ECE 254: Lab 4

Producer Consumer Problem

Ramie Raufdeen & Sadman khan

2015

# Overview

In this lab, the objective was to use the POSIX API’s functionalities to solve the classic producer-consumer problem. The code was then used to generate timings for various factors such as the number of queued messages, initialization of the system, and the data transmission as a whole.

There is potential for bias with the data below as the *printf()* function could cause a delay in the program with its IO wait. This could mean that the data might be a bit skewed.

# Timing Data

## Initialization Time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Average System Initialization Time | | | | | |
| N\B | 1 | 2 | 4 | 8 | 10 |
| 20 | 0.000259 | 0.000267 | 0.000262 | 0.000259 | 0.000271 |
| 40 | 0.000279 | 0.000269 | 0.000262 | 0.000272 | 0.000276 |
| 80 | 0.000249 | 0.000262 | 0.00031 | 0.000262 | 0.000259 |
| 160 | 0.00027 | 0.000271 | 0.000263 | 0.000261 | 0.000259 |
| 320 | 0.000254 | 0.000249 | 0.00026 | 0.000265 | 0.00025 |
| Standard Deviation of System Initialization Time | | | | | |
| N\B | 1 | 2 | 4 | 8 | 10 |
| 20 | 0.000251 | 0.000263 | 0.000244 | 0.000243 | 0.000253 |
| 40 | 0.000279 | 0.000271 | 0.000256 | 0.000276 | 0.000293 |
| 80 | 0.000233 | 0.000257 | 0.001186 | 0.000252 | 0.000236 |
| 160 | 0.0003 | 0.000284 | 0.000254 | 0.000262 | 0.000253 |
| 320 | 0.000246 | 0.000221 | 0.000258 | 0.000259 | 0.000231 |

## Data Transmission Time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Average Data Transmission Time | | | | | |
| N\B | 1 | 2 | 4 | 8 | 10 |
| 20 | 0.001322 | 0.001347 | 0.001331 | 0.001476 | 0.001564 |
| 40 | 0.001413 | 0.001446 | 0.001457 | 0.001406 | 0.001399 |
| 80 | 0.001542 | 0.001525 | 0.001494 | 0.0015 | 0.001491 |
| 160 | 0.001782 | 0.001718 | 0.001664 | 0.001643 | 0.001627 |
| 320 | 0.002196 | 0.002108 | 0.00202 | 0.002013 | 0.002094 |
| Standard Deviation of Data Transmission Time | | | | | |
| N\B | 1 | 2 | 4 | 8 | 10 |
| 20 | 0.000425 | 0.000429 | 0.00043 | 0.000493 | 0.006447 |
| 40 | 0.000492 | 0.000484 | 0.000556 | 0.00045 | 0.000492 |
| 80 | 0.000469 | 0.000503 | 0.00048 | 0.000477 | 0.000477 |
| 160 | 0.000595 | 0.000549 | 0.00047 | 0.000514 | 0.000499 |
| 320 | 0.000681 | 0.000626 | 0.000591 | 0.000594 | 0.000584 |

## Graphs

### System Initialization Time

The plot above shows that for any fixed N and a change in buffer size, the resulting initialization time in seconds. Analyzing this graph, one can note there is a steady range of approximately 0.25ms to 0.29ms (excluding the outlier of 0.31ms from n=80) for the initialization time regardless of buffer size. One can conclude from the graph that there is no relationship between the buffer size, number of messages, and the initialization time.

### Data Transmission Time

For a fixed N, there is a change in data transmission time as the buffer size changes. In the plot, there is a decrease in average data transmission time as the buffer size increases.

### Histogram

# Further Discussion & Conclusions

There are three key relationships to conclude from this lab.

## Altering the number of messages for a fixed buffer size

If you hold B as a constant and increase the N, It will result in longer times for transmission. This makes sense because when the message queue is filled, the producer has to stop producing and wait for the consumer to finish consuming all the elements from the queue. Likewise when the consumer has consumed all the elements from the queue, it has to block it self until there are more messages in the queue again. This causes the system to have to do multiple context switches, which obviously takes a lot of time.

### Example

|  |  |
| --- | --- |
| N\B | 1 |
| 20 | 0.001322 |
| 40 | 0.001413 |
| 80 | 0.001542 |
| 160 | 0.001782 |
| 320 | 0.002196 |

## Altering the buffer size with a fixed number of messages

If you hold N as a constant and increase the value of B, It will result in shorter times for transmission. This makes sense because more of the messages can be transferred in one go, without the processes having to block themselves. Likewise this results in a lower number of context switches, which makes the transition time shorter over all.

### Example

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N\B | 1 | 2 | 4 | 8 | 10 |
| 320 | 0.002196 | 0.002108 | 0.00202 | 0.002013 | 0.002094 |

## Effect on system initialization time

Regardless of what you set as N or B, it will have no effect in system initialization time. This is because the system initiation time is simply the time between right before the fork and the time the producer started producing. In other words, it’s a measurement of how long it took to fork a child process.

## Conclusion

Using all the data we had for data transmission, we were able to plot a 3-dimensional surface graph to show the relationship between the numbers of messages sent, buffer size, and the corresponding transmission time. Each color represents a specific height range; from lowest to highest it goes grey, yellow, and blue respectively. The height of the surface increases as it travels along the message size axis, we see that it goes from grey, to yellow, to blue color. However as we move along the buffer size axis, the surface height stays nearly the same or gets smaller. Thus, we can conclude that both the number of messages as well as buffer size play a role in the timing of data transmissions for the producer-consumer problem.

# Source Code Listing (Appendix)

## Producer.c

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <mqueue.h>
4. #include <sys/stat.h>
5. #include <sys/types.h>
6. #include <sys/wait.h>
7. #include <unistd.h>
8. #include <errno.h>
9. #include <sys/time.h>
10. #include <time.h>
11. #include <string.h>


15. **int** spawn (**char**\* program, **char**\*\* arg\_list, mqd\_t qdes) {
17. arg\_list[0] = program;
19. pid\_t child\_pid;
20. child\_pid = fork();
22. **if** (child\_pid != 0) {
23. **return** child\_pid;
24. } **else** {
25. // ./consume is the compiled file of consumer.c
26. execvp ("./consume", arg\_list);
28. fprintf(stderr, "an error occurred in exec\n" );
29. abort ();
30. }
31. }
33. **double** time\_in\_seconds(){
34. **struct** timeval tv;
35. **double** t1;
37. gettimeofday(&tv, NULL);
38. t1 = tv.tv\_sec + tv.tv\_usec/1000000.0;
40. **return** t1;
41. }

44. **int** main(**int** argc, **char** \*argv[]) {
46. /\* ensure valid input \*/
48. **if** (argc < 3){
49. printf("Invalid Inputs\n");
50. exit(0);
51. }

54. /\* process the inputs \*/
56. **int** number\_of\_messages;
57. **int** queue\_size;
59. number\_of\_messages = atoi(argv[1]);
60. queue\_size = atoi(argv[2]);
62. **if** (number\_of\_messages < 0 || queue\_size < 0) {
63. printf("Invalid inputs");
64. exit(1);
65. }
67. /\* setting up and trying to open the message queue for the producer \*/
69. **struct** mq\_attr attr;                // message queue attributes
70. mqd\_t qdes;                         // message queue struct
71. mode\_t mode = S\_IRUSR | S\_IWUSR;    // some kind of mode?
73. attr.mq\_maxmsg  = queue\_size;
74. attr.mq\_msgsize = **sizeof**(**int**);
75. attr.mq\_flags   = 0;                // a blocking queue
77. **char** \*qname = "/mailbox\_lab4";
79. //open the queue
80. qdes  = mq\_open(qname, O\_RDWR | O\_CREAT, mode, &attr);
81. **if** (qdes == -1 ) {
82. perror("mq\_open() failed");
83. exit(1);
84. }
86. /\* forking a child proccess to be consumer \*/
88. **double** starting\_time = time\_in\_seconds();
89. pid\_t child\_pid = spawn("consumer", argv, qdes);
90. **double** time\_at\_first\_int = time\_in\_seconds();
92. /\* sending random numbers from the producer into the message queue \*/
94. **int** i = 0;
95. **int** \*random\_number;
96. **int** message\_priority = 0;
97. srand(time(0));
99. **for**(i = 0; i < number\_of\_messages; i++ ){
100. random\_number  = rand() % 100;
102. //send message to the queue
103. **if**(mq\_send(qdes, (**char**\*)&random\_number, **sizeof**(**int**), message\_priority) == -1){
104. perror("mq\_send() failed");
105. }
106. }
108. /\* ensure that the consumer has consumed all the ints \*/
110. **int** return\_status;
111. waitpid(child\_pid, &return\_status, 0);
113. **if**(return\_status == 0) {
114. **double** ending\_time = time\_in\_seconds();
115. **double** time\_to\_initialize = time\_at\_first\_int - starting\_time ;
116. **double** time\_to\_consume = ending\_time - time\_at\_first\_int;

119. printf("Time to initialize system: %f\n", time\_to\_initialize );
120. printf("Time to transmit data: %f\n", time\_to\_consume);
122. } **else** {
123. printf("There has been an error in the child process");
124. exit(1);
125. }


129. /\* closing and unlinking the message queue \*/
131. **if** (mq\_close(qdes) == -1){
132. perror("mq\_close() failed");
133. exit(2);
134. }
136. **if** (mq\_unlink(qname) != 0) {
137. perror("mq\_unlink() failed");
138. exit(3);
139. }
141. **return** 0;


145. }

## Consumer.c

1. #include <stdbool.h>
2. #include <string.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <time.h>
6. #include <mqueue.h>
7. #include <sys/stat.h>
8. #include <signal.h>
10. **int** main(**int** argc, **char** \*argv[]) {
12. /\* processing inputs \*/
13. **int** number\_of\_messages = atoi(argv[1]);
14. **int** queue\_size = atoi(argv[2]);

17. /\* opening the message queue for the consumer \*/
19. mqd\_t qdes;
20. mode\_t mode = S\_IRUSR | S\_IWUSR;
21. **struct** mq\_attr attr;
23. attr.mq\_maxmsg  = queue\_size;
24. attr.mq\_msgsize = **sizeof**(**int**);
25. attr.mq\_flags   = 0;    /\* a blocking queue  \*/
27. **char** \*qname = "/mailbox\_lab4";
29. //open the queue
30. qdes  = mq\_open(qname, O\_RDONLY, mode, &attr);
31. **if** (qdes == -1 ) {
32. perror("mq\_open()");
33. exit(1);
34. }

37. /\* consume elements \*/
39. **int** i;
40. **int** message\_priority = 0;
41. **for**(i = 0; i < number\_of\_messages; i++){
42. **int** recieved\_message;
44. //receive message from the queue
45. **if** (mq\_receive(qdes, (**char** \*)&recieved\_message, **sizeof**(**int**), message\_priority) == -1){
46. printf("mq\_receive() failed\n");
47. **return** 1;
48. } **else** {
49. printf("%d is consumed\n", recieved\_message);
50. }
52. }
54. /\* closing the message queue on the consumer side \*/
56. **if** (mq\_close(qdes) == -1) {
57. perror("mq\_close() failed");
58. exit(2);
59. }
61. **return** 0;
63. }