Ticket Algorithm

//—

Para

EP até amanhã

—//—

ideia: senhar por ordem de chegada

variáveis compartilhadas:

number e next - começam com 1

vetor turn[n] começa com zero

=

0 se não quiser entrar na região crítica

ou ticket caso contrário.

Para entrar na secão crítica o procno CS i faz primeiro { turn[i] = number;
number++;

O procno CS i então espera até que next seja igual ao seu número para en-

trar na seção crítica. Ao sair incre-
menta next.

```
int number = 1, next = 1, turn[1 & n] = ([n]0)
procesr <S [i:= 1 to n] {
    while (true) {
*(      <turn[i] = number; number++;>
        <await (turn[i] == next);>
            seção crítica;
        <next++;>
        seção não crítica.
    }
}
```

* se unir, todos esperam antes e perdem a justiça. Quem chega depois não pega a senha na hora.

intrução Fetch-and-Add

```
FA (var, iner):
< int tmp = var; var = var + iner;
  return (tmp); >

< aux = number++; >
  turn[i] = aux;
```

estouro de inteiro : dificuldade em usar módulo

versão final:

```
procesS[i = 1 to n] {
    while (true) {
        turn[i] = FA (number, 1);
        while (turn != next) skip;
        seção crítica
        next = next + 1;
        seção não crítica
    }
}
```

sem fetch-and-add

Protocolo de Entrada

turn[i] = number; Tie-break

number++;

Protocolo de saída

" região crítica pequena não altera muito a ordem, pois a fila é curta."

" Quando ocorre falta qualquer um pode ser o seguinte"

The Bakery Algorithm

Objetivo: algoritmo sem instruções especiais

Ideia: os clientes verificam entre si quem é o próximo

```
int turn[1:n] = ([n]0);
proces CS[i = 1 to n]{
    while (true) {
        < turn[i] = max (turn[1:n])+1; >
        for[j = 1 to n tal que j ≠ i]
            <await (turn[j] == 0 or turn[i] < turn[j]);>

            região crítica

            turn[i] = 0

            região não crítica
```

problemas:

< turn[i] = max (turn[1:n]) + 1; >

não existe instrução atômica para isto

< await (turn[j] == 0 or turn[i] < turn[j]); >

duas referências a turn[j]

Abordagem: resolver o problema para dois processos.

Protocolo de entrada CS1

turn 1 = turn 2 + 1;

while (turn 2 != 0 and turn 1 > turn 2)

skip

Protocolo de entrada CS2

turn 2 = turn 1 + 1;

while (turn 1 != 0 and turn 2 >= turn 1)

skip;

não vale exclusão mútua

Para resolver os problemas ↖ pé na porta:

logo depois dos protocolos de entrada, colocar

turn = 1

Para generalizar, a condição teria que ser simétrica...

Artifício:

$(a,b) > (c,d)$ == true se $a > c$ ou se $a == c$ e $b > d$

== false caso contrário

generalização

```
int turn[1:n] - ([n] 0);
proc <S[i = 1 to n]{
    while (true) {
        turn[i] = 1;
        turn[i] = max(turn[1:n]) + 1;
        for[j = 1 to n tal que j ≠ i]
        : while (turn[j] != 0 and
        :           (turn[i], i) > (turn[j], j))
        :       skip;
        : ;
        : região crítica
```

Comp. ~~Matriz~~[.]=0; 1/4/09 108

Página 13 das notas de aula. seção não crítica;

} // while

} // process