Computação em Larga Escala

Problem of the producers and the consumers – Algorithmic analysis

António Rui Borges

Summary

- *Problem formulation*
- *Synchronization*

Problem formulation - 1

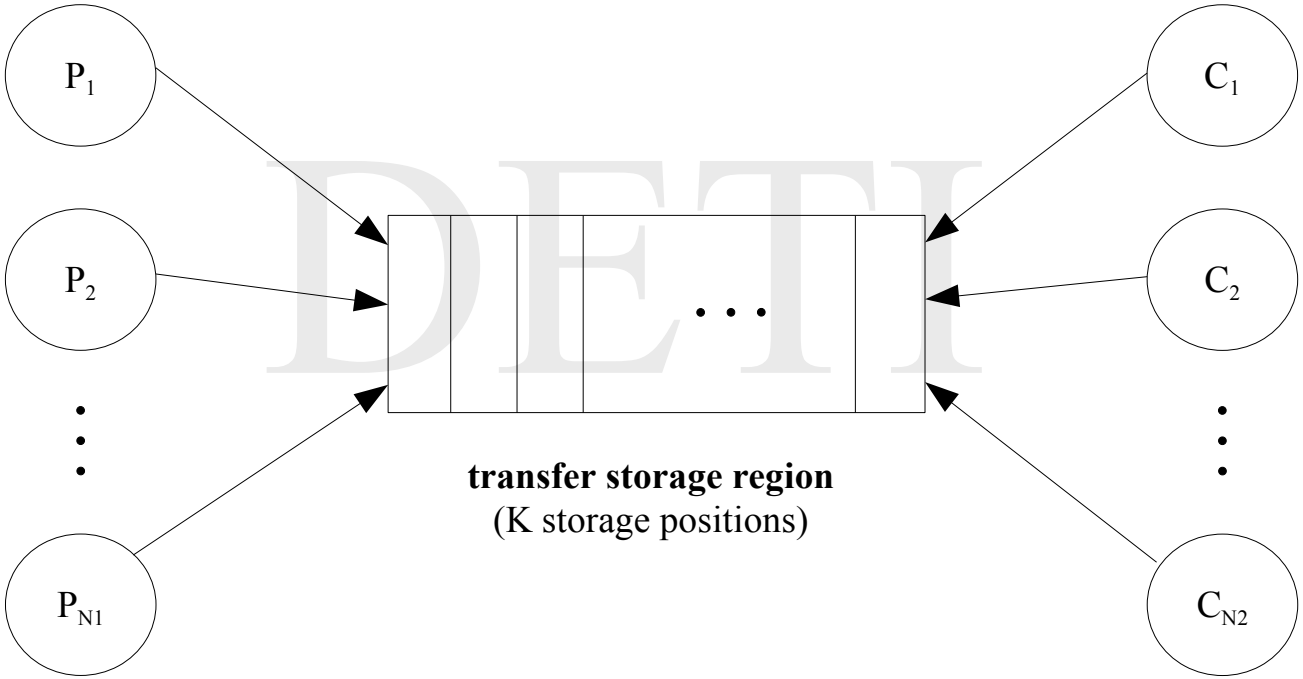
The *problem of the producers and the consumers* is a problem that is traditionally used to evaluate the functionality of new synchronization mechanisms in concurrent programming.

There are two classes of entities

- the *producers*, which execute an infinite loop where they produce data of some sort, stored it in a shared region and do something else
- the *consumers*, which also execute an infinite loop where they retrieve data from the shared region, consume it in some way and do something else.

The interaction among them must proceed in an organized manner, independently of the number of producers and consumers, the time they take to fulfill they loop activities and the storage capacity of the shared region.

Problem formulation - 2



Problem formulation - 3

Producer life cycle

```
forever
{ produceVal (&data);
  putVal (data);
  doSomethingElse ();
}
```

Consumer life cycle

```
forever
{ getVal (&data);
  consumeVal (data);
  doSomethingElse ();
}
```

Synchronization

Since multiple activities, of type `putVal` and `getVal`, are taking place concurrently over the transfer region, one must avoid racing conditions which will lead to inconsistency of information, deadlock or starvation.

Thus, one must ensure that

- *mutual exclusion* – only a single `putVal` or `getVal` operation may be executed at a time
- *producer synchronization point* – when the transfer region is full, the *producers* can not store data and must wait for a location to be empty
- *consumer synchronization point* – when the transfer region is empty, the *consumers* can not retrieve data and must wait for a location to be occupied
- every *producer* must be able to store sooner or later data in the transfer region
- every consumer should be able to retrieve sooner or later data from the transfer region
- every piece of data, stored in the transfer region, should be sooner or later retrieved.

Solution with monitors

A correct solution to the problem can be obtained by transforming the transfer region into a monitor. If the code which implements the solution is written in C, one can use the functionalities provided by the library `pthread` to do this.

In this sense, one has

- *mutual exclusion* – through the calling of the operations of `lock` and `unlock` on a *mutex* variable
- *producer synchronization point* – through the calling of the operations of `wait` (to block) and `signal` (to wake up) on a *condition* variable
- *consumer synchronization point* – through the calling of the operations of `wait` (to block) and `signal` (to wake up) on a *condition* variable.

Compiling and linking the code

```
gcc -Wall -O3 -o producersConsumers producersConsumers.c fifo.c -lpthread -lm
```