① 11ª Aula

Exclusão mútua

- Problema: recursos não podem ser usados simultaneamente por vários processos
- Solução: garantir exclusividade de acesso
- Exclusividade = Exclusão mútua

②

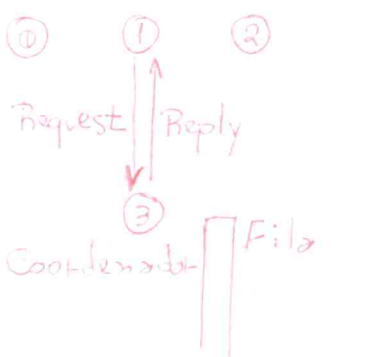
- Regiões (Seções) críticas
  - * Concorrência
  - * Consistência
- Semáforos
  - * Sistemas com monoprocessamento
- Sistemas distribuídos?

  Resposta: "Semáforos distribuídos"
  - * Centralizado
  - * Descentralizado (Não veremos!) → Pouco eficiente!
  - * Distribuído
  - * Anel

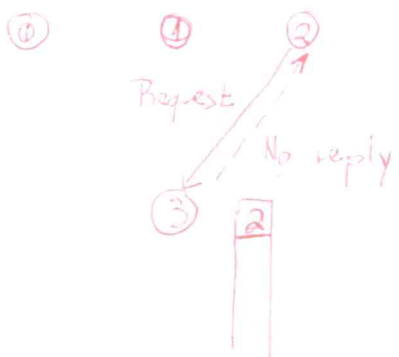
(3) Algoritmo centralizado

- "Similar" a um sistema monoprocessado
- Um dos processos é eleito coordenador
- O coordenador garante a exclusão mútua
- Três tipos de mensagem
  - * Request (requisição)
  - * Reply (conceder)
  - * Release (liberação)

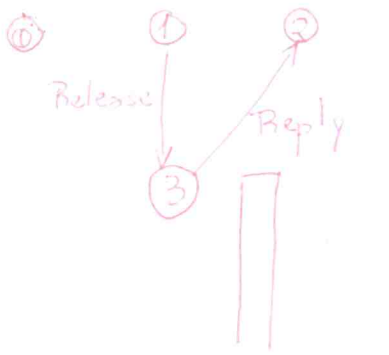
(4)



1 solicita permissão para o coordenador e entra na RC



2 solicita permissão para entrar na RC, mas o coordenador "nega"



1 libera a RC e o coordenador concede permissão para 2

⑤ Algoritmo distribuído

- Não existe coordenador e decisões são tomadas em grupo
- Proposta de Ricarte Agrawala é uma melhoria da ideia de Lamport
- Tal solução exige ordenação global dos eventos do sistema (timestamp)

⑥ Procedimento

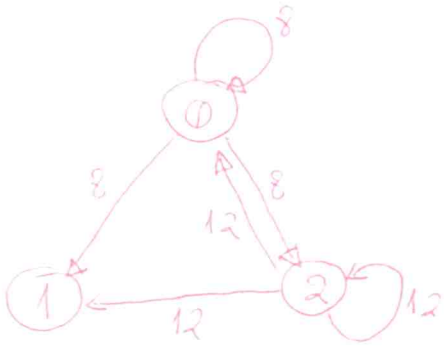
- Se um processo i deseja entrar em uma Região Crítica x (RC_x) ele gera um TimeStamp (TS) e envia Request(i, RC_x, TS).
- O processo i saberá que a RC x está livre se receber Reply de TODOS os outros processos

(7)

→ Quando um processo recebe Request

* se não estiver acessando o RCx e não quiser acessá-la devolve OK (Reply)
* se ele estiver acessando o RCx não responde e coloca o Request em uma fila de espera
* se ele também desejo acessar o RCx, mantém uma fila de espera e envia o Reply (OK) para o Request com menor TS

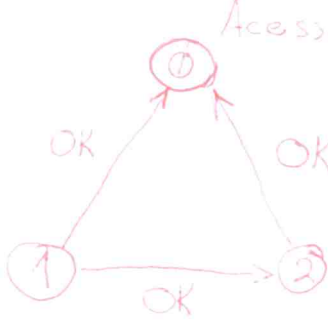
(8) Exemplo



Processos 0 e 2 querem acessar um recurso ao mesmo "tempo"

8 e 12 são os timestamp dos Request

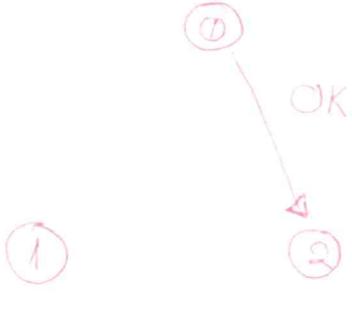
9



Processo 1 não tem interesse no recurso e envia OK para ① e 2

① e 2 percebem o conflito, mas vence quem tem o menor timestamp

10

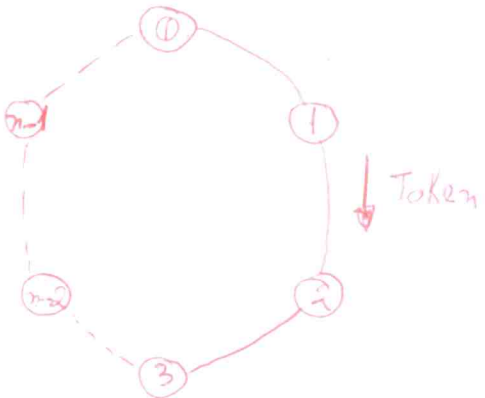


Quando o processo ① libera o recurso ele envia OK para o processo 2 que estava na fila de espera.

(11) Algoritmo em anel

- Conhecido como Token Ring
- Token = "ficha"
- Processos pertencem a um anel lógico
- Quem possui o token é o que tem direito de acessar a região crítica

(12)



Quando o anel é inicializado o processo 0 recebe o Token.

(13) Comparação dos algoritmos

Algoritmo	Problema
Centralizado	Queda do coordenador
Distribuído	Queda de qualquer processo
Anel	Perda do Token ou queda de um processo

(14) Programação com MPI

Rotinas de comunicação coletiva:

- Uma comunicação coletiva envolve todos os processos no âmbito de um comunicador (grupo)
- Todos os processos são, por padrão, membros do comunicador MPI_COMM_WORLD

(15)

Tipos de operações coletivas:

- Sincronização - espera membros de um grupo chegar em um determinado ponto, por exemplo
- Movimento de dados - broadcast, por exemplo.
- Computação coletiva (reduções) - um membro do grupo executa a coleta dos dados dos outros membros e exerce uma operação (min, max, etc) sobre esses dados.

(16) MPI_Barrier()

- Cria uma barreira de sincronização em um grupo
- Cada processo, quando encontra uma chamada MPI_Barrier, fica bloqueado até que todos os processos no grupo cheguem no mesmo ponto.
- Exemplo: MPI_Barrier(MPI_COMM_WORLD);

(17) MPI_Bcast()

→ Comunicação coletiva em que um único processo envia dados para todos os processos do comunicador.

→ MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

→ Exemplo: Pesquisar!

(18) MPI_Reduce()

→ Aplica uma operação de redução em todos os processos do grupo e armazena/coloca o resultado em um processo.

→ MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

→ Exemplo: Pesquisar!

(19) Exemplo

```
# include <math.h>
# include <mpi.h>
# include <stdio.h>

int main (int argc, char * argv[]){
int n, myid, numprocs, i, rc;
double mypi, pi, h, x, sum = 0.0;
MPI_Init (&argc, &argv);
MPI_Comm_size (MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

(20)

```
// Cálculo do Pi
if (myid == 0){
// printf ("Entre com o no de intervalos: ");
// scanf ("%d", &n);
   n= atoi(argv[1]);
}
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
if( n != 0){
   h= 1.0/(double) n;
```

(21)

```
for( i = myid +1; i <= n; i += numprocs){
    x= h * ((double) i - 0,5);
    sum += (4.0/( 1.0 + x * x));
}

// Fim do calculo do Pi

mypi = h * sum;

MPI_Reduce (&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0, MPI_COMM_WORLD);
```

(22)

```
if (myid == 0)
    printf("Valor aproximado de Pi: %.16f \n", pi);
}
MPI_Finalize();
}
```