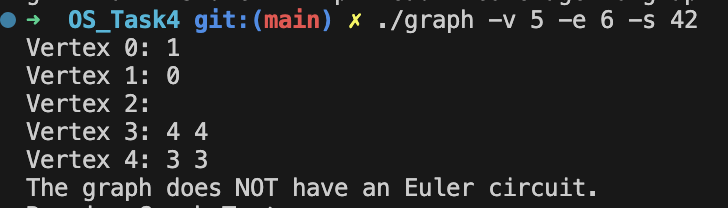
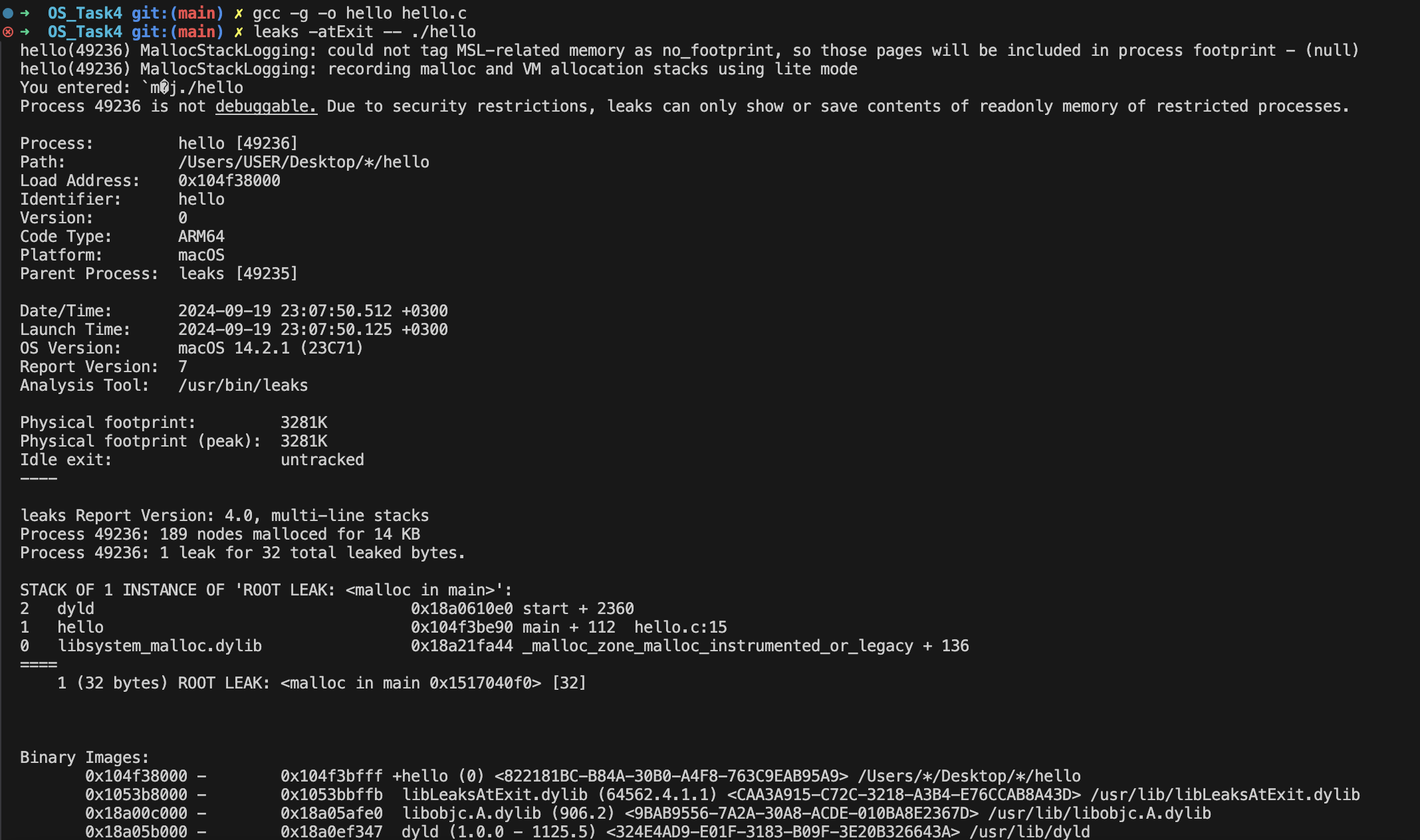
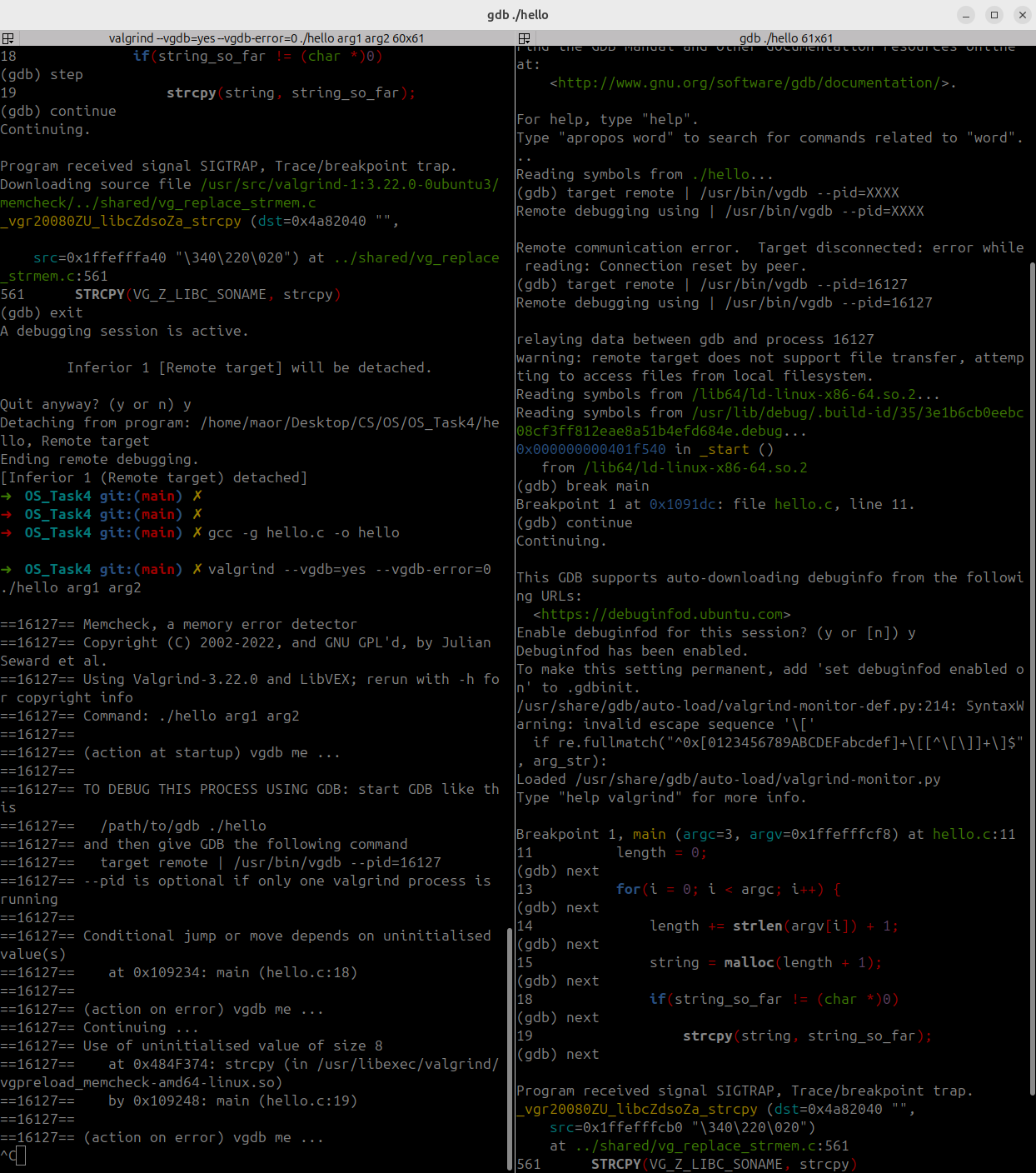
Exercise 4 OS course Ariel:  
  
  
Task3:

This is a random graph generated using argc,argv with getopt(3) which is implemented in the main file.

Task 5 - Valgrind check.  
  
Note: I used Leaks here as I am using MAC and didn’t find a way to run valgrind here.  
  
  
  
We can see in the screenshot above that the leak source is in the “malloc” and we can identify in the code that we are not freeing the memory after we are done using it.  
The code allocates memory to string in each iteration of the loop without freeing the previously allocated memory, leading to a memory leak.

Task 7:  
  
I used “ThreadSanitizer” (TSan) Instead of Helgrind as Helgrind is not supported on my MAC.   
  
  
  
ThreadSanitizer flagged a data race in the following line of the code:  
accum += x \* x;

Task 6 Valgrind + GDB debugger:  
  


As seen in the terminals above I ran valgrind on the left while connecting through the PID on the right while in GDB with  
“target remote | /usr/bin/vgdb --pid=16127”  
and connected successfuly to debugging  
 and I created a breaking point in main with “break main”  
after multiple “next” we hit the issue with the valgrind terminal as seen in the picture.  
  
commands on left terminal:  
gcc -g hello.c -o hello  
algrind --vgdb=yes --vgdb-error=0 ./hello arg1 arg2

Commands on right terminal:

gdb ./hello

target remote | /usr/bin/vgdb --pid=16127  
  
bereak main

Continue

Next next next

multiple threads are accessing and modifying “accum” concurrently.  
without any sync leading to the race condition.  
  
  
Terminal footage:  
