Name: Huy Tran

Email: [hptran@pdx.edu](mailto:hptran@pdx.edu)

CS533

**Project 5: Report**

1. **A description of what I did and how I tested it**
   1. **Stage 0**

* **Configuration:**
  + Test 1: Test lock (main\_test\_lock.c)
    - 5 user threads, each increments a shared counter by 1 three times
  + Test 2: Test condition variable (main\_test\_condition.c)
    - 2 producers, each of them puts ‘0’,’1’,’2’ to a shared buffer
    - 1 consumer who gets a character from the buffer (6 times)
    - Buffer’s size: 3 (‘X’: empty slot)
* **Tests:**
  + Test 1:

gcc \*.s threadmap.c scheduler.c queue.c main\_test\_lock.c

./a.out

* + Test 2:

gcc \*.s threadmap.c scheduler.c queue.c main\_test\_condition.c

./a.out

* **Test results**

|  |  |
| --- | --- |
| Test 1 | Test 2 |
| thread 1: shared\_counter = 1  thread 2: shared\_counter = 2  thread 1: shared\_counter = 3  thread 3: shared\_counter = 4  thread 4: shared\_counter = 5  thread 2: shared\_counter = 6  thread 5: shared\_counter = 7  thread 1: shared\_counter = 8  thread 3: shared\_counter = 9  thread 4: shared\_counter = 10  thread 2: shared\_counter = 11  thread 5: shared\_counter = 12  thread 3: shared\_counter = 13  thread 4: shared\_counter = 14  thread 5: shared\_counter = 15 | Producer 1 puts: 0, buffer: 0XX  Producer 2 puts: 0, buffer: 00X  Producer 1 puts: 1, buffer: 001  Consumer 1 gets: 0, buffer: X01  Producer 2 puts: 1, buffer: 101  Consumer 1 gets: 0, buffer: 1X1  Producer 2 puts: 2, buffer: 121  Consumer 1 gets: 1, buffer: 12X  Producer 1 puts: 2, buffer: 122  Consumer 1 gets: 1, buffer: X22  Consumer 1 gets: 2, buffer: XX2  Consumer 1 gets: 2, buffer: XXX |
| The results are as expected | The results are as expected |

* 1. **Stage 1**

The configurations and tests are the same as the stage 0. I modified to print kernel thread ID for testing also.

* **Test results**

|  |  |
| --- | --- |
| **Test 1** | **Test 2** |
| kernel\_thread 29973: user\_thread 1: shared\_counter = 1  kernel\_thread 29973: user\_thread 1: shared\_counter = 2  kernel\_thread 29973: user\_thread 1: shared\_counter = 2  kernel\_thread 29974: user\_thread 2: shared\_counter = 2  kernel\_thread 29973: user\_thread 3: shared\_counter = 3  kernel\_thread 29973: user\_thread 5: shared\_counter = 4  kernel\_thread 29973: user\_thread 5: shared\_counter = 4  kernel\_thread 29974: user\_thread 4: shared\_counter = 4  kernel\_thread 29973: user\_thread 1: shared\_counter = 5  kernel\_thread 29974: user\_thread 2: shared\_counter = 6  kernel\_thread 29974: user\_thread 2: shared\_counter = 6  kernel\_thread 29973: user\_thread 3: shared\_counter = 7  kernel\_thread 29974: user\_thread 5: shared\_counter = 8  kernel\_thread 29974: user\_thread 5: shared\_counter = 8  kernel\_thread 29973: user\_thread 4: shared\_counter = 9  kernel\_thread 29974: user\_thread 5: shared\_counter = 8  kernel\_thread 29973: user\_thread 4: shared\_counter = 9  kernel\_thread 29974: user\_thread 2: shared\_counter = 10  kernel\_thread 29973: user\_thread 3: shared\_counter = 11  kernel\_thread 29974: user\_thread 5: shared\_counter = 12  ERROR in scheduling  ERROR in scheduling  ERROR in scheduling  Segmentation fault | kernel\_thread: 359, Producer 1 puts: 0, buffer: 0XX  kernel\_thread: 359, Producer 1 puts: 0, buffer: 0XX  kernel\_thread: 360, Producer 2 puts: 0, buffer: 00X  kernel\_thread: 360, Consumer 1 gets: 0, buffer: X0X  kernel\_thread: 360, Producer 2 puts: 1, buffer: X01  kernel\_thread: 360, Consumer 1 gets: 0, buffer: XX1  kernel\_thread: 360, Producer 2 puts: 2, buffer: 2X1  kernel\_thread: 360, Consumer 1 gets: 1, buffer: 2XX  kernel\_thread: 359, Producer 1 puts: 1, buffer: 21X  kernel\_thread: 360, Consumer 1 gets: 2, buffer: X1X  kernel\_thread: 359, Producer 1 puts: 2, buffer: X12  kernel\_thread: 360, Consumer 1 gets: 1, buffer: XX2  Segmentation fault |
| The results are nondeterministic and the program’s execution crashed. | The results are nondeterministic and the program’s execution crashed. |

* 1. **Stage 2**
* **Configuration:**
  + Test 1: Test spin lock ([spinlock\_test.c](http://web.cecs.pdx.edu/~kstew2/cs533/project/assign5/spinlock_test.c))
    - Repeat 10 times
  + Test 2 and 3 are the same as test 1 and 2 in stage 0.
* **Tests:**
  + Test 1:

gcc -I ~kstew2/local/include spinlock\_test.c

./a.out

* **Test results**

For test 1, the results are the same (success!).

For test 2 and 3,

|  |  |
| --- | --- |
| **Test 1** | **Test 2** |
| kernel\_thread 4248: user\_thread 1: shared\_counter = 1  kernel\_thread 4247: user\_thread 2: shared\_counter = 2  kernel\_thread 4247: user\_thread 3: shared\_counter = 3  kernel\_thread 4248: user\_thread 4: shared\_counter = 4  kernel\_thread 4247: user\_thread 5: shared\_counter = 5  kernel\_thread 4247: user\_thread 1: shared\_counter = 6  kernel\_thread 4248: user\_thread 2: shared\_counter = 7  kernel\_thread 4247: user\_thread 3: shared\_counter = 8  kernel\_thread 4248: user\_thread 4: shared\_counter = 9  kernel\_thread 4248: user\_thread 5: shared\_counter = 10  kernel\_thread 4247: user\_thread 1: shared\_counter = 11  kernel\_thread 4248: user\_thread 2: shared\_counter = 12  kernel\_thread 4248: user\_thread 3: shared\_counter = 13  kernel\_thread 4248: user\_thread 4: shared\_counter = 14  Segmentation fault | kernel\_thread: 4566, Producer 1 puts: 0, buffer: 0XX  kernel\_thread: 4567, Producer 2 puts: 0, buffer: 001  kernel\_thread: 4566, Producer 1 puts: 1, buffer: 001  kernel\_thread: 4567, Consumer 1 gets: 0, buffer: X01  kernel\_thread: 4566, Producer 1 puts: 2, buffer: 201  kernel\_thread: 4567, Consumer 1 gets: 0, buffer: 2X1  kernel\_thread: 4566, Producer 2 puts: 1, buffer: 211  kernel\_thread: 4566, Consumer 1 gets: 1, buffer: 21X  kernel\_thread: 4566, Producer 2 puts: 2, buffer: 212  kernel\_thread: 4566, Consumer 1 gets: 2, buffer: X12  kernel\_thread: 4566, Consumer 1 gets: 1, buffer: XX2  kernel\_thread: 4566, Consumer 1 gets: 2, buffer: XXX  Segmentation fault |
| The results are still nondeterministic and the program’s execution crashed. | The results are still nondeterministic and the program’s execution crashed. |

* 1. **Stage 3**

In this stage, besides protecting ready list, I also redefine printf with safe\_print function to synchronize printf. For testing the schedule in this stage:

* **Configuration:** I created a new program (main\_test\_spinlock.c)
  + 5 user threads, each increments a shared counter by 1 (three times)
  + Manipulating shared counter using spinlock (instead of blocking lock)
* **Tests:**

gcc -I ~kstew2/local/include \*.s threadmap.c scheduler.c queue.c main\_test\_spinlock.c

./a.out

* **Test results:**

|  |  |
| --- | --- |
| **Test 1** | **Test 2** (comment out spinlock\_lock and spinlock\_unlock) |
| kernel\_thread 4120: user\_thread 1: shared\_counter = 1  kernel\_thread 4121: user\_thread 2: shared\_counter = 2  kernel\_thread 4120: user\_thread 3: shared\_counter = 3  kernel\_thread 4121: user\_thread 1: shared\_counter = 4  kernel\_thread 4120: user\_thread 4: shared\_counter = 5  kernel\_thread 4121: user\_thread 2: shared\_counter = 6  kernel\_thread 4120: user\_thread 3: shared\_counter = 7  kernel\_thread 4121: user\_thread 5: shared\_counter = 8  kernel\_thread 4120: user\_thread 1: shared\_counter = 9  kernel\_thread 4121: user\_thread 4: shared\_counter = 10  kernel\_thread 4120: user\_thread 2: shared\_counter = 11  kernel\_thread 4121: user\_thread 3: shared\_counter = 12  kernel\_thread 4120: user\_thread 5: shared\_counter = 13  kernel\_thread 4121: user\_thread 4: shared\_counter = 14  kernel\_thread 4120: user\_thread 5: shared\_counter = 15 | kernel\_thread 4535: user\_thread 1: shared\_counter = 1  kernel\_thread 4536: user\_thread 2: shared\_counter = 1  kernel\_thread 4535: user\_thread 3: shared\_counter = 2  kernel\_thread 4536: user\_thread 1: shared\_counter = 2  kernel\_thread 4536: user\_thread 2: shared\_counter = 3  kernel\_thread 4535: user\_thread 4: shared\_counter = 3  kernel\_thread 4536: user\_thread 3: shared\_counter = 4  kernel\_thread 4535: user\_thread 5: shared\_counter = 4  kernel\_thread 4536: user\_thread 1: shared\_counter = 5  kernel\_thread 4535: user\_thread 2: shared\_counter = 5  kernel\_thread 4536: user\_thread 4: shared\_counter = 6  kernel\_thread 4535: user\_thread 3: shared\_counter = 6  kernel\_thread 4536: user\_thread 5: shared\_counter = 7  kernel\_thread 4535: user\_thread 4: shared\_counter = 7  kernel\_thread 4536: user\_thread 5: shared\_counter = 8 |
| The results are deterministic and correct. (This test is repeated more than 10 times) | The results are nondeterministic (as expected) because there is no synchronization. |

* 1. **Stage 4**
* Configurations:
  + Test 1 and 2 are the same as the stage 0.
  + Test 3: repeats test1 100 times (except the shared counter is increased 1, 100 times) by each of five user threads.
* Tests:
  + Test 3:

gcc -I ~kstew2/local/include \*.s threadmap.c scheduler.c queue.c main\_test\_lock\_repeat.c

./a.out

* Test results:

|  |  |
| --- | --- |
| **Test 1** | **Test 2** |
| kernel\_thread 30097: user\_thread 1: shared\_counter = 1  kernel\_thread 30098: user\_thread 2: shared\_counter = 2  kernel\_thread 30097: user\_thread 3: shared\_counter = 3  kernel\_thread 30098: user\_thread 4: shared\_counter = 4  kernel\_thread 30098: user\_thread 5: shared\_counter = 5  kernel\_thread 30098: user\_thread 1: shared\_counter = 6  kernel\_thread 30097: user\_thread 2: shared\_counter = 7  kernel\_thread 30098: user\_thread 3: shared\_counter = 8  kernel\_thread 30098: user\_thread 4: shared\_counter = 9  kernel\_thread 30098: user\_thread 5: shared\_counter = 10  kernel\_thread 30098: user\_thread 1: shared\_counter = 11  kernel\_thread 30098: user\_thread 2: shared\_counter = 12  kernel\_thread 30098: user\_thread 3: shared\_counter = 13  kernel\_thread 30097: user\_thread 4: shared\_counter = 14  kernel\_thread 30098: user\_thread 5: shared\_counter = 15 | kernel\_thread: 29323, Producer 1 puts: 0, buffer: 00X  kernel\_thread: 29324, Producer 2 puts: 0, buffer: 00X  kernel\_thread: 29323, Consumer 1 gets: 0, buffer: X0X  kernel\_thread: 29324, Producer 1 puts: 1, buffer: X01  kernel\_thread: 29323, Producer 2 puts: 1, buffer: 101  kernel\_thread: 29324, Consumer 1 gets: 0, buffer: 1X1  kernel\_thread: 29323, Producer 1 puts: 2, buffer: 121  kernel\_thread: 29324, Consumer 1 gets: 1, buffer: 12X  kernel\_thread: 29323, Producer 2 puts: 2, buffer: 122  kernel\_thread: 29324, Consumer 1 gets: 1, buffer: X22  kernel\_thread: 29324, Consumer 1 gets: 2, buffer: XX2  kernel\_thread: 29324, Consumer 1 gets: 2, buffer: XXX |
| The results are deterministic and correct. (This test is repeated more than 10 times) | The results shows correct behavior of producers and consumers (as stage 0) |

* **Test 3:** The result is correct all of the iterations.
  1. **Stage 5**
* **Configuration:**

I modified scheduler.c and scheduler.h to allow freeing thread’s memory spaces and setting up barriers. Besides, I modified parallel mergesort from this [website](http://sc12.supercomputing.org/hpceducator/PythonForParallelism/codes/parallelMergesort.c) to test my scheduler’s scalability.

My adapted version of the parallel megrgesort : parallelMergesort.c

* **Tests:**

gcc -I ~kstew2/local/include \*.s threadmap.c scheduler.c queue.c parallelMergesort.c

./a.out num\_kernel\_threads [size of randomlist] (default size: 1M)

* **Test results:**

The average time (second) for performing parallel merge sort. (I repeat each experiment 5 times to get the average result). You can view my data in this file (stage5\_experiment\_data.xls)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Size of random list** | **Number of Kernel threads** | | | | |
|  | **1** | **2** | **4** | **8** | **16** |
| **10^6** | 0.2611284 | 0.1270724 | 0.0939548 | 0.0905756 | 0.1709304 |

1. **The results of your efforts from Part 5**

* **Compose a written report (at least two pages, in your own words) analysing the scalability and performance of this system. Identify some of the scalability bottlenecks and propose some design alternatives that you believe would improve performance. Be sure to provide sufficient detail about why you think your alternatives would work.**
* **Implement an optimization that you will believe will improve the system's performance, and conduct experiments to prove that it actually does. Produce a brief written report of your results.**