Operating System homework 1

platform

CPU information	memory size	Operating System	kernel version	machine type
Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz Core: 2 Logic processor: 4	12G	ubuntu 18.04	4.15.0-29-generic	physical machine

The measurement result

I generate random arrays of integers of different lengths. The execution times of the query elements measured under single process and single thread conditions were used as a control group to measure the execution times under different process and thread conditions.

To better represent the parallelization of multi-process and multi-linear processing, the test program uses "taskset -c 0, 1, 2, 3" to bind the 4 logical cores of the CPU.

The compile command： gcc calculateNum.c -lpthread -Wall -std=c99 -o calculateNum
Execution of orders： taskset -c 0,1,2,3 ./calculateNum

The parameters are listed in the table below.

information	
Array length	256, 262144, 2097152, 4194304
Number of processes	2, 4, 8
Number of threads	2, 4, 8

The following list shows the execution times for different array sizes

- **Array length: 256**

  Single process execution time: 0.001 (ms). the element number is: 31

  - multi-process and multi-threads execution time table list:

Number of processes/ threads	Used times (ms) (process)	Used times (ms) (thread with mutex)	Used times (ms) (thread without mutex)
2	0.246	0.082	0.221
4	0.705	0.668	0.082
8	0.695	0.178	0.208

Analysis:

1. With an array length of 256, the single-process program performs well because the calculation itself is not too complex.
2. Multi-processes require a certain amount of time for resource requests and process scheduling compared to single-processes, so performance is not as good as single-process programs.
3. Multi-threading requires a certain amount of thread scheduling time compared to single process so performance is not as good as single-process programs.

- **Array length: 262144**

  Single process execution time: 0.83 (ms) the element number is: 31.

  - multi-process and multi-threads execution time table list:

Number of processes/ threads	Used times (ms) (process)	Used times (ms) (thread with mutex)	Used times (ms) (thread without mutex)
2	0.71	0.384	0.377

Number of processes/ threads	Used times (ms) (process)	Used times (ms) (thread with mutex)	Used times (ms) (thread without mutex)
4	0.628	0.305	0.413
8	0.874	0.426	0.408

Analysis:

1. With an array length of 262144, multi-process resource requests and scheduling consume most of the time, so performance is about the same as single-process performance.
2. The time consumed by thread scheduling during multi-threaded execution is already negligible with an array length of 262144, so multi-threaded programs perform better than single and multi-process programs.
3. The use of locks in multithreading may result in a thread-wait situation, so the performance of programs using locks is not as good as that of programs not using locks.

- **Array length: 2097152**

  Single process execution time: 6.315 (ms) the element number is: 29.

  - multi-process and multi-threads execution time table list:

Number of processes/ threads	Used times (ms) (process)	Used times (ms) (thread with mutex)	Used times (ms) (thread without mutex)
2	4.832	4.03	2.859
4	3.153	2.232	2.10
8	3.878	2.819	2.312

Analysis:

1. When the array length reaches 2097152, the resource request and scheduling time of the multiprocess process is negligible, so the performance of the multiprocess program starts to exceed that of the single process.
2. Multi-threaded programs do not need to request additional resources compared to multi-processed programs so multi-threaded programs perform better than multi-processed programs.
3. Multi-threaded programs that do not use locks do not create a thread wait situation, so programs that do not use locks perform better than those that do.

- **Array length: 4194304**
  Single process execution time: 12.022 (ms) the element number is: 29.
  - multi-process and multi-threads execution time table list:

Number of processes/ threads	Used times (ms) (process)	Used times (ms) (thread with mutex)	Used times (ms) (thread without mutex)
2	10.114	5.702	5.65
4	5.477	5.198	4.365
8	6.43	4.671	4.154

Analysis:

1. When the array length reaches 4194304, the advantages of multi-processing and multi-threading start to become apparent.
2. Since the cpu has 4 logical cores, ideally 4 process programs and 4 threaded programs run on separate cores, so the best performance is achieved when the number of processes or threads is 4.
3. With a process or thread count of 2, CPU resources are

not fully utilized, so performance is not as good as with 4 processes or threads

4. With a process or thread count of 8, two processes or threads are allocated on each CPU and the CPU needs to do process or thread scheduling, causing the process or thread to go into a block state, so performance is not as good as with a process or thread count of 4.

Conclusion

1. At short array lengths (256), multi-processes and multi-threads do not perform as well as single-process results due to the time consumed by resource requests and process/thread scheduling.
2. When the array length is relatively large (2097152) the advantages of multi-processing and multi-threading begin to emerge. This is because resource requests and process/thread scheduling times are no longer significant compared to computation times.
3. Multi-threaded processes outperform multi-processes because they require resource requests and memory copies, which consume some time
4. In the case of an array length greater than 2097152, the execution time of 2 processes/threads is longer than 4 processes/threads because the number of logical cores of the computer is 4. With 2 processes/threads, the parallel processing power of the CPU is not fully utilized.
5. In the case of an array length greater than 2097152, the performance of 8 processes/threads is not as good as 4 processes/threads because in the 8 processes/threads case, there are only 4 logical cores of the computer and 2 threads need to be executed on each core, which causes the scheduling of processes or threads and makes some of them go into a block state.
6. Comparing locked and unlocked threads reveals that unlocked threads perform better because locks cause threads to wait

To summarise, ideally, a program should perform best when the number of processes or threads is equal to the number of cores in the cpu. Minimising the use of locks while ensuring correct results.