Producer - Consumer Queue

Gitopoulos Giorgos, 9344

Real Time Embeded Systems @ ECE AUTh

1. Introduction

In this project, a **Producer - Consumer** queue is implemented, using **C** and **P-Threads**. Each producer adds to the queue a specific number of works and the consumers execute these works following **FIFO (First In First Out)** method. Also, the average waiting time of a work in the queue for variable number of consumers and predefined number of producers is computed, to find the optimized number of consumers in each case. The code of the project can be found **here**.

2. Implementation

2.1. P-Threads & FIFO

We initialize *NUM_OF_PRO* **P-Threads** that work as producers and *NUM_OF_CON* **P-Threads** that work as consumers. Each producer adds to the queue *WORKS_PER_PRO* works and the size of the queue is *QUEUESIZE*. If the queue is full for a moment, the producers block, until a consumer remove a work from the queue. Simultaneously, each consumer removes the firstly added of the remaining works of the queue (according to **FIFO** method) and carries it out. In the same way as previously, if the queue is empty, the consumers block and wait for a producer to add a work in it. The works are simple tasks (in our code printing a message).

2.2. Deadlocks & Race Conditions

In order to avoid **deadlocks**, when both producers and consumers finish their work and consumers remain blocked (queue is empty) waiting for a producer to add a work, the last of the producers sends continuous signals to unblock all the consumers. Moreover, when the queue or other global variables are modified by a thread, **mutexes** are used to avoid **race conditions**.

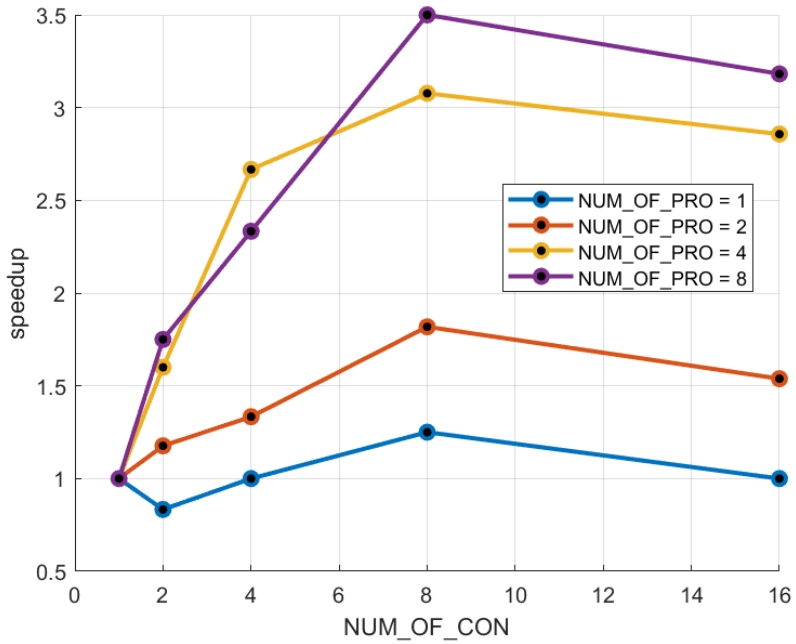
3. Performance

The code was tested locally, using *AMD Ryzen 7 4700U* processor (8 core - 8 threads) and we present the results about the average waiting time per work in the queue (the time between the moment that the work was added to the queue and the moment that a consumer removed it from the queue - execution time was not included). The static values *QUEUESIZE* = 10 and *WORKS_PER_PRO* = 50 were used during our tests. The values of *NUM_OF_PRO* and *NUM_OF_CON* were selected in a logical range according to the number of processor cores.

Table 1: *Average waiting time per work in queue* (μs)

	NUM_OF_CON = 1	*NUM_OF_CON = 2*	*NUM_OF_CON = 4*	*NUM_OF_CON = 8*	*NUM_OF_CON = 16*
NUM_OF_PRO = 1	102	123	98	80	99
NUM_OF_PRO= 2	198	167	151	112	131
NUM_OF_PRO = 4	406	253	148	130	141
NUM_OF_PRO = 8	688	410	289	205	221

Figure 1: *Speedup comparing to single consumer operation*



4. Results

We can notice in *Table 1* and *Figure 1*, that for different number of producer - threads, the waiting time is minimized using **8 threads as consumers**. This result is reasonable, as the maximum number of threads that can run in parallel in our processor is 8. Using more than 8 consumers is expected just to serialize the operation and not to improve the performance. Although some of our threads are originally assigned as producers, each time a producer finishes its work, the thread is released and it can work as a consumer. The parameters *QUEUESIZE* and *WORKS_PER_PRO* are crucial for the average waiting time, but their effect is complicated and beyond the scope of this project. However, we expect that the optimal performance in every case will be achieved using approximately 8 consumer - threads.