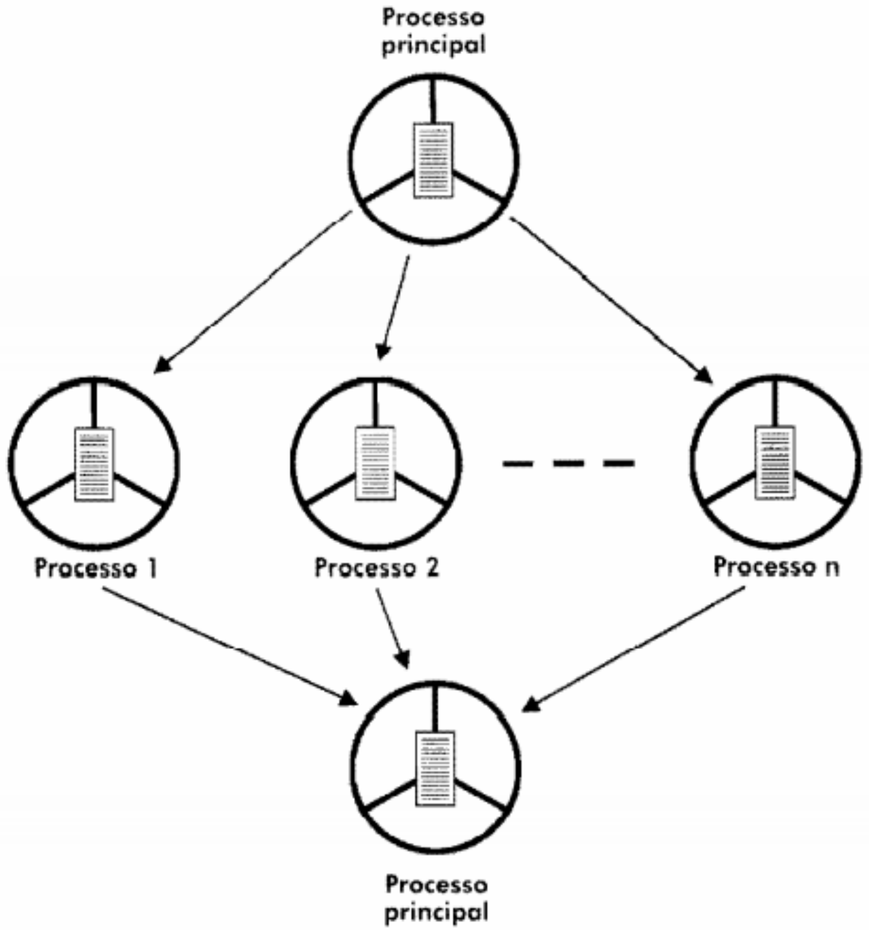


Fig. 7.2 Concorrência em programas.

```
PROGRAM Expressao;
    VAR X, Temp1, Temp2, Temp3 : REAL;
BEGIN
    PARBEGIN
        Temp1 := SQRT (1024);
        Temp2 := 35.4 * 0.23;
        Temp3 := 302 / 7;
    PAREND;
    X := Temp1 + Temp2 - Temp3;
    WRITELN ('x = ', X);
END.
```

X: = SQRT(1024) + (35.4 * 0.23) - (302 / 7)

Problemas no Compartilhamento de Recursos

```
PROGRAM Conta_Corrente;
    .
    .
    READ (Arq_Contas, Reg_Cliente);
    READLN (Valor_Dep_Ret);
    Reg_Cliente.Saldo := Reg_Cliente.Saldo + Valor_Dep_Ret;
    WRITE (Arq_Contas, Reg_Cliente);
    .
    .
END.
```

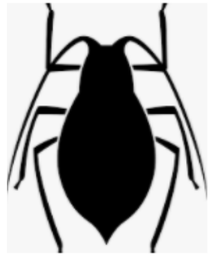
Gaps:

1) Dados em memória e disco
2) Sequencia de execução sem a atualização em disco
3) Tratamento de transações

Caixa	Comando	Saldo arquivo	Valor dep/ret	Saldo memória
1	READ	1.000	*	1.000
1	READLN	1.000	-200	1.000
1	:=	1.000	-200	800
2	READ	1.000	*	1.000
2	READLN	1.000	+300	1.000
2	:=	1.000	+300	1.300
1	WRITE	800	-200	800
2	WRITE	1.300	+300	1.300

Problemas no Compartilhamento de Recursos

Processo A	Processo B
X := X + 1 ;	X := X - 1;



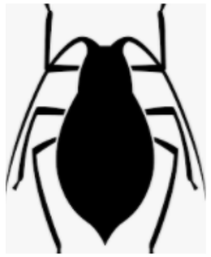
Gaps:

1) Variáveis sendo referenciadas interferem no calculo
2) Sequencia de execução precisa estar estabelecida e controlada
3) Tratamento de transações

Processo A	Processo B
LOAD x,R_a	LOAD x,R_b
ADD 1,R_a	SUB 1,R_b
STORE R_a,x	STORE R_b,x

Problemas no Compartilhamento de Recursos

Processo A	Processo B
LOAD x,R_a	LOAD x,R_b
ADD 1,R_a	SUB 1,R_b
STORE R_a,x	STORE R_b,x



Considerando que X inicia com 2

```
X := X + 1 ;        X := X - 1;
```

Processo	Comando	X	R_a	R_b
A	LOAD X,R_a	2	2	*
A	ADD 1,R_a	2	3	*
B	LOAD X,R_b	2	*	2
B	SUB 1,R_b	2	*	1
A	STORE R_a,X	3	3	*
B	STORE R_b,X	1	*	1

Race Conditions

TRATANDO ESTAS SITUAÇÕES...

```
BEGIN
    .
    .
    Entra_Regiao_Critica ;  (* Protocolo de Entrada *)
    Regiao_Critica;
    Sai_Regiao_Critica;   (* Protocolo de Saída *)
    .
    .
END.
```

Exclusão Mútua:

1) Controla a entrada dos processos em áreas críticas;

2) Atenção ao código que ficará na região crítica;

3) Pode gerar situações indesejadas como o "starvation" ou espera indefinida;

4) O uso de filas para recursos em áreas criticas pode ser util para ordenar o "starvation".

Exclusão Mútua

FORMAS DE TRATAMENTO POR HARDWARE

DESABILITAR INTERRRUPÇÕES

Solução simples que consiste em desabilitar as interrupções antes de entrar em uma região crítica, reabilitando ao sair. Tal artifício impede a mudança de contexto, deixando o acesso exclusivo, por outro lado, compromete a concorrência de recursos.

INSTRUÇÃO TEST-AND-SET

Intrução de máquina especial que permite ler uma variável, armazenar seu conteúdo em uma outra área e atribuir um novo valor a mesma variável. Trata-se de uma instrução indivisível, garantindo que dois processos não manipulem a mesma variável.

Exclusão Mútua

FORMAS DE TRATAMENTO POR SOFTWARE

```
PROGRAM Algoritmo_1;
    VAR Vez : CHAR;
PROCEDURE Processo_A;
BEGIN
    REPEAT
        WHILE (Vez = 'B') DO (* Nao faz nada *);
        Regiao_Critica_A;
        Vez := 'B';
        Processamento_A;
    UNTIL False;
END;
PROCEDURE Processo_B;
BEGIN
    REPEAT
      WHILE (Vez = 'A') DO (* Nao faz nada *);
      Regiao_Critica_B;
      Vez := 'A';
      Processamento_B;
    UNTIL False;
END;
```

```
BEGIN
    Vez := 'A';
    PARBEGIN
      Processo_A;
      Processo_B;
    PAREND;
END.
```

PRIMEIRO ALGORITMO

1) Trata a exclusão entre dois processos;
2) Passa a vez ao próximo processo após a execução;
3) Pode gerar desequilibrio no uso dos recursos.

Exclusão Mútua

FORMAS DE TRATAMENTO POR SOFTWARE

```
PROGRAM Algoritmo_Peterson;
    VAR CA,CB: BOOLEAN;
        Vez : CHAR;
PROCEDURE Processo_A;
BEGIN
    REPEAT
        CA := true;
        Vez := 'B';
        WHILE (CB and Vez = 'B') DO (* Nao faz nada *);
        Regiao_Critica_A;
        CA := false;
        Processamento_A;
    UNTIL False;
END;

PROCEDURE Processo_B;
BEGIN
    REPEAT
        CB := true;
        Vez := 'A';
        WHILE (CA and Vez = 'A') DO (* Nao faz nada *);
        Regiao_Critica_B;
        CB := false;
         Processamento_B;
    UNTIL False;
END;
```

```
BEGIN
    CA := false;
    CB := false;
    PARBEGIN
        Processo_A;
        Processo_B;
    PAREND;
END.
```

ALGORITMO DE PETERSON

1) Trata a exclusão entre dois processos;
2) Elimina a situação de bloqueio indefinido;
3) Persiste o problema da espera ocupada (busy wait).

Exclusão Mútua

SINCRONIZAÇÃO CONDICIONAL

```
PROGRAM Produtor_Consumidor_1;
    CONST TamBuf   = (* Tamanho qualquer *);
    TYPE Tipo_Dado = (* Tipo qualquer *);
    VAR Buffer : ARRAY [1..TamBuf] OF Tipo_Dado;
        Dado_1 : Tipo_Dado;
        Dado_2 : Tipo_Dado;
        Cont   : 0..TamBuf;

PROCEDURE Produtor;
BEGIN
    REPEAT
        Produz_Dado (Dado_1);
        WHILE (Cont = TamBuf) DO (* Nao faz nada *);
        Grava_Buffer (Dado_1, Buffer);
        Cont := Cont + 1;
    UNTIL False;
END;
```

```
PROCEDURE Consumidor;
BEGIN
    REPEAT
        WHILE (Cont = 0) DO (* Nao faz nada *);
        Le_Buffer (Dado_2, Buffer);
        Consome_Dado (Dado_2);
        Cont := Cont - 1;
    UNTIL False;
END;

BEGIN
    Cont := 0;
    PARBEGIN
        Produtor;
        Consumidor;
    PAREND;
END.
```

O que ficou na mente?

1) O que é exclusão mútua e como é implementada?

2) O que é "starvation"?

3) Qual o problema gerado se usarmos o recurso de desabilitar as interrupções para implementar a exclusão mútua?

4) O que é espera ocupada e qual seu impacto no sistema computacional?

5) Explique o que é sincronização condicional. Cite algum exemplo prático.