Fundamentos de Sistemas de Operação

MIEI 2017/2018

Laboratory session 5

Objectives

Concurrent programming with Pthreads [1] – multi-threaded file copy using the producer-consumer strategy. Practice with locks, conditional variables and semaphores.

The producer-consumer problem

The producer-consumer problem (or the Dijkstra's bounded buffer problem) [2] is typical example of a concurrency problem between two (or more) entities (e.g. processes or threads) that communicate/coordinate with each other via produced data, i.e. one entity depends on/consumes the data the other entity produces.

The producer and consumer share a limited sized buffer for communication (e.g. a queue i.e. with a FIFO strategy) whose access must be coordinated among them. Namely, the producer's role is to be continuously producing some data item, put it on the shared buffer if there is enough space or, otherwise, wait (e.g. go to sleep) until there is enough space for the item (the data item could also be discarded). The consumer's role, in turn, is to wait (e.g. go to sleep) until there is a data item to be consumed and then extract it from the buffer; the consumer repeats this behaviour until there is some indication that the producer has end its execution.

The producer-consumer strategy allows decoupling the communication between the producer and the consumer, i.e. the producer can produce data items at its own rhythm, as long as it has enough space in the buffer; the consumer can also consume data at its own pace and as long as some item is available in the buffer. In the limit situations, the producer must block/go to sleep if there is not enough space for a data item in the buffer, and it must be awaken/unblocked as soon as there is. The producer, in turn, blocks/goes to sleep if there is no data item to be consumed and it also must be awaken/unblocked as soon as a new data item is available in the shared buffer.

Multithreaded file copy

The goal of this work is to implement a file copy (e.g. similarly to lab 01) using the producer-consumer strategy – the *producer thread* reads data from a file and sends it to the shared buffer and the *consumer threads* reads data from the shared buffer and writes it to a new file. In order to coordinate their actions, the producer and consumer use *conditional variables* and *locks*/*mutexes* for accessing the buffer and synchronise their actions.

To implement your solution, you must adapt functions `put` and `get` comprised in the codebase available from the discipline's site on the clip system, and depicted below. You are also requested to reply to the questions in the code.

```
void put( msg_t *b ) // must execute
                     // concurrently with get
{
  OBJ tmp = malloc( sizeof(msg_t));
  memcpy(tmp, b, sizeof(msg_t));

  while ( bufFull() )
    ;

  bufPut(tmp);
}
```

```
msg_t *get( void )  // must execute
                     // concurrently with put
{
   while ( bufEmpty() )
    ;

  msg_t *r = (msg_t*) bufGet();

  return r;
}
```

Moreover, recall the time that the sequential program from lab 01 took to perform the file copy and compare it to your program's execution time.

Semaphores

Adapt your solution with locks and conditional variables now using semaphores.

Bibliography

[1] Sections about Concurrency of the recommended book, “Operating Systems: Three Easy Pieces Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau”

[2] http://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf

[3] http://pages.cs.wisc.edu/~remzi/OSTEP/threads-sema.pdf

[4] Slides T15 (available from CLIP)