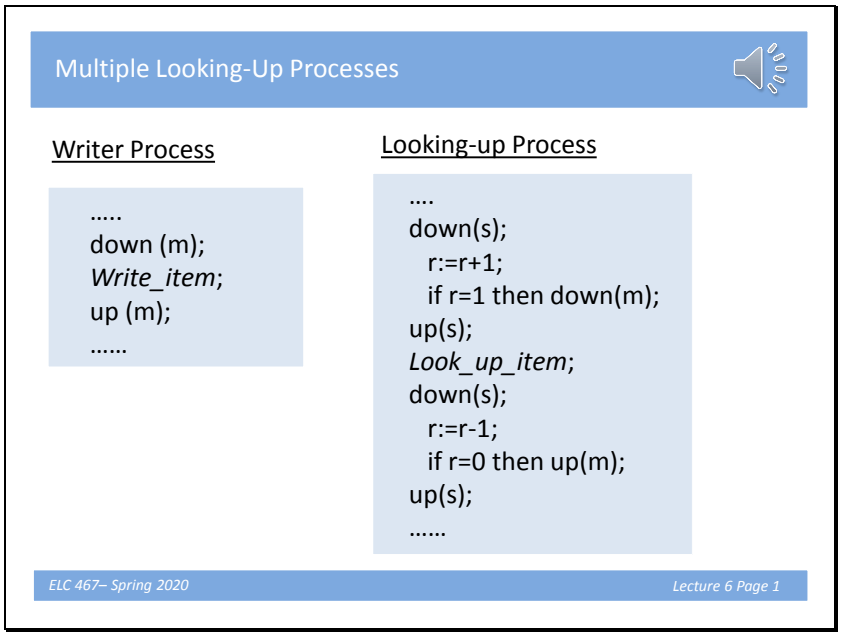


Slide 1

In the racing problem considered before, errors occur when several processes try to access the same data and some of the processes modify the data. If any number of processes read the data only (or look it up as we call it here) they may run together without problems. In the following, we assume that many processes look up data, and one process try to modify it.

Slide 2

The process modifying data is called here the writer process. Since its critical section must access data exclusively, it uses a mutual exclusion semaphore "m" initially equal to 1 as before.

Slide 3

Each looking-up process uses the shown code. Here, “r” is a normal integer variable (not a semaphore!). It is used to count how many processes are currently looking up data, and initially equals 0. Since “r” will be accessed by many processes, it should be protected by semaphore “s” to avoid racing on it. Again, “s” initially equals 1.

Slide 4

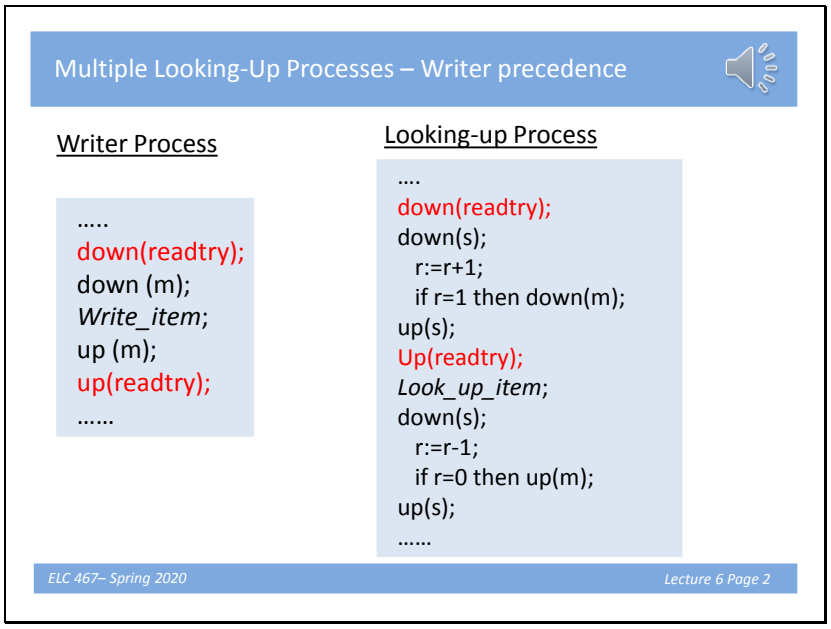
Before looking up data, process increments r, showing that one more process is going to read data. IF r=1, this means that this is the first looking up process. Thus, it must perform down on m as writer may be accessing data now. If r is not equal to 1, this means that there are already processes looking up data. In this case, process access data directly without a need for down operation.

Slide 5

When process finishes looking up data, it decrements r to indicate that number of reading processes will be reduced by 1. If r=0, this means there are no more looking up processes. Thus, process performs up(m) to allow writer to run. If r is not 0, this means that other processes are still looking up data. In this case, process ends without performing an up operation. In short, first reader performs down(m), and last reader performs an up(m).

Slide 6

There is, however, a shortcoming in this solution. If writer wants to access data it will wait not only for currently looking up processes, but also for any number of new starting looking up processes. We say that this solution gives precedence to looking up processes. This may not be the logical solution. For example, if a process is going to reserve an airplane seat, we do not want many processes to read that the seat is empty first.



Slide 7

To give precedence to writer, we modify the solution as shown. Readtry is a semaphore initially equal to 1. When writer tries to write, it decrements this semaphore to 0. This will cause new looking up processes to be blocked. Writer will wait thus for currently reading processes only. When writer finishes, new looking up processes will run.