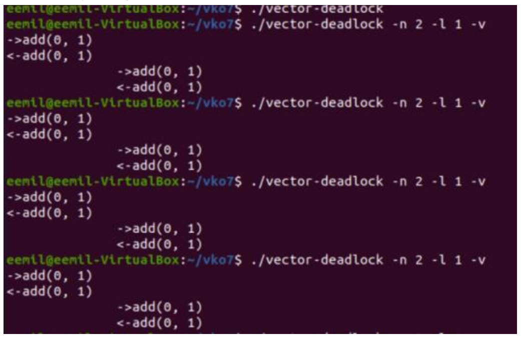
Viikkotehtävä 7.

Eemil Aspholm 0567753

CT30A3370 Käyttöjärjestelmät ja systeemiohjelmointi

1. Running the program with command ./vector-deadlock -n 2 -l 1 -v, does not effect to the output. Output does not seem to change from run to run. Only thing that might change is the order of threads in the program, but this does not affect the output syntax.

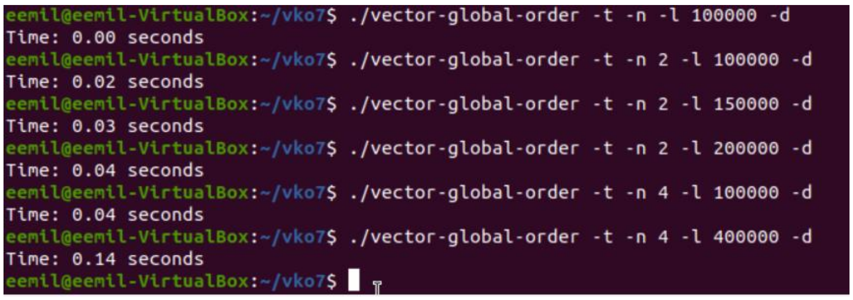
   ```
   eemil@eemil-VirtualBox:~/vko7$ ./vector-deadlock
   eemil@eemil-VirtualBox:~/vko7$ ./vector-deadlock -n 2 -l 1 -v
   ->add(0, 1)
   <-add(0, 1)
                 ->add(0, 1)
                 <-add(0, 1)
   eemil@eemil-VirtualBox:~/vko7$ ./vector-deadlock -n 2 -l 1 -v
   ->add(0, 1)
   <-add(0, 1)
                 ->add(0, 1)
                 <-add(0, 1)
   eemil@eemil-VirtualBox:~/vko7$ ./vector-deadlock -n 2 -l 1 -v
   ->add(0, 1)
   <-add(0, 1)
                 ->add(0, 1)
                 <-add(0, 1)
   eemil@eemil-VirtualBox:~/vko7$ ./vector-deadlock -n 2 -l 1 -v
   ->add(0, 1)
   <-add(0, 1)
                 ->add(0, 1)
                 <-add(0, 1)
   ```

   When adding more loops into the code it will deadlock sometimes. I couldn't get program to deadlock with loop values lower than 1000, and after increasing the loop numbers from 1000 to 10000 the program still did not deadlock always. (Couldn't add a snip, since the output wont fit in a single picture)

   Increasing the number of threads will make the program perform more add operations, the program does not seem to deadlock with small loop values but with bigger ones it deadlocks. Also, running the program with only 1 or 0 thread values will ensure that no deadlocks will occur. (Single or none thread can deadlock)

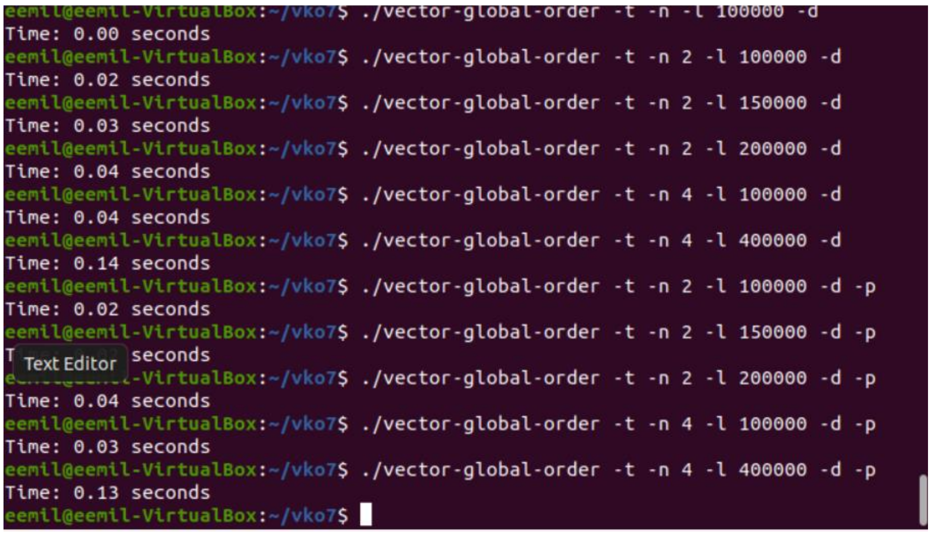
2. When inspecting the vector-global-order.c file, we can see that the code avoids deadlocks because the order the locks are taken into the Pthread_mutex_lock() function is in total order, which is defined by the virtual memory address of the vector structure. The special case is intended for situations where the destination and source are the same and only one lock is required. Without handling the special case, the program tries to acquire a lock that's already been held, which will lead into guaranteed deadlock situation.

   As we can see from the picture below, the number of loops (-l value) and the number of threads will increase the total run time.

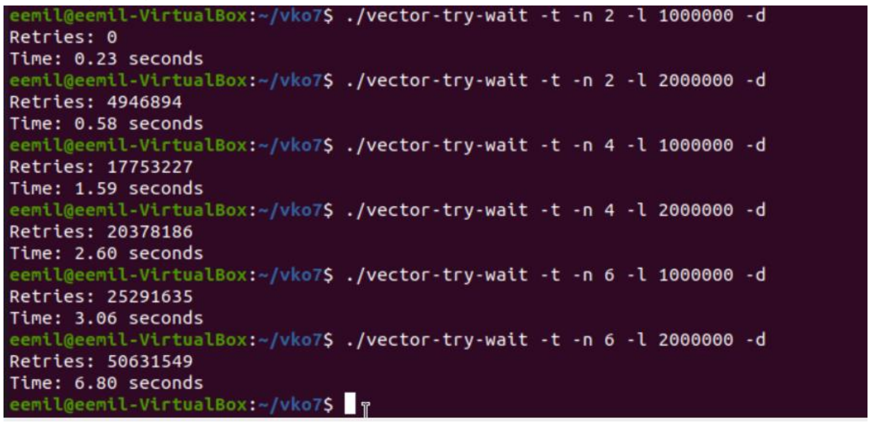
   ```
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n -l 100000 -d
   Time: 0.00 seconds
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 100000 -d
   Time: 0.02 seconds
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 150000 -d
   Time: 0.03 seconds
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 200000 -d
   Time: 0.04 seconds
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 100000 -d
   Time: 0.04 seconds
   eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 400000 -d
   Time: 0.14 seconds
   eemil@eemil-VirtualBox:~/vko7$
   ```

When turning on the parallelism flag (-p) we can see that the performance is getting better, this is simply because the threads doesn't have to wait on the same two locks. (The biggest performance difference can be seen when the number of threads is increased not so much when the loop number is only changed)

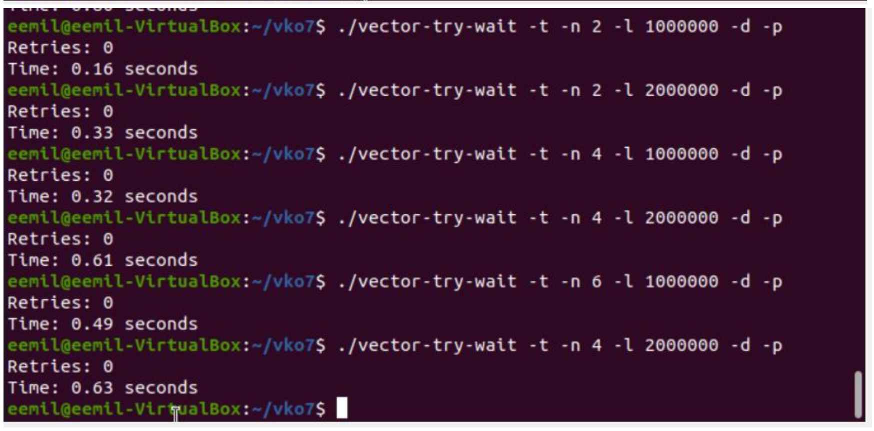


```
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n -l 100000 -d
Time: 0.00 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 100000 -d
Time: 0.02 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 150000 -d
Time: 0.03 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 200000 -d
Time: 0.04 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 100000 -d
Time: 0.04 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 400000 -d
Time: 0.14 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 100000 -d -p
Time: 0.02 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 150000 -d -p
T Text Editor seconds
e-VirtualBox:~/vko7$ ./vector-global-order -t -n 2 -l 200000 -d -p
Time: 0.04 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 100000 -d -p
Time: 0.03 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-global-order -t -n 4 -l 400000 -d -p
Time: 0.13 seconds
eemil@eemil-VirtualBox:~/vko7$
```

3. After viewing the file vector-try-wait.c, I would say that the first call to the pthread_mutex_trylock() is required. This is because without that call the destination vector is not going to be locked.



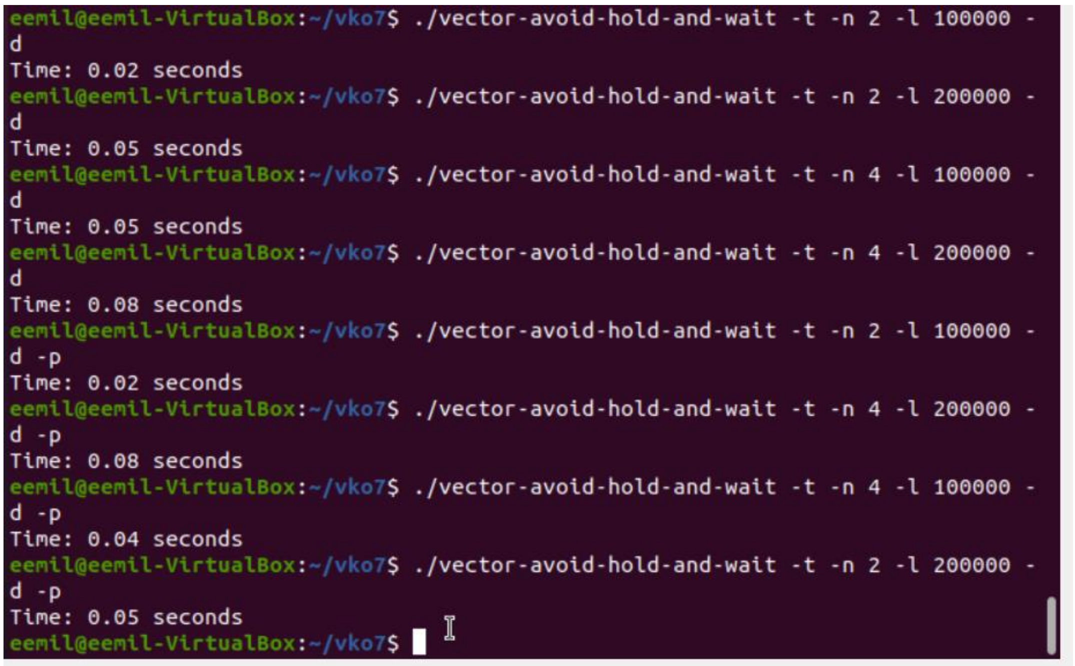
```
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 2 -l 1000000 -d
Retries: 0
Time: 0.23 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 2 -l 2000000 -d
Retries: 4946894
Time: 0.58 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 4 -l 1000000 -d
Retries: 17753227
Time: 1.59 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 4 -l 2000000 -d
Retries: 20378186
Time: 2.60 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 6 -l 1000000 -d
Retries: 25291635
Time: 3.06 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 6 -l 2000000 -d
Retries: 50631549
Time: 6.80 seconds
eemil@eemil-VirtualBox:~/vko7$
```



```
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 2 -l 1000000 -d -p
Retries: 0
Time: 0.16 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 2 -l 2000000 -d -p
Retries: 0
Time: 0.33 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 4 -l 1000000 -d -p
Retries: 0
Time: 0.32 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 4 -l 2000000 -d -p
Retries: 0
Time: 0.61 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 6 -l 1000000 -d -p
Retries: 0
Time: 0.49 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-try-wait -t -n 4 -l 2000000 -d -p
Retries: 0
Time: 0.63 seconds
eemil@eemil-VirtualBox:~/vko7$
```

As we can see from the images above, the number of threads without parallelism will increase the time and amount of retries taken to complete the program. But with parallelism the program will not have retries and the time increasement will be more alike linear.

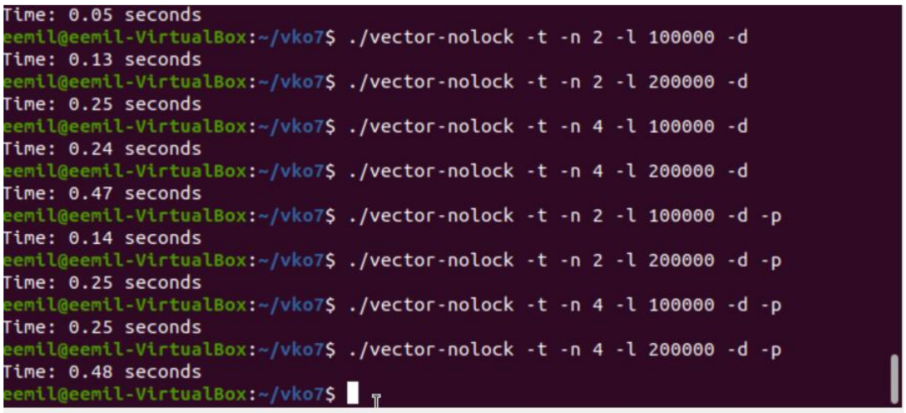
After inspecting the vector-avoid-hold-and-wait file I would say the problem in the approach is that it takes more time since it uses single global lock, when the others use more than one.



```
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 2 -l 100000 -
d
Time: 0.02 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 2 -l 200000 -
d
Time: 0.05 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 4 -l 100000 -
d
Time: 0.05 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 4 -l 200000 -
d
Time: 0.08 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 2 -l 100000 -
d -p
Time: 0.02 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 4 -l 200000 -
d -p
Time: 0.08 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 4 -l 100000 -
d -p
Time: 0.04 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-avoid-hold-and-wait -t -n 2 -l 200000 -
d -p
Time: 0.05 seconds
eemil@eemil-VirtualBox:~/vko7$
```

Comparing the run times with the vector-global-order and avoid-hold-and-wait they're similar without parallelism. But with parallelism flag added the values for the global locking (vector-avoid-hold-and-wait) are much faster (almost twice as fast as global order).

Last, when inspecting the vector-nolock.c file, we can find that the version does not use locks at all. The function uses fetch and add instead of locks, method atomically increments the contents of memory location.



```
Time: 0.05 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 2 -l 100000 -d
Time: 0.13 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 2 -l 200000 -d
Time: 0.25 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 4 -l 100000 -d
Time: 0.24 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 4 -l 200000 -d
Time: 0.47 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 2 -l 100000 -d -p
Time: 0.14 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 2 -l 200000 -d -p
Time: 0.25 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 4 -l 100000 -d -p
Time: 0.25 seconds
eemil@eemil-VirtualBox:~/vko7$ ./vector-nolock -t -n 4 -l 200000 -d -p
Time: 0.48 seconds
eemil@eemil-VirtualBox:~/vko7$
```

While comparing the nolock method to vector-global-order and global locking method we can see that it is a lot slower than global method and global locking method.