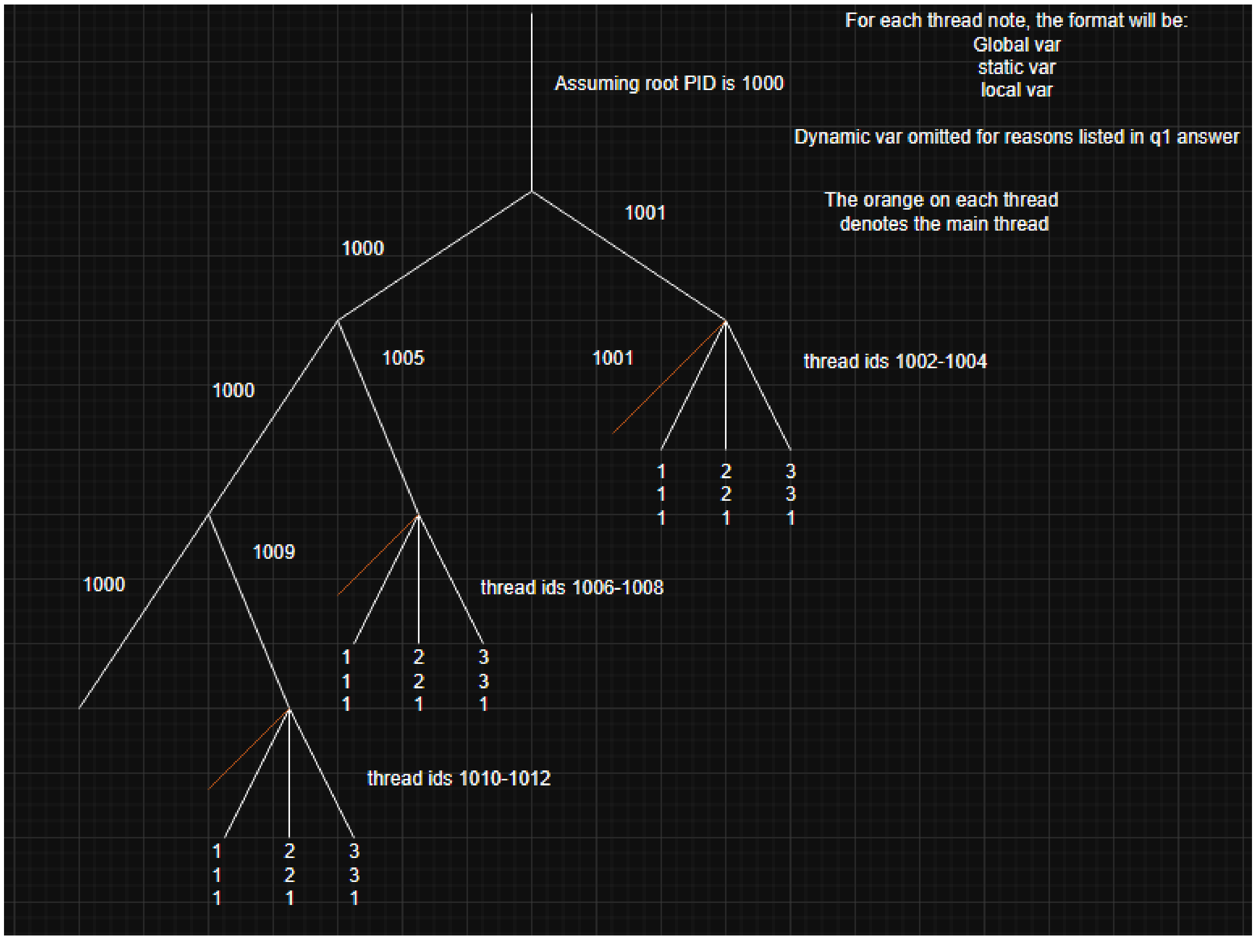
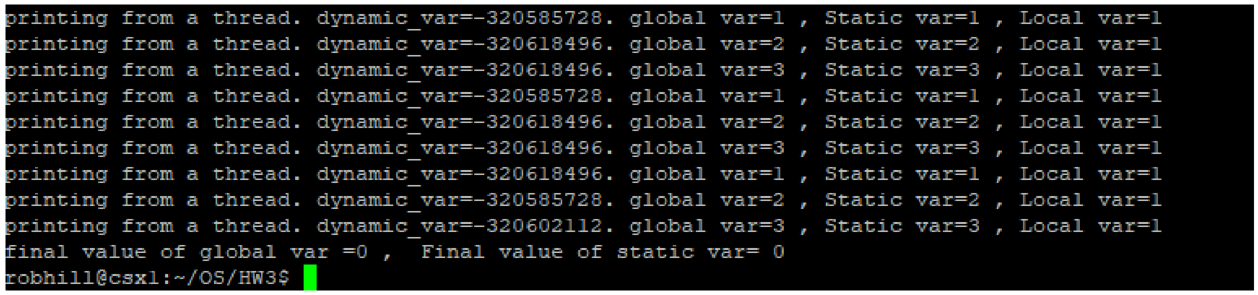
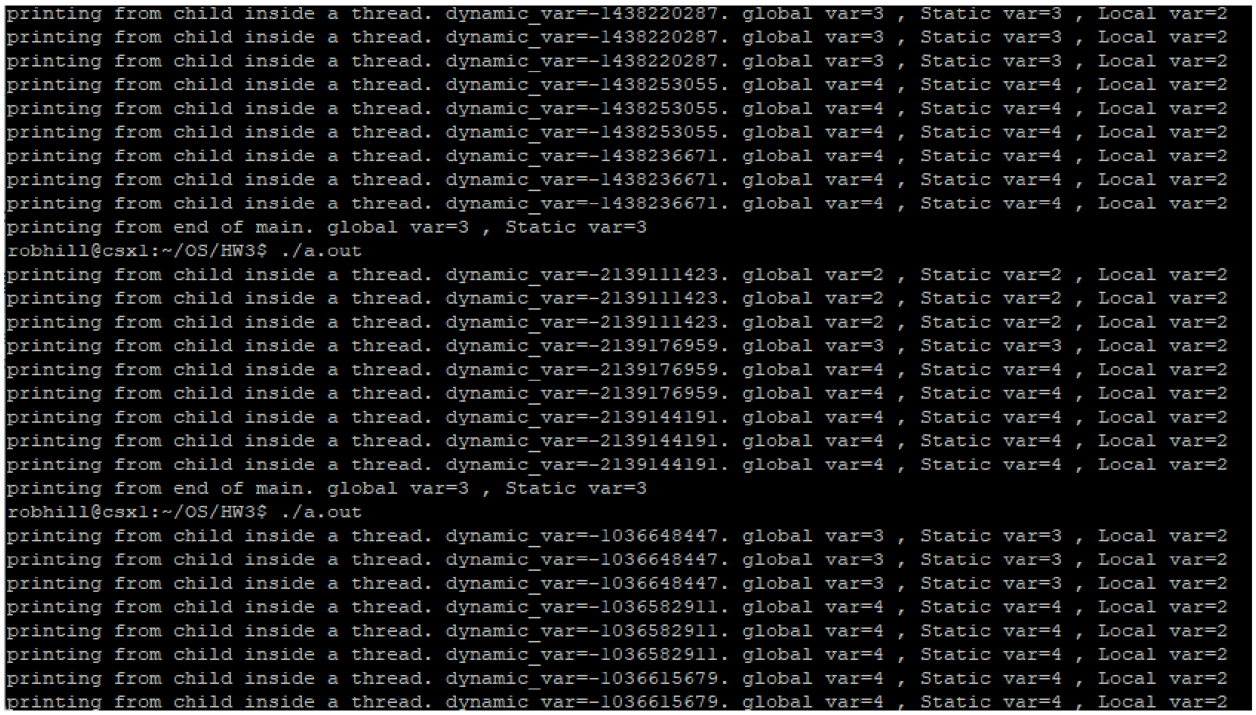
1. For all threads, the value of dynamic var at the end (after freeing) is essentially random because we have a dangling pointer to dynamic var after the memory space has been freed.
   Local var doesn't increment between threads because it is a local variable within each thread.
   global and static variables follow each other because no threads are accessing memory at the same time and overwriting each other. They also don't increment between forks because they are stored on each fork independently, so they reset back to 0 at each fork.
   This also explains why the final print shows 0 for both global and static var


For each thread note, the format will be:
Global var
static var
local var
Assuming root PID is 1000
Dynamic var omitted for reasons listed in q1 answer
The orange on each thread denotes the main thread
1000
1001
1000
1005
1001
thread ids 1002-1004
1 1 1
2 2 1
3 3 1
1000
1009
thread ids 1006-1008
1 1 1
2 2 1
3 3 1
1000
thread ids 1010-1012
1 1 1
2 2 1
3 3 1



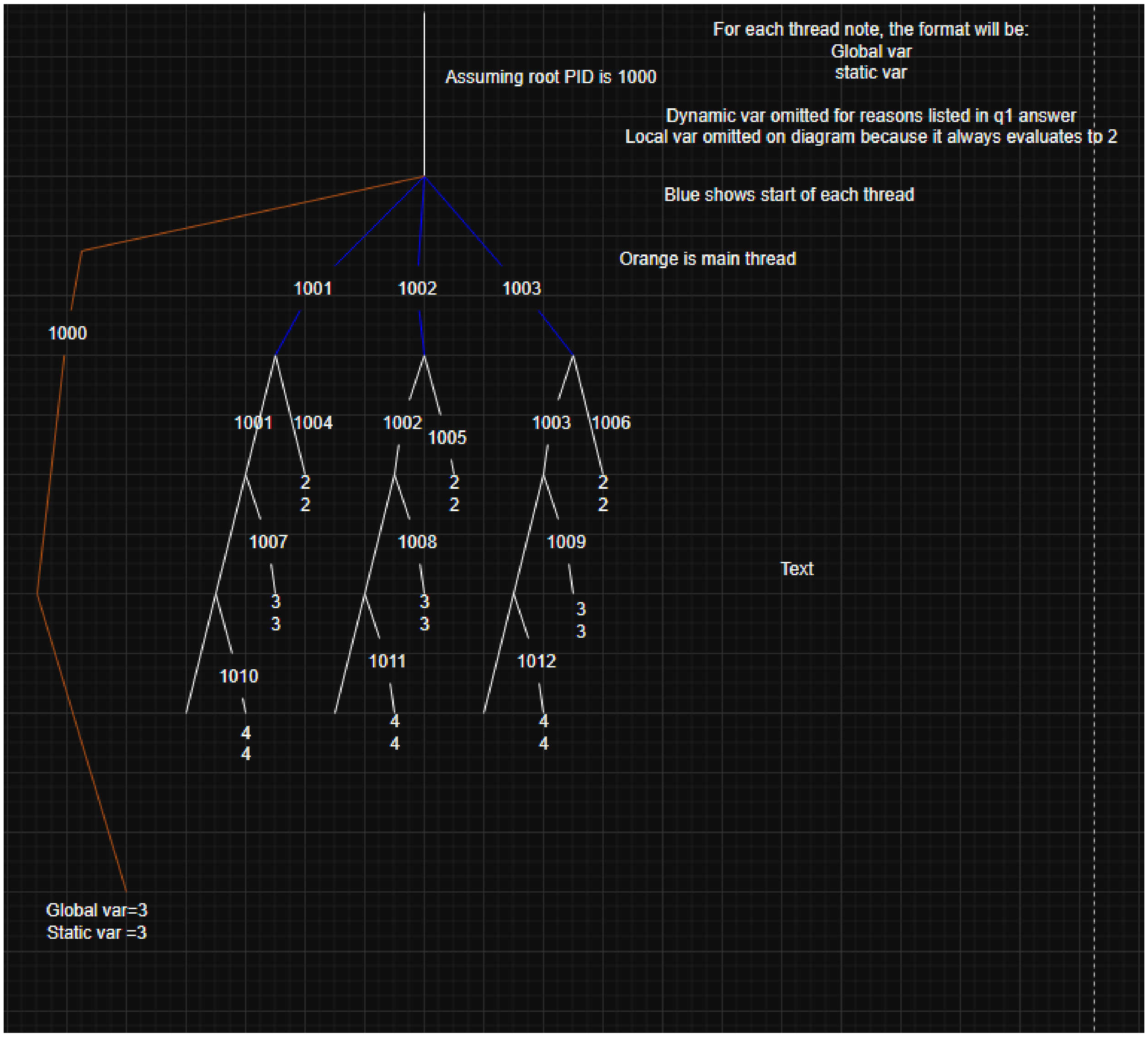
```
printing from a thread. dynamic_var=-320585728. global var=1 , Static var=1 , Local var=1
printing from a thread. dynamic_var=-320618496. global var=2 , Static var=2 , Local var=1
printing from a thread. dynamic_var=-320618496. global var=3 , Static var=3 , Local var=1
printing from a thread. dynamic_var=-320585728. global var=1 , Static var=1 , Local var=1
printing from a thread. dynamic_var=-320618496. global var=2 , Static var=2 , Local var=1
printing from a thread. dynamic_var=-320618496. global var=3 , Static var=3 , Local var=1
printing from a thread. dynamic_var=-320618496. global var=1 , Static var=1 , Local var=1
printing from a thread. dynamic_var=-320585728. global var=2 , Static var=2 , Local var=1
printing from a thread. dynamic_var=-320602112. global var=3 , Static var=3 , Local var=1
final value of global var =0 ,  Final value of static var= 0
robhill@csx1:~/OS/HW3$
```

2. For all outputs, Local var evaluated to 2, because the main thread sets it to one, and each fork takes a copy of that one and increments it
   Dynamic var is garbage data because the memory address is freed before it's read



```
printing from child inside a thread. dynamic_var=-1438220287. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1438220287. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1438220287. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1438253055. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1438253055. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1438253055. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1438236671. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1438236671. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1438236671. global var=4 , Static var=4 , Local var=2
printing from end of main. global var=3 , Static var=3
robhill@csx1:~/OS/HW3$ ./a.out
printing from child inside a thread. dynamic_var=-2139111423. global var=2 , Static var=2 , Local var=2
printing from child inside a thread. dynamic_var=-2139111423. global var=2 , Static var=2 , Local var=2
printing from child inside a thread. dynamic_var=-2139111423. global var=2 , Static var=2 , Local var=2
printing from child inside a thread. dynamic_var=-2139176959. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-2139176959. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-2139176959. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-2139144191. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-2139144191. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-2139144191. global var=4 , Static var=4 , Local var=2
printing from end of main. global var=3 , Static var=3
robhill@csx1:~/OS/HW3$ ./a.out
printing from child inside a thread. dynamic_var=-1036648447. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1036648447. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1036648447. global var=3 , Static var=3 , Local var=2
printing from child inside a thread. dynamic_var=-1036582911. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1036582911. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1036582911. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1036615679. global var=4 , Static var=4 , Local var=2
printing from child inside a thread. dynamic_var=-1036615679. global var=4 , Static var=4 , Local var=2
```

Global and static var evaluate to 3 on the main thread because each thread increments them by 1 before going into the loop with forks, and the further value changes aren't reflected in the main thread.
Within each loop, global and static var increment together in the body of the loop and are carried through the loops and forks

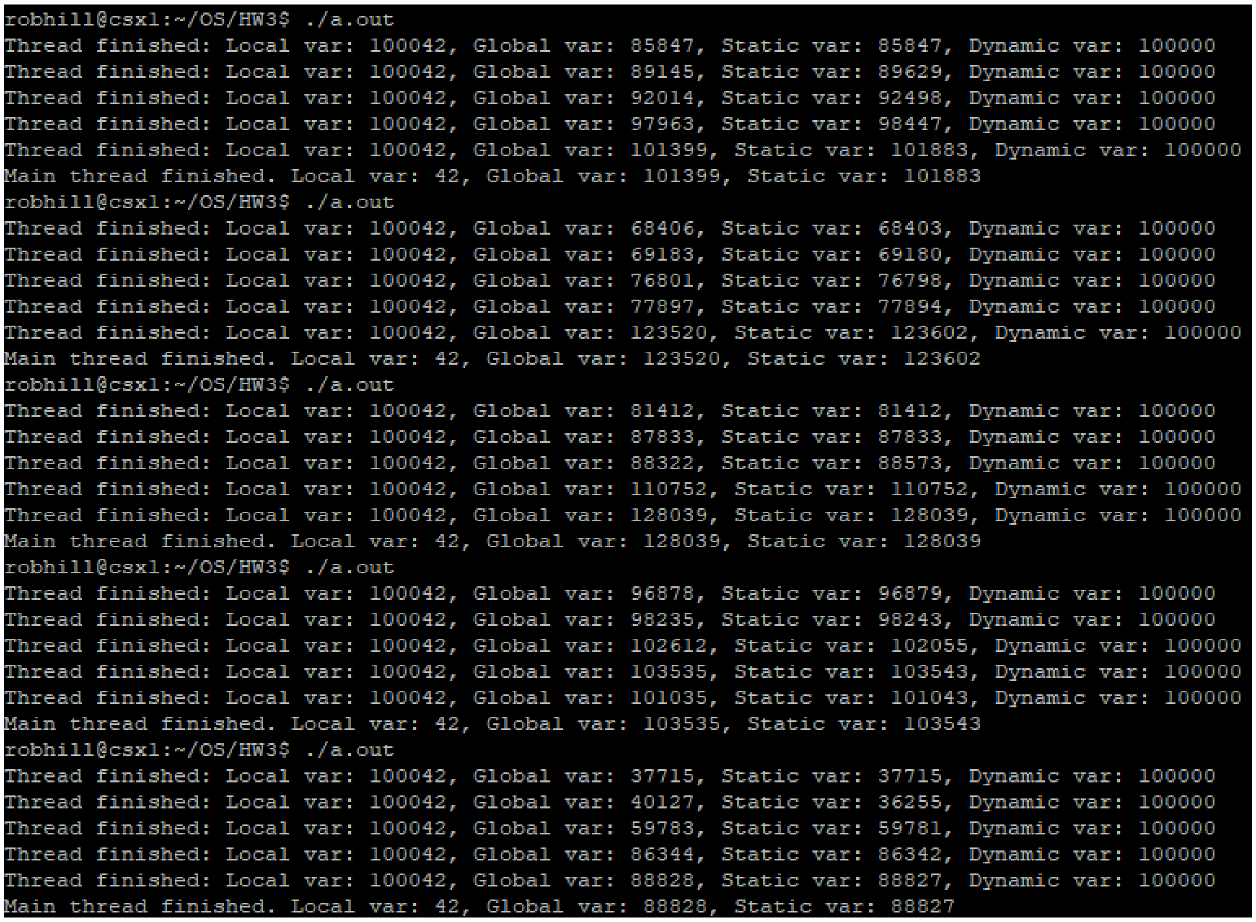

For each thread note, the format will be:
Global var
static var
Assuming root PID is 1000
Dynamic var omitted for reasons listed in q1 answer
Local var omitted on diagram because it always evaluates to 2
Blue shows start of each thread
Orange is main thread
1000
1001
1002
1003
1001
1004
1002
1005
1003
1006
2 2
2 2
2 2
1007
1008
1009
Text
3 3
3 3
3 3
1010
1011
1012
4 4
4 4
4 4
Global var=3
Static var =3

2b: If you wanted each thread to have it's own copy, you would need to use thread local storage, declaring the static_var as a __thread int. This would change the final output by making static var in the main thread =0 because each thread would have its own copy of it, and the main thread would not see any of the changes in the other threads.

3. Outputs: Thread 1 and 2 can be created in different orders. And the thread 2 done message never comes
The messages for main process done, child executing and parent continuing can also come in any order
Also because only 1 thread is created, it can get overwritten when the main program wants to open another thread program.
The parent and child process can also finish in different orders, and both parent and child try to join the thread, but depending on the implementation of fork, the child may not have a thread to join.

The issues I saw were that thread 1 and 2 weren't made from separate threads, so I fixed that. The parent process also didn't wait for the forked off child, which could create a zombie.
Both parent and child also attempted to close the threads.

4. Here are a few runs of the code for example.



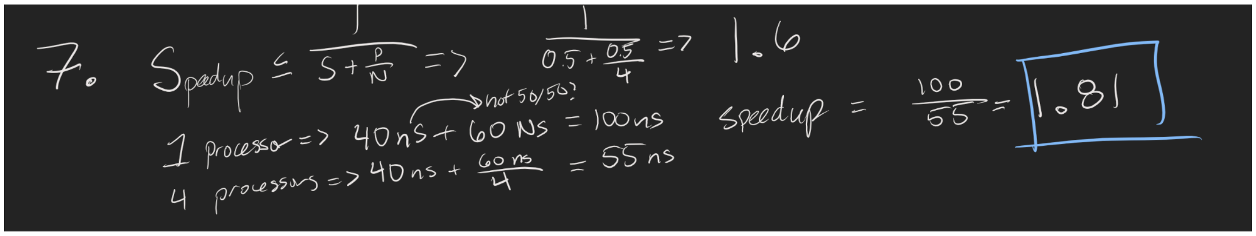
```
robhill@csx1:~/OS/HW3$ ./a.out
Thread finished: Local var: 100042, Global var: 85847, Static var: 85847, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 89145, Static var: 89629, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 92014, Static var: 92498, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 97963, Static var: 98447, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 101399, Static var: 101883, Dynamic var: 100000
Main thread finished. Local var: 42, Global var: 101399, Static var: 101883
robhill@csx1:~/OS/HW3$ ./a.out
Thread finished: Local var: 100042, Global var: 68406, Static var: 68403, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 69183, Static var: 69180, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 76801, Static var: 76798, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 77897, Static var: 77894, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 123520, Static var: 123602, Dynamic var: 100000
Main thread finished. Local var: 42, Global var: 123520, Static var: 123602
robhill@csx1:~/OS/HW3$ ./a.out
Thread finished: Local var: 100042, Global var: 81412, Static var: 81412, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 87833, Static var: 87833, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 88322, Static var: 88573, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 110752, Static var: 110752, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 128039, Static var: 128039, Dynamic var: 100000
Main thread finished. Local var: 42, Global var: 128039, Static var: 128039
robhill@csx1:~/OS/HW3$ ./a.out
Thread finished: Local var: 100042, Global var: 96878, Static var: 96879, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 98235, Static var: 98243, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 102612, Static var: 102055, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 103535, Static var: 103543, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 101035, Static var: 101043, Dynamic var: 100000
Main thread finished. Local var: 42, Global var: 103535, Static var: 103543
robhill@csx1:~/OS/HW3$ ./a.out
Thread finished: Local var: 100042, Global var: 37715, Static var: 37715, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 40127, Static var: 36255, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 59783, Static var: 59781, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 86344, Static var: 86342, Dynamic var: 100000
Thread finished: Local var: 100042, Global var: 88828, Static var: 88827, Dynamic var: 100000
Main thread finished. Local var: 42, Global var: 88828, Static var: 88827
```

Local var: All threads get a separate copy of local var passed in, so each increments it independently. Going from the starting value to 100,042
Global and static var: neither uses thread local storage, so each thread grabs the real variable when incrementing it, meaning due to timing issues and memory access, each incrementation can mess with and overwrite each other thread. This essentially randomizes their values from anywhere from the starting value all the way up to 500,000 if each memory access went perfectly and didn't collide
Dynamic var: it gets created by each thread independently
5. If both threads execute completely sequentially, the program should print 1 and then 2. If the memory accesses for incrementation end up happening at or near the same time, both threads could end up printing 1.
6. The output of this program depends on the orders that the threads access memory and execute.
If thread 1 executes fully before thread 2, it would print

```
Thread1 (before fun1_1 call): Count value is = 11
Thread1 (inside fun1_1: Count value is = 12
Thread1 (after fun1_1 call): Count value is = 12
Thread2: Count value is = 13
```

Though, depending on when thread 2’s function access count to increment it, it could end up incrementing count earlier than expected, or overwriting one of the increments because 2 threads might access count at the same time, resulting in only one incrementation going through.

7.



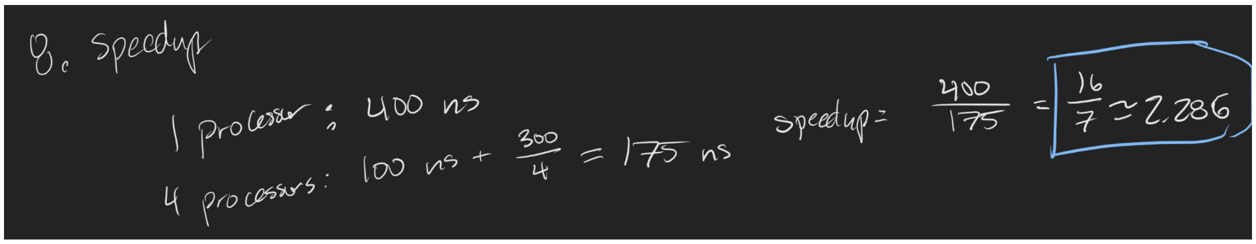
7. $\text{Speedup} \le \frac{1}{S + \frac{P}{N}} \Rightarrow \frac{1}{0.5 + \frac{0.5}{4}} \Rightarrow 1.6$

1 processor $\Rightarrow$ 40 ns + 60 ns = 100 ns $\rightarrow$ not 50/50?

4 processors $\Rightarrow$ 40 ns + $\frac{60 \text{ ns}}{4}$ = 55 ns

$\text{speedup} = \frac{100}{55} = \boxed{1.81}$

8.



8. Speedup

1 processor: 400 ns

4 processors: 100 ns + $\frac{300}{4}$ = 175 ns

$\text{speedup} = \frac{400}{175} = \boxed{\frac{16}{7} \simeq 2.286}$