amor começar com dois processos

1ª tentativa

```
int turn = 1
procss CS1 {
    while (true) {
        <await (turn == 1);>
        seção crítica
        turn = 2;
        seção não crítica
    }
}
procss CS2 {
    :
    <await (turn == 2);>
    seção crítica
    turn = 1;
    :
```

não ocorre ausência de atraso ~
desnecessário (safety), pois um proces-
so pode ser mais longo que outro, au-
mentando a espera do primeiro que não
necessariamente deve rodar em alter-
nância com o outro

Obs.: se tirar a atomicidade do await,
não há exclusão mútua

~ 3ª tentativa

~~boolean in1 = false, in2 = false;~~
~~proc CS1{~~

generalizando:

```
boolean in = false;
process CS i {
    while (true) {
        <await (!in) in = true;>
        seção crítica
        in = false
        seção não crítica
    }
}
```

Obs.: entrada garantida - sim com justiça forte.

"banheiro"

Usar await

1) primitivas de hardware
2) Algoritmos mais sofisticados

instruções do tipo test and-set

fáceis de implementar de forma atômica

```
boolean TS(boolean lock) {
    < boolean inicial = lock;
      lock = true;
      return inicial; >
}
```

```
< await (!in) in = true; >
```

↓

```
while (TS(in))
    skip;
```

← busy waiting

entrada garantida: skip aleatório

otimização para ambientes multi-
processador

```
while (in) skip;
while (TS(in)) {
    while (in) skip;
}
```

para evitar invalidação de cache

Como implementar $\langle S; \rangle$

protocolo de entrada
S;
protocolo de saída

como impl.

<await(B)S;>?

while (!B) - ne vale no máximo 1 vez

proto entrada
while (!B) { protocolo de saída ; skip;
protocolo de entrada } S;
proto saída

outras primitivas de hardware

fetch-and-add

exchange (a, b)

compare-and-swap

```
fetch-and-add (int i)
<
    int val = i;
    i++;
    return val;
>
```

protocolo de entrada while (in != 0);

(FA(in) != 0)

while (in != 0)

skip;

protocolo de saída in = 0;