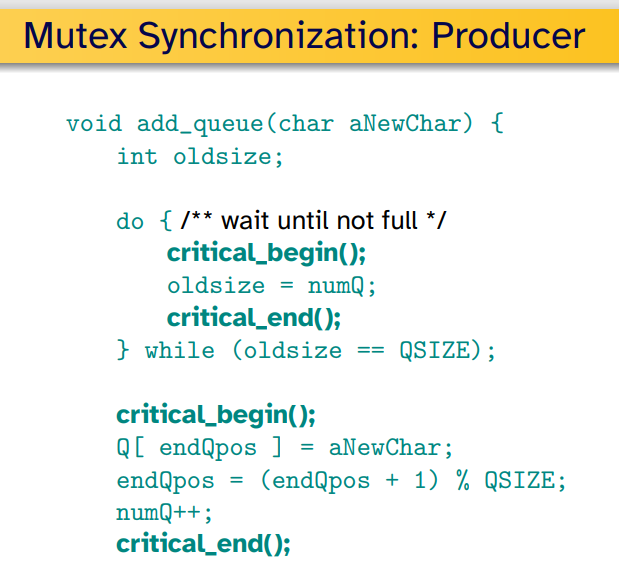
Spltty vs critical\_section

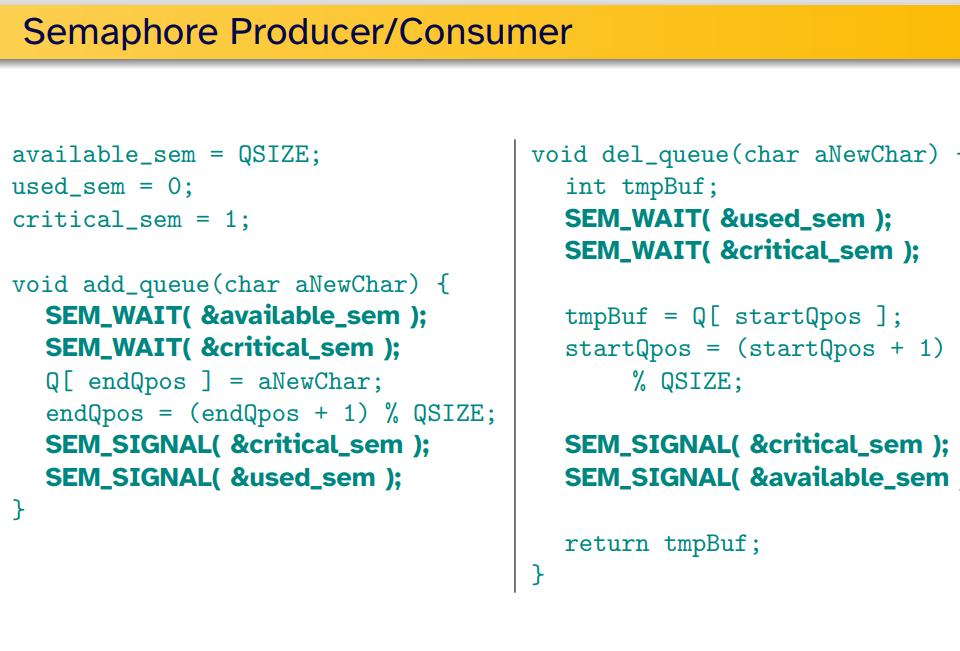
-> manage critical section

Prevent interrupt from happening



Problem of this, it keeps looping so it is a burden to cpu.

* Sleep could be a good alterative



Del\_queue 부터 불렸다면

SEM\_WAIT(&used\_sem); 에서 멈춘다. 왜? Used\_sem 이 0 니까

결국 쉽게 말하면 WAIT 은 하나줄이고 0이아니면 내보내줌

근데 들어갈때부터 0 이면 바로 멈춰버림

Signal 은 하나 더하기 결국에

SEM\_WAIT(&critical\_sem);

SEM\_SIGNAL(&critical\_sem);

이 사이가 결국 임계구간임 왜냐면 ciritical\_sem 은 1이기떄문에 . 하나더해지고 하나빼지고

정리 – available\_sem = 은 가능한 공간을 뜻 즉 0 이면 add 가 안됨(꽉찬거임)

Used\_sem 은 얼마나 queue 에 차있는지, 뜻함. 즉 0개면 del 은 안된다.

Signal 은 increment always succeeds

Wait - > is close to a gate

Deadlock - > when two processes try to use resources but they are all allocated to each other and both of them are waiting until one of them releases their resource which won’t happen cuz they are both waiting for each other.

Java method locking synchronized - > it’s using one singular mutex for the entire program so if you put any loops in it , it could directly affect to the performance of your program.

block locking: public void someMethod() { while(keepGoing()) { synchronized(someObject) { count++; } } } -> This could use more than one mutex so it’s better in terms of the performance compare to method.