ARISTOTLE UNIVERSITY OF THESSALONIKI

PRODUCER-CONSUMER PROBLEM

Real Time Operating Systems Course

TZOUVARAS EVANGELOS
AEM: 9659
Email: tzouevan@ece.auth.gr

Table of Contents

Introduction

The objective of this project is to modify a source file, which implements the producer – consumer problem, to fill in the queue with struct elements. Therefore, multiple producers will insert into the queue pointers of a structure, and multiple consumers will take each pointer and will execute the struct function. The communication between the producers and consumers is done by using pthreads and mutexs.

You could find the whole project (code, matlab scripts, .txt results, report) o the github: https://github.com/tzouv/RTES_Producer_Consumer_Work

Source file implementation

First of all it is necessary to declare the new struct and to import it as a variable to the queue struct, as it shown at the figure 1. Now the queue will contain a buffer which elements are workFunction type.

```
typedef struct {
  void *(*work)(void *);
  void *arg;
}workFunction;

typedef struct {
  workFunction buffer[QUEUESIZE];          // An array with type workFunction

  int buf[QUEUESIZE];                      // Buf: array, size: QUEUESIZE
  long head, tail;                         // head and tail: Indexes for the array
  int full, empty;                         // full, empty: Use as booleans: TRUE, FALSE
  pthread_mutex_t *mut;                    // A mutex variable
  pthread_cond_t *notFull, *notEmpty;      // Two condition variables, notFull, notEmpty
} queue;
```

Figure 1.1: Declare the new struct variable

The work function inside the struct takes as an argument an integer array and calculates for each number its sin value.

```
// work function: Argument a pointer
void *work(void *work_array){
  float *work_return;
  work_return = (float *)malloc(10*sizeof(float));
  int i = 0;
  int * work_int = (int *)(work_array);
  for(i = 0; i < 10; i++){
    work_return[i] = (float)sin(work_int[i]);
  }
  printf("\n");
  return (NULL);
}
```

Figure 1.2: work function

Then multiple producers and consumers are initialized and created into the main function. The producers and to the queue workFunction elements, by using the *queueAdd* function and the consumers take the elements, delete them by using the *queueDel* function and execute the function.

```
//arg in_queue into the queue
in_queue.arg = p;
in_queue.work = work;
queueAdd (fifo, in_queue);
```

```
queueDel (fifo, &out);
out.work(out.arg);
free(out.arg);
pthread_mutex_unlock (fifo->mut);
```

Figure1.3: Producer and consumer implementation

Time measurement and results processing

To measure the wait time of each element before it will be executed by the consumer, the gettimeofday() function. A global variable timeval struct is declared with size 2*QUEUESIZE, to keep the producing time and the consuming time. When an element is deleted from a consumer, the waiting time is calculated and is saved as an element on the global array *times*. Also a counter is used to keep the number of elements that are executed.

All these time results will be saved on a .txt to be ready for processing. Due to while(1) loop on the consumer (as a real time program), the only

way to terminate the program is to kill it (Ctrl+C signal). To save the data before the forced execution a signal handler function is used.

```c
/*A handler to close the file when we kill the program
*/
void handler(int sig){
  FILE *fp = fopen("time_measurement_q60_con10.txt","wb");
  int x = 0;
  for(x = 0; x < count; x++){
    char str[32];
    sprintf(str, "%d", times[x]);              // Convert the time into string to save it on the file
    fputs(str, fp);                            // Place the string(time) into the file
    fputc('\n',fp);
  }
  fclose(fp);                                  // Close the file
}
```

Figure1.4: Signal handler function

Then, the .txt files are imported to matlab to create all the statistic graphs. (You could see the matlab script code inside the .zip file)

Waiting time – queue size

The first experiment had ten producers and ten consumers, and the variable parameter was the *queue size* (from 10 to 100). It is observed a linear dependent between the queue size and the waiting time, a logical result, because the bigger queue means more elements that need to be executed on the same system, so the waiting time will be increased.

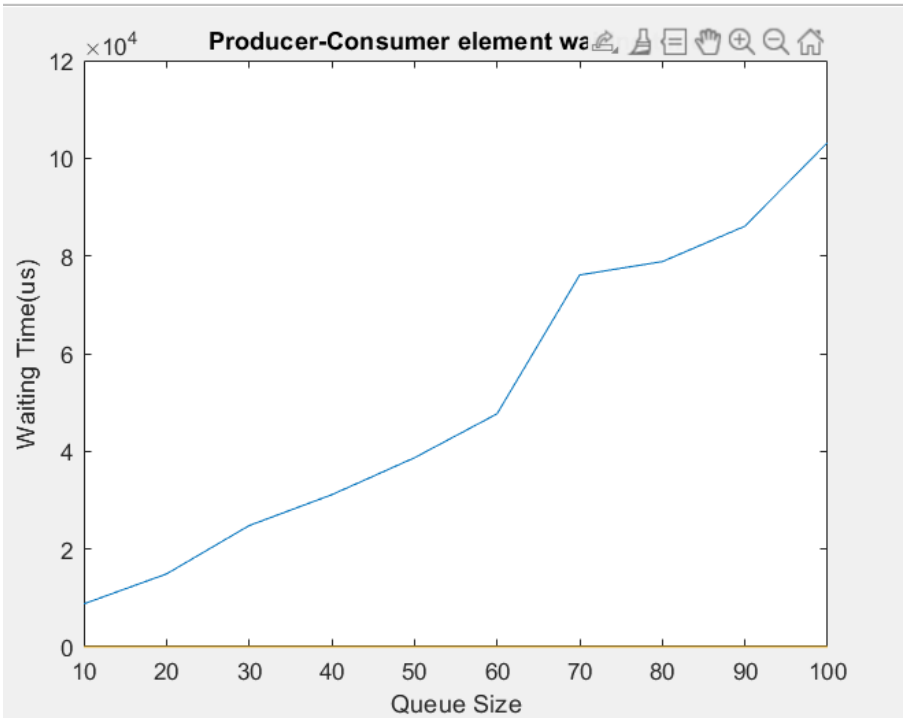

Figure1.5: Waiting time- queue size graph

Waiting time – number of consumers

On the next experiment, the queue and the producer have constant values (queue size = 20, number of producers = 10) and the variable was the number of consumers. In the figure 1.6 are presented the results. The more consumers the smaller waiting time, because the system has more consumers to execute the total work. After a number of 16 consumers, there is a saturation and there is no important improvement on the waiting time. So it does not make any sense to create more consumers for the system.