Systems Programming

Peter Dodd
2308057D

Assessed Exercise 2 - *Report*

Status

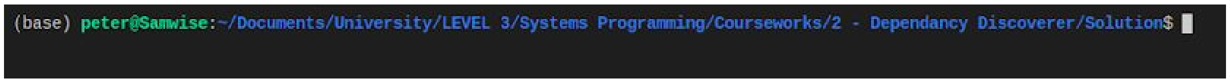
The solution I submitted implemented the multithreaded version of dependencyDiscoverer. The solution compiles without warnings or errors when using -Wall and -Werror on the stlinux servers. The solution implements use of the CRAWLER_THREADS environment variable, so can be run with a set amount of threads. This is automatically set to two if nothing is given for the variable. When tested against the expected output file with the command

```
../dependencyDiscoverer *.y *.l *.c | diff - output
```

the program successfully produces no output. Therefore, I believe the function to run perfectly with my multithreaded solution.

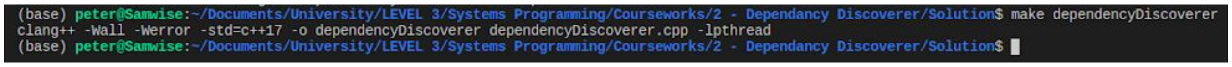
Build and Sequential + 1-Thread Runtimes

(a) Execution Path



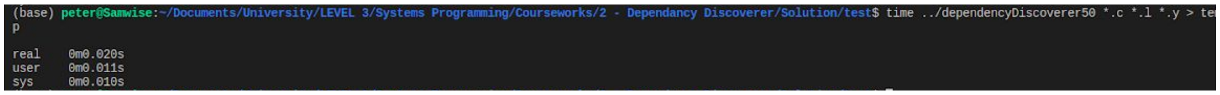
```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution$
```

(b) Compilation



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution$ make dependencyDiscoverer
clang++ -Wall -Werror -std=c++17 -o dependencyDiscoverer dependencyDiscoverer.cpp -lpthread
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution$
```

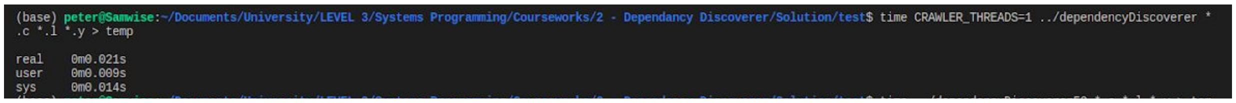
(c) Time to run sequential version of program (dependencyDiscoverer50 is sequential version)



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time ../dependencyDiscoverer50 *.c *.l *.y > te
p

real    0m0.020s
user    0m0.011s
sys     0m0.010s
```

(d) Time to run single threaded version of program

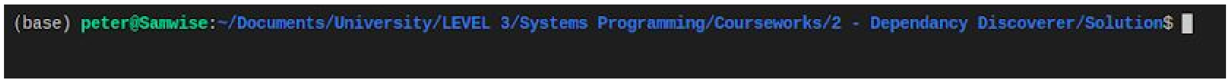


```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=1 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.021s
user    0m0.009s
sys     0m0.014s
```

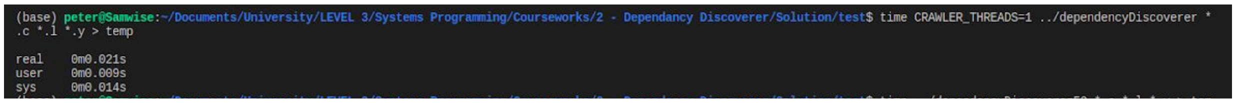
Local Runtime with Multiple Threads

(a) Execution Path



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution$
```

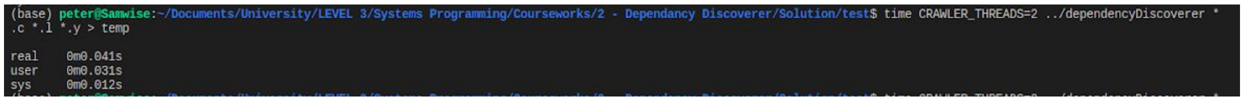
(b) 1-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=1 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.021s
user    0m0.009s
sys     0m0.014s
```

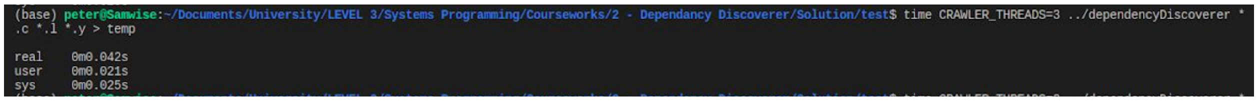
(c) 2-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=2 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.041s
user    0m0.031s
sys     0m0.012s
```

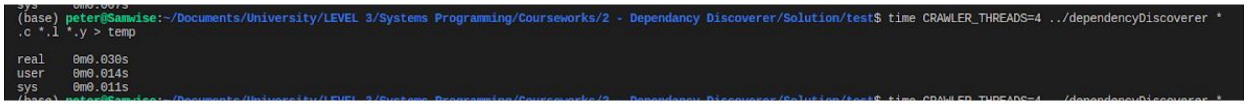
(d) 3-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=3 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.042s
user    0m0.021s
sys     0m0.025s
```

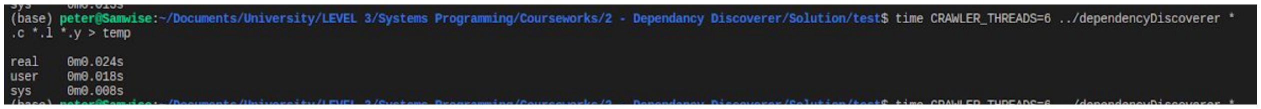
(e) 4-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=4 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.030s
user    0m0.014s
sys     0m0.011s
```

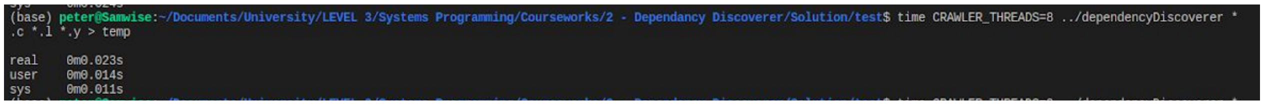
(f) 6-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=6 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.024s
user    0m0.018s
sys     0m0.008s
```

(g) 8-Thread



```
(base) peter@Samwise:~/Documents/University/LEVEL 3/Systems Programming/Courseworks/2 - Dependancy Discoverer/Solution/test$ time CRAWLER_THREADS=8 ../dependencyDiscoverer *
.c *.l *.y > temp

real    0m0.023s
user    0m0.014s
sys     0m0.011s
```

I believe the results for these times to be extremely unreliable and untelling. Although I was running on my linux system as lightly as possible, my laptop boasts an *extremely* modest Intel Pentium processor.

stlinux Thread Runtime Experiment Results

For the results, I took the real time value for each execution.

CRAWLER_ THREADS	Elapsed Time					
	1	2	3	4	6	8
Execution 1	0.053s	0.031s	0.024s	0.020s	0.020s	0.018s
Execution 2	0.086s	0.031s	0.025s	0.021s	0.020s	0.019s
Execution 3	0.051s	0.027s	0.023s	0.021s	0.018s	0.020s
Median	0.053s	0.031s	0.024s	0.021s	0.020s	0.019s

Discussion

Comparing these results using the schools multicore stlinux servers, to my own humble Pentium, it is clear that multiple cores drastically improve the performance of multithreaded programs. Not only can each core process different information simultaneously, but it means that a mutex lock has less impact on the performance, as different cores can carry on processing other information, rather than basically halting processing on a lower-cored computer.

The elapsed time for the is a lot more expected for these results. It can be seen that when changing between 1 and 2 threads, the impact is big on performance. However, past that, the performance only improves fractionally. This is because I believe the system reaches a bottleneck, where the program is pretty much peaking at maximum efficiency for the capability of the processor at around 4 or 6 threads.