

# Sistemas Operativos

LEI - 2019/2020

**:: Sincronização de processos ::**

**Escola Superior de Tecnologia de Setúbal - IPS**

# Conteúdos

- Problema da secção crítica
- Soluções práticas para o problema

# Motivação

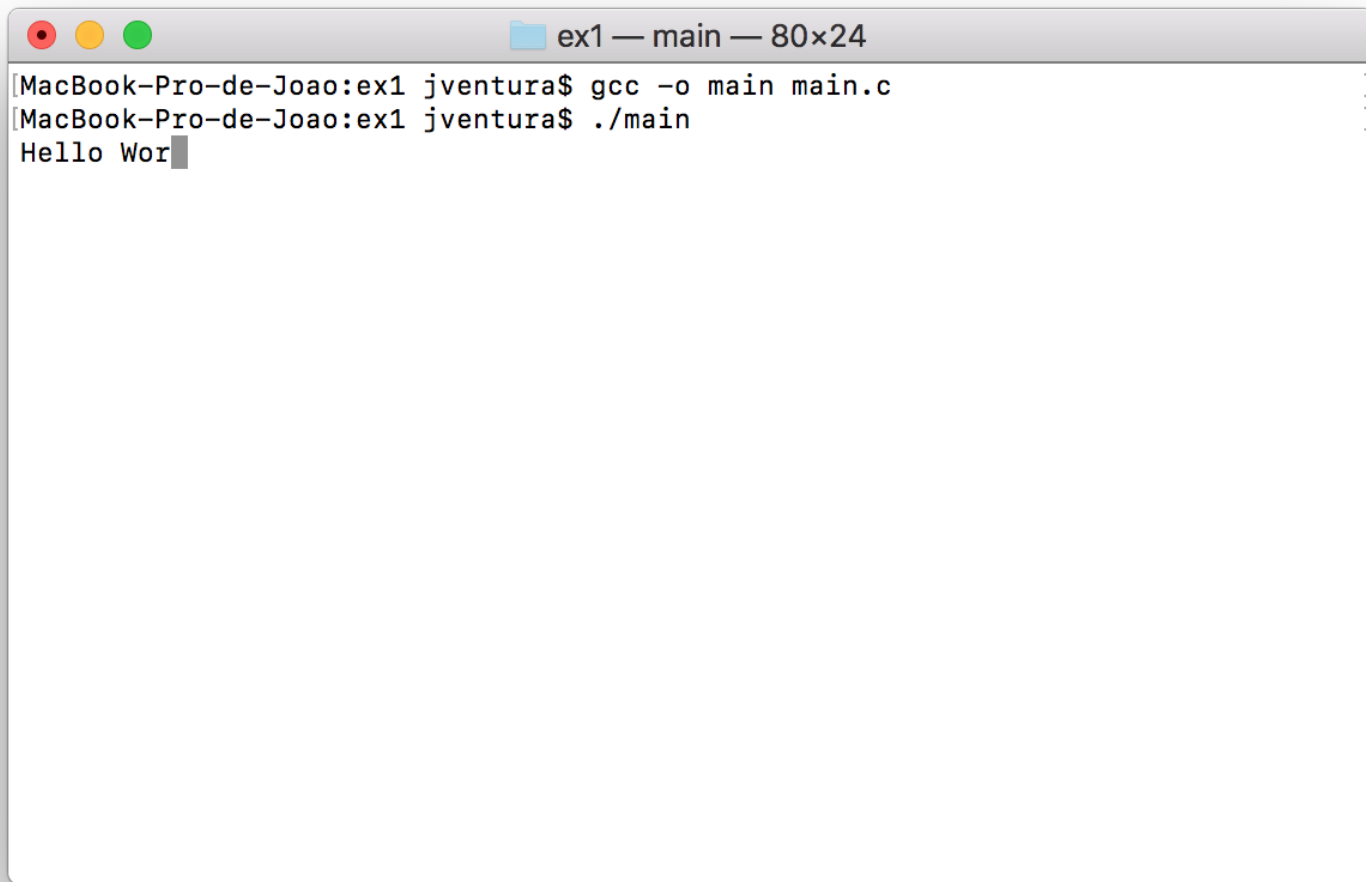
- Acesso concorrential a recursos partilhados pode dar origem a falhas, ex: inconsistência de dados
- Esses recursos devem estar protegidos!

# Exemplo

```
#include <stdio.h>
#include <unistd.h>

// Writes a string char by char
void print(char *str, int msec)
{
    for (int i=0; str[i] != '\0'; i++) {
        printf("%c", str[i]);
        fflush(stdout);
        usleep(msec*1000);
    }
}

int main()
{
    print("Hello World!\n", 100);
    return 0;
}
```

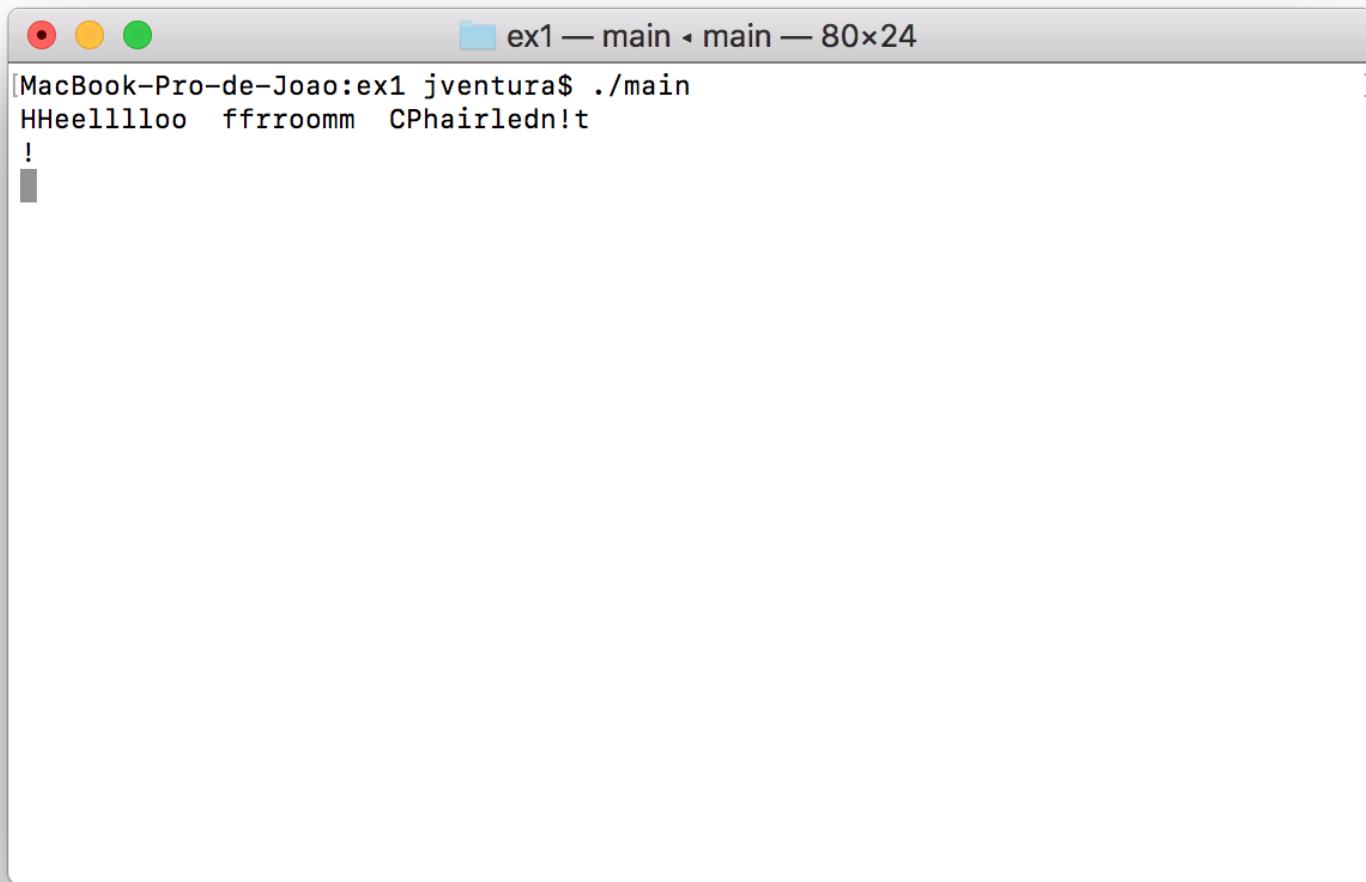


A terminal window titled "ex1 — main — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the following commands and output:

```
[MacBook-Pro-de-Joao:ex1 jventura$ gcc -o main main.c ]  
[MacBook-Pro-de-Joao:ex1 jventura$ ./main ]  
Hello Wor█
```

# Múltiplos processos?

```
int main()
{
    if (fork() == 0) {
        print("Hello from Child!\n", 100);
    } else {
        print("Hello from Parent!\n", 100);
    }
    return 0;
}
```



A macOS terminal window titled "ex1 — main ◀ main — 80x24". The window contains the following text:

```
[MacBook-Pro-de-Joao:ex1 jventura$ ./main  
HHeellllloo  ffrrooomm  CPhairledn!t  
!  
█
```

# Secção crítica

- O "print" do exemplo é uma secção crítica
- Apenas um processo de cada vez deveria poder aceder
- Exclusão mútua (mutual exclusion)



Soluções???



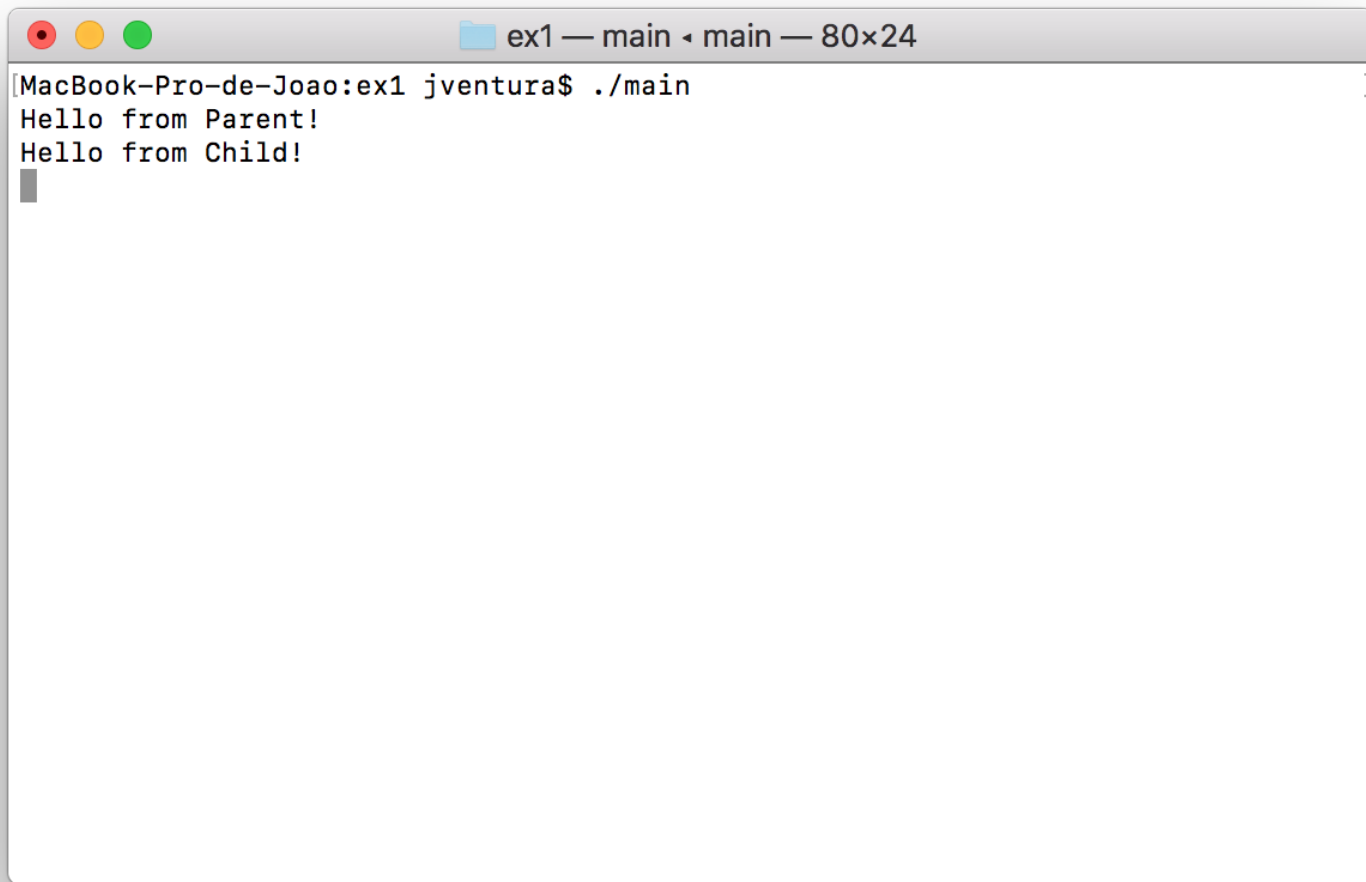
# Mutex locks (MUTual EXclusion)

- Processos devem adquirir a "fechadura" antes de entrar na secção crítica.
- Devem libertar a "fechadura" após sair da secção crítica.

```
acquire()  
    print("Hello from Child!\n", 100);  
release()
```

# Acquire e release

```
void acquire() {  
    while (available == false) {  
        // busy wait  
    }  
    available = false;  
}  
  
void release() {  
    available = true;  
}
```

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, followed by a folder icon and the text "ex1 — main ◀ main — 80x24". The terminal content shows a shell prompt "[MacBook-Pro-de-Joao:ex1 jventura\$ ./main" followed by two lines of output: "Hello from Parent!" and "Hello from Child!". A small grey cursor block is visible on the line following the second output line.

```
[MacBook-Pro-de-Joao:ex1 jventura$ ./main  
Hello from Parent!  
Hello from Child!  
█
```

# Semáforos

- Declara uma variável inteira  $n$
- Acedida através de *wait()* e *post()*

```
wait(S)
    print("Hello from Child!\n", 100);
post(S)
```

# Wait e post

```
void wait(S) {  
    while (S == 0) {  
        // busy wait  
    }  
    S = S - 1;  
}
```

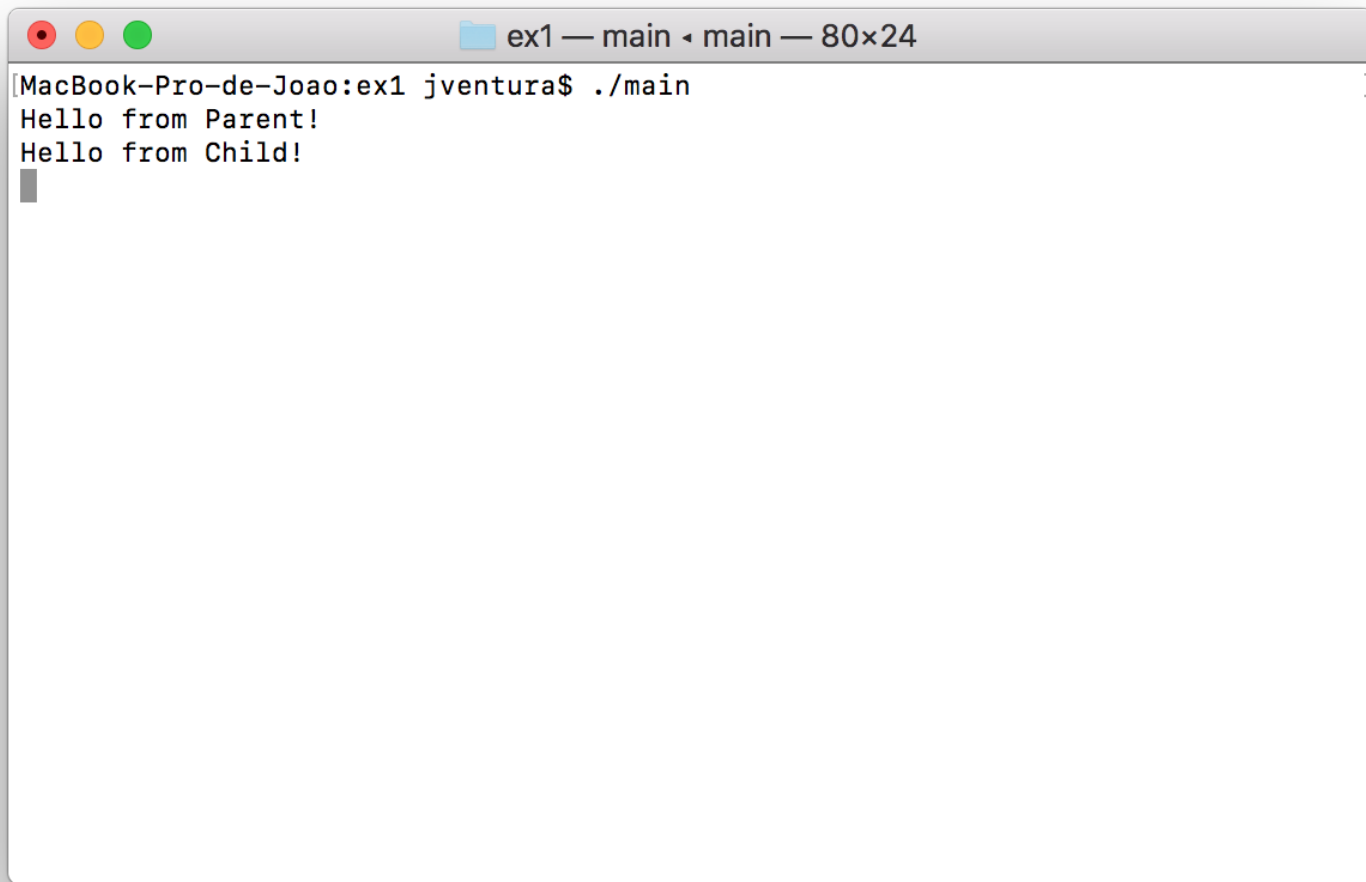
```
void post(S) {  
    S = S + 1;  
}
```

# Exemplo em C/Linux

```
int main()
{
    sem_unlink("mymutex");
    sem_t *mutex = sem_open("mymutex", O_CREAT, 0644, 1);

    if (fork() == 0) {
        sem_wait(mutex);
        print("Hello from Child!\n", 100);
        sem_post(mutex);
    } else {
        sem_wait(mutex);
        print("Hello from Parent!\n", 100);
        sem_post(mutex);
    }

    sem_close(mutex);
    return 0;
}
```

A screenshot of a macOS terminal window. The title bar at the top shows three colored window control buttons (red, yellow, green) on the left, followed by a folder icon and the text "ex1 — main ◀ main — 80x24". The terminal content shows a shell prompt "[MacBook-Pro-de-Joao:ex1 jventura\$ ./main" followed by two lines of output: "Hello from Parent!" and "Hello from Child!". A small grey cursor block is visible on the line following the second output line.

```
[MacBook-Pro-de-Joao:ex1 jventura$ ./main  
Hello from Parent!  
Hello from Child!  
█
```

# Exemplos

```
void critical_section(int pid)
{
    printf("Process %d enters critical section!\n", pid);
    sleep(2);
    printf("Process %d leaves critical section!\n", pid);
}
```



## Qual o output (semáforo = 1)?

```
int main()
{
    sem_unlink("mysem");
    sem_t *sem = sem_open("mysem", O_CREAT, 0644, 1);

    for (int i=0; i<10; i++) {
        if (fork() == 0) {
            sem_wait(sem);
            critical_section(getpid());
            sem_post(sem);
            exit(0);
        }
    }

    sem_close(sem);
    return 0;
}
```

```
ex1 — -bash — 80x24
[MacBook-Pro-de-Joao:ex1 jventura$ ./main
Process 23584 enters critical section!
MacBook-Pro-de-Joao:ex1 jventura$ Process 23584 leaves critical section!
Process 23585 enters critical section!
Process 23585 leaves critical section!
Process 23586 enters critical section!
Process 23586 leaves critical section!
Process 23587 enters critical section!
Process 23587 leaves critical section!
Process 23588 enters critical section!
Process 23588 leaves critical section!
Process 23589 enters critical section!
Process 23589 leaves critical section!
Process 23590 enters critical section!
Process 23590 leaves critical section!
Process 23591 enters critical section!
Process 23591 leaves critical section!
Process 23592 enters critical section!
Process 23592 leaves critical section!
Process 23593 enters critical section!
Process 23593 leaves critical section!
█
```

## Qual o output (semáforo = 2)?

```
int main()
{
    sem_unlink("mysem");
    sem_t *sem = sem_open("mysem", 0_CREAT, 0644, 2);

    for (int i=0; i<10; i++) {
        if (fork() == 0) {
            sem_wait(sem);
            critical_section(getpid());
            sem_post(sem);
            exit(0);
        }
    }

    sem_close(sem);
    return 0;
}
```

```
ex1 — -bash — 80x24
[MacBook-Pro-de-Joao:ex1 jventura$ ./main
Process 23624 enters critical section!
Process 23625 enters critical section!
MacBook-Pro-de-Joao:ex1 jventura$ Process 23624 leaves critical section!
Process 23625 leaves critical section!
Process 23627 enters critical section!
Process 23626 enters critical section!
Process 23626 leaves critical section!
Process 23627 leaves critical section!
Process 23628 enters critical section!
Process 23629 enters critical section!
Process 23629 leaves critical section!
Process 23628 leaves critical section!
Process 23630 enters critical section!
Process 23631 enters critical section!
Process 23630 leaves critical section!
Process 23631 leaves critical section!
Process 23632 enters critical section!
Process 23633 enters critical section!
Process 23633 leaves critical section!
Process 23632 leaves critical section!
```

# Qual o output (semáforo = 4)?

```
ex1 — -bash — 80x24
[MacBook-Pro-de-Joao:ex1 jventura$ ./main
Process 23659 enters critical section!
Process 23660 enters critical section!
Process 23661 enters critical section!
Process 23662 enters critical section!
MacBook-Pro-de-Joao:ex1 jventura$ Process 23659 leaves critical section!
Process 23662 leaves critical section!
Process 23660 leaves critical section!
Process 23661 leaves critical section!
Process 23663 enters critical section!
Process 23664 enters critical section!
Process 23665 enters critical section!
Process 23666 enters critical section!
Process 23664 leaves critical section!
Process 23663 leaves critical section!
Process 23665 leaves critical section!
Process 23666 leaves critical section!
Process 23668 enters critical section!
Process 23667 enters critical section!
Process 23667 leaves critical section!
Process 23668 leaves critical section!
█
```

# Deadlocks

P1:

```
wait(S)
wait(Q)
·
·
post(S)
post(Q)
```

P2:

```
wait(Q)
wait(S)
·
·
post(Q)
post(S)
```

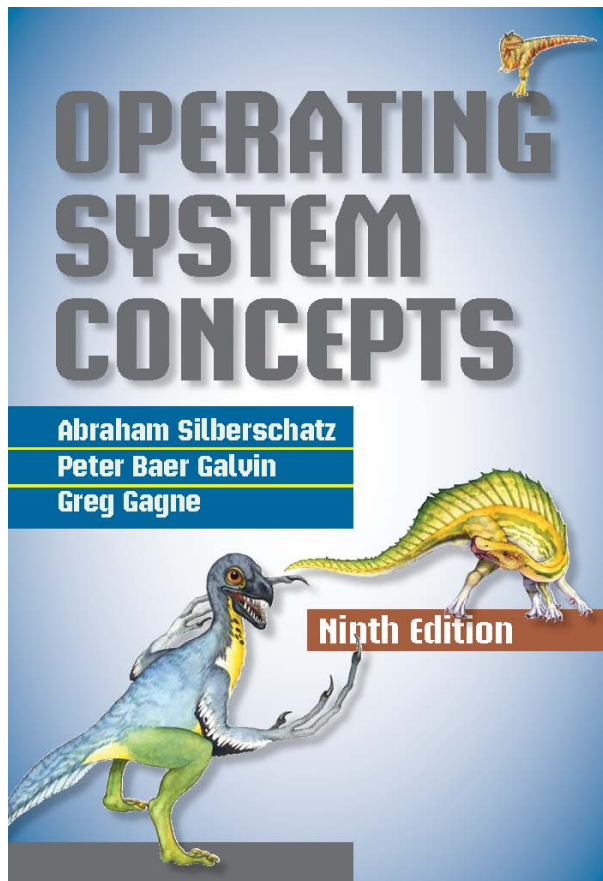
- P1 acquire **S** e P2 acquire **Q**
- P1 fica à espera de **Q** e não liberta **S**
- P2 fica à espera de **S** e não liberta **Q**

**Quiz...**

# Sumário

- Exclusão mútua garante acesso único a secções críticas
- Na prática usa-se essencialmente mutexes e semáforos
- Os Sistemas Operativos fornecem estes mecanismos





Ler capítulo 5...