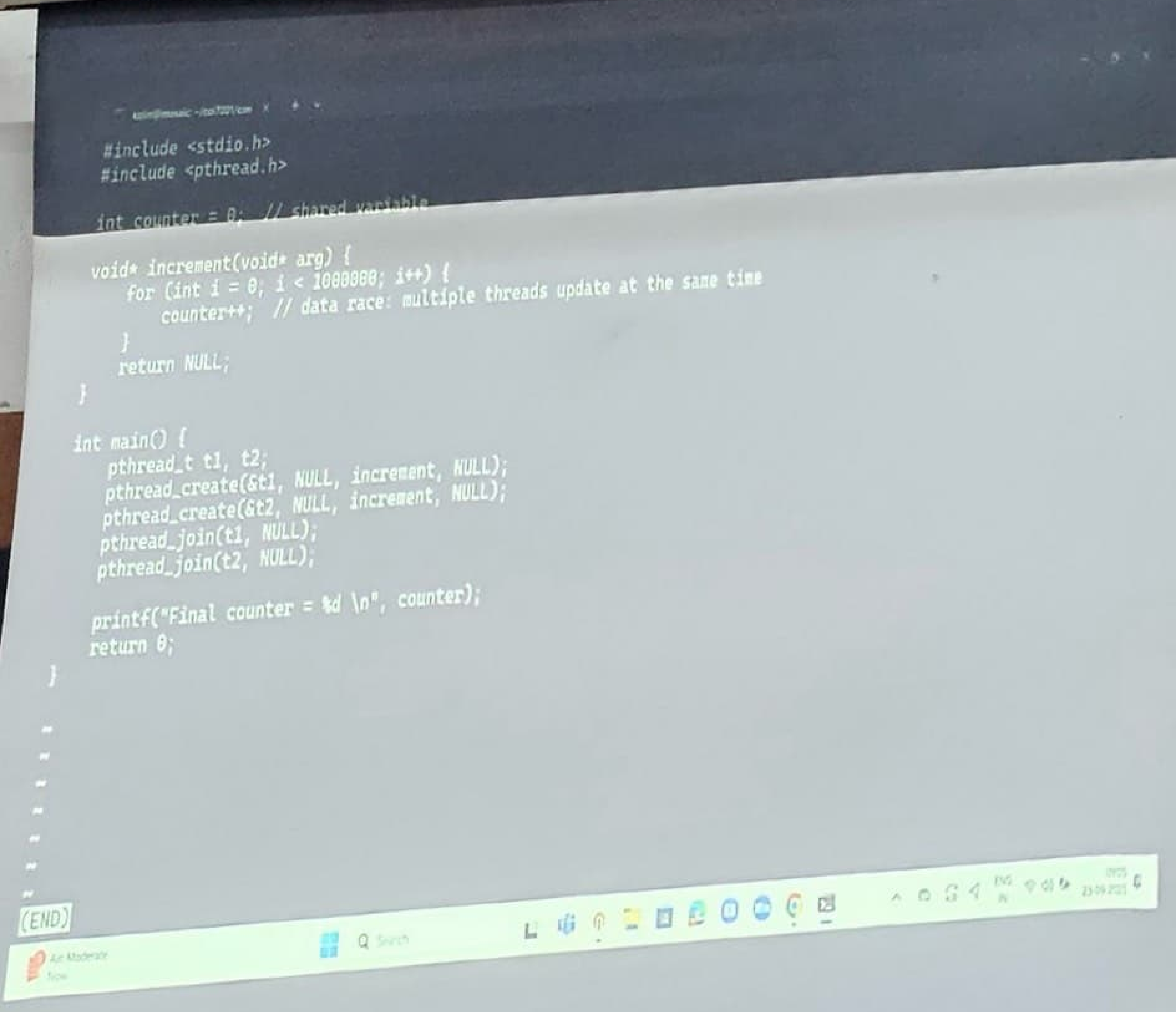
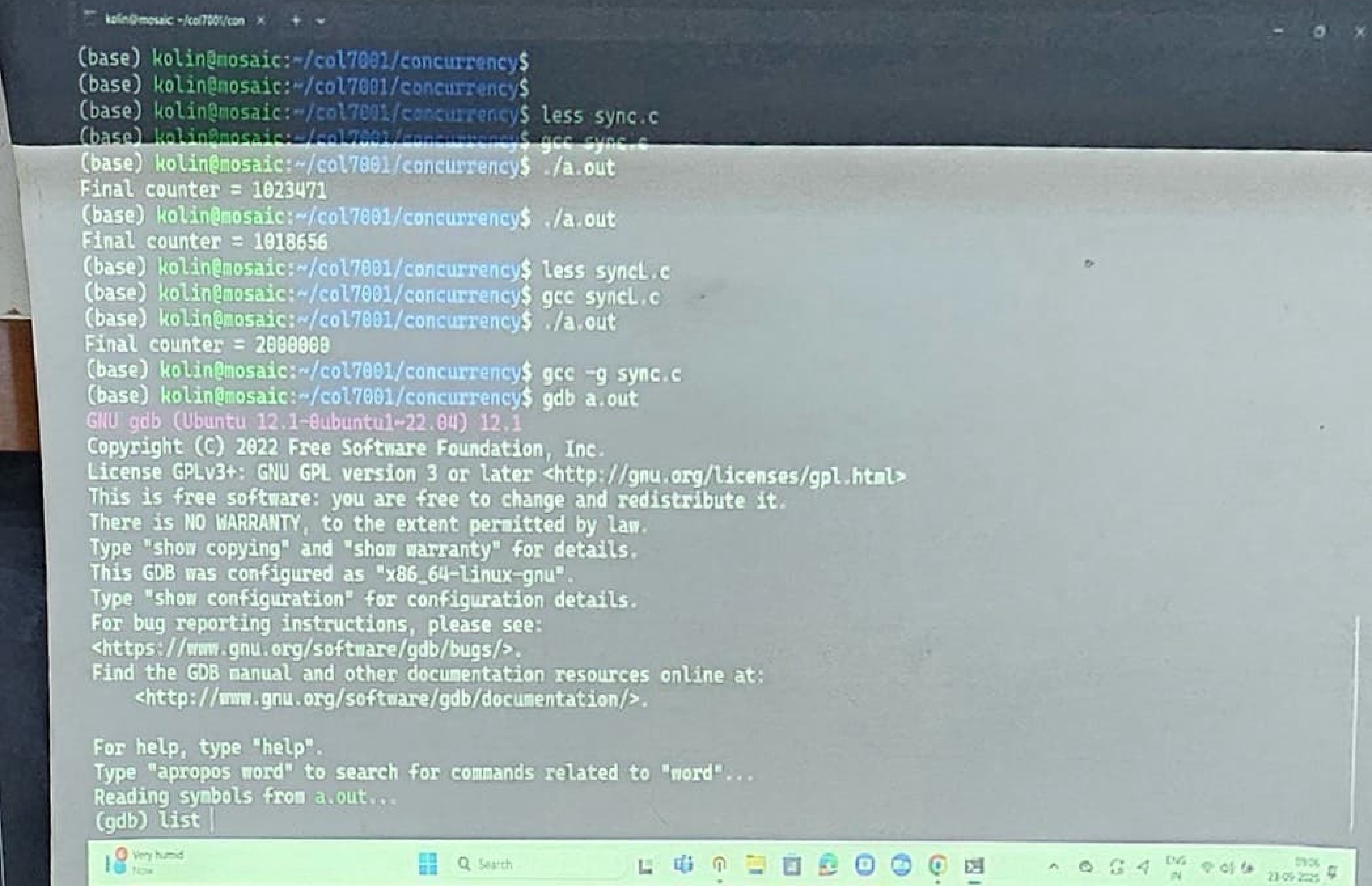
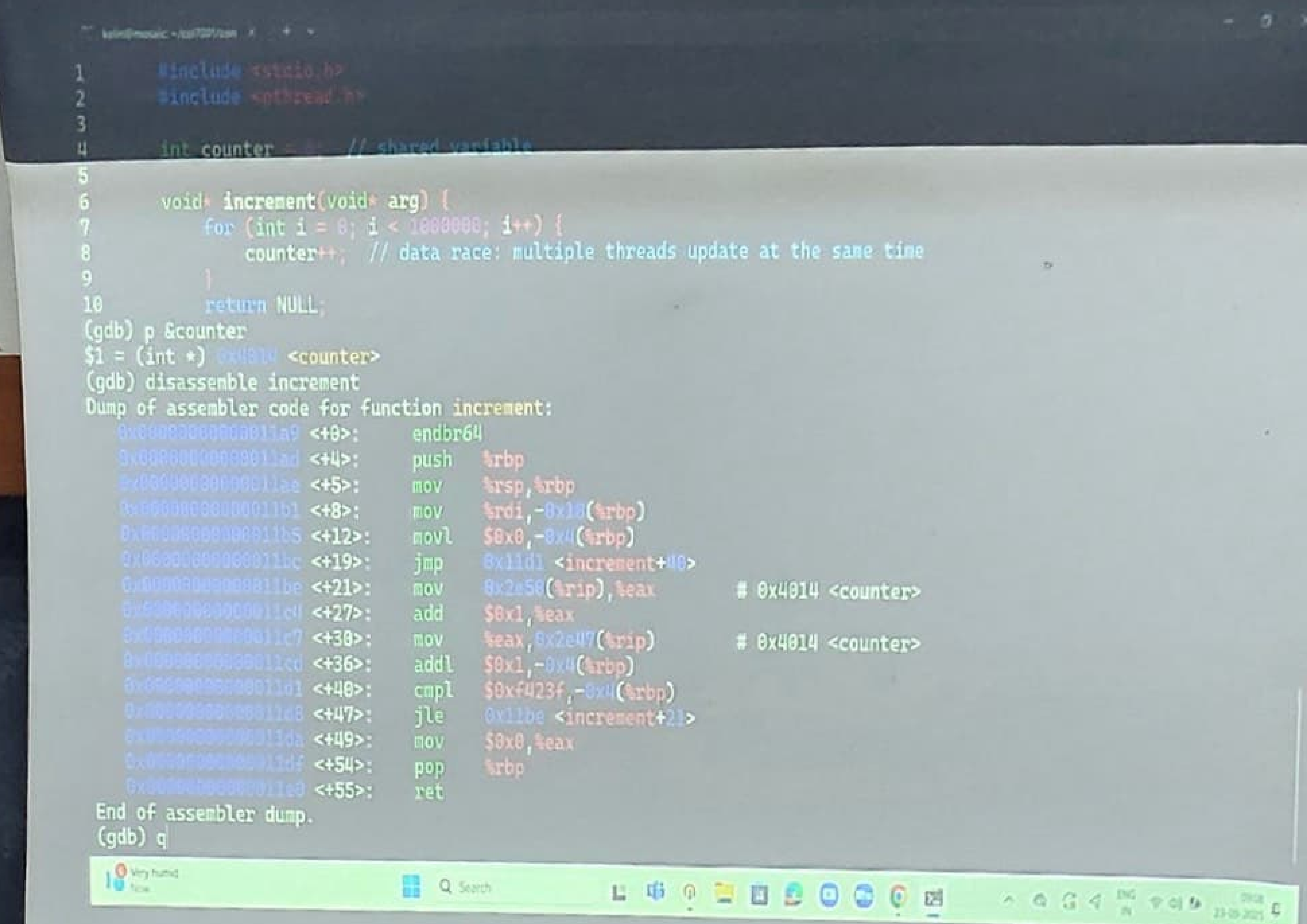
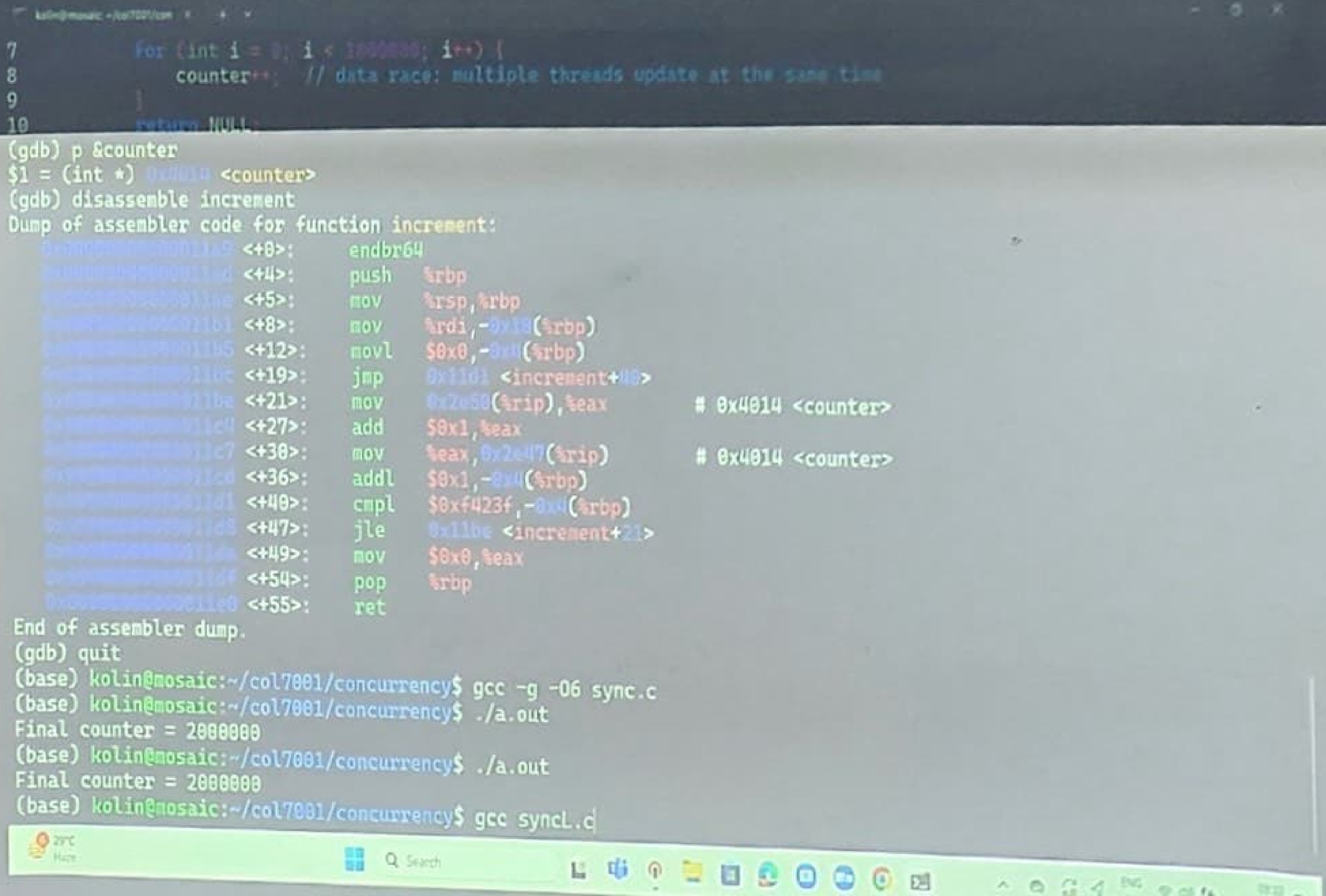




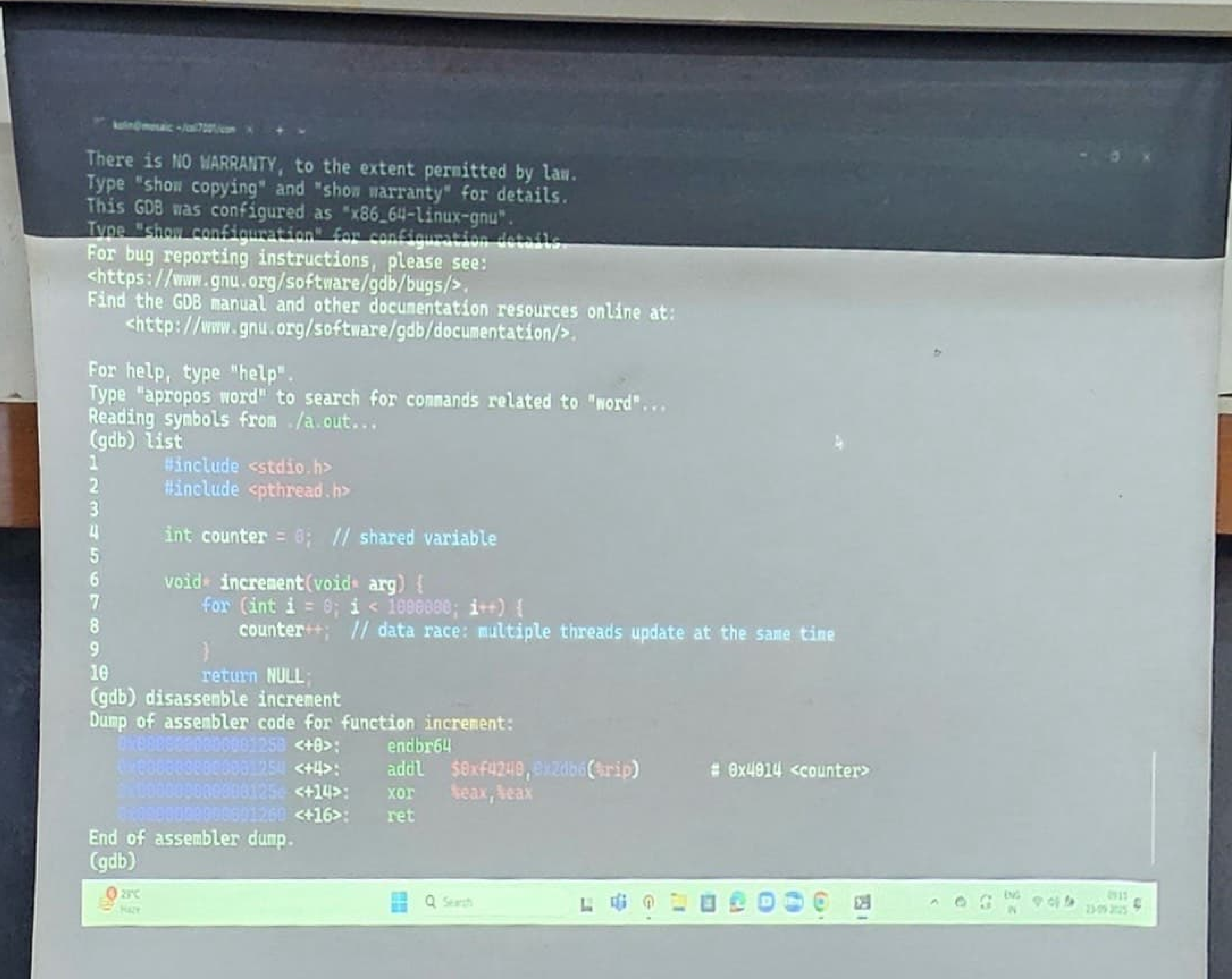
```
(base) kolin@mosaic:~/col7001/concurrency$
(base) kolin@mosaic:~/col7001/concurrency$
(base) kolin@mosaic:~/col7001/concurrency$ less sync.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc sync.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Final counter = 1023471
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Final counter = 1018656
(base) kolin@mosaic:~/col7001/concurrency$ less syncL.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc syncL.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Final counter = 2000000
(base) kolin@mosaic:~/col7001/concurrency$ gcc -g sync.c
(base) kolin@mosaic:~/col7001/concurrency$ gdb a.out
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) list
```



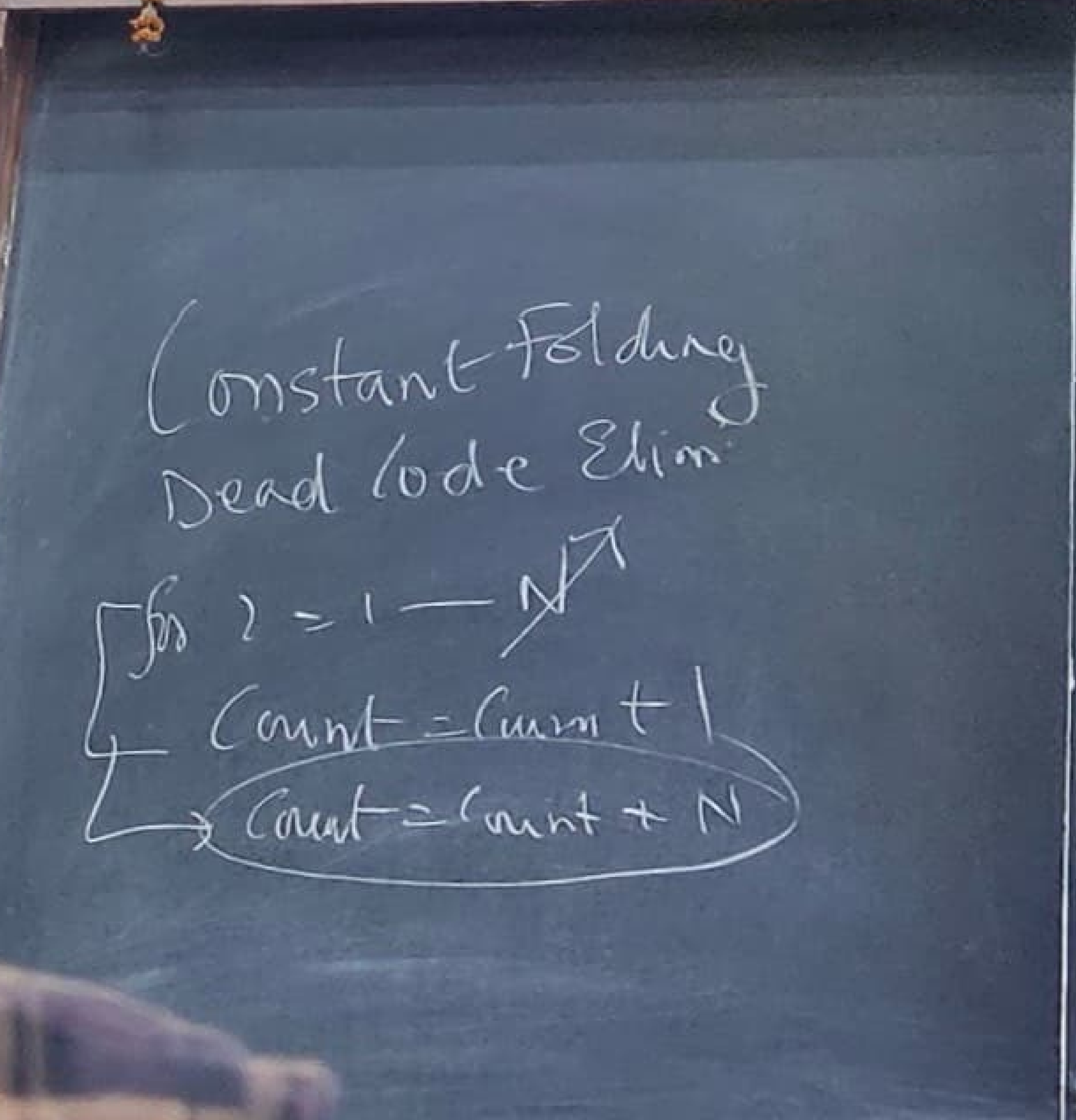
```
1       #include <stdio.h>
2       #include <pthread.h>
3
4       int counter = 0;    // shared variable
5
6       void* increment(void* arg) {
7           for (int i = 0; i < 1000000; i++) {
8               counter++;  // data race: multiple threads update at the same time
9           }
10          return NULL;
(gdb) p &counter
$1 = (int *) 0x4014 <counter>
(gdb) disassemble increment
Dump of assembler code for function increment:
   0x00000000000011a9 <+0>:     endbr64
   0x00000000000011ad <+4>:     push   %rbp
   0x00000000000011ae <+5>:     mov    %rsp,%rbp
   0x00000000000011b1 <+8>:     mov    %rdi,-0x18(%rbp)
   0x00000000000011b5 <+12>:    movl   $0x0,-0x4(%rbp)
   0x00000000000011bc <+19>:    jmp    0x11d1 <increment+40>
   0x00000000000011be <+21>:    mov    0x2e50(%rip),%eax        # 0x4014 <counter>
   0x00000000000011c4 <+27>:    add    $0x1,%eax
   0x00000000000011c7 <+30>:    mov    %eax,0x2e47(%rip)        # 0x4014 <counter>
   0x00000000000011cd <+36>:    addl   $0x1,-0x4(%rbp)
   0x00000000000011d1 <+40>:    cmpl   $0xf423f,-0x4(%rbp)
   0x00000000000011d8 <+47>:    jle    0x11be <increment+21>
   0x00000000000011da <+49>:    mov    $0x0,%eax
   0x00000000000011df <+54>:    pop    %rbp
   0x00000000000011e0 <+55>:    ret
End of assembler dump.
(gdb) q
```



```
        for (int i = 0; i < 1000000; i++) {
            counter++;  // data race: multiple threads update at the same time
        }
        return NULL;
(gdb) p &counter
$1 = (int *) 0x4014 <counter>
(gdb) disassemble increment
Dump of assembler code for function increment:
   0x00000000000011a9 <+0>:     endbr64
   0x00000000000011ad <+4>:     push   %rbp
   0x00000000000011ae <+5>:     mov    %rsp,%rbp
   0x00000000000011b1 <+8>:     mov    %rdi,-0x18(%rbp)
   0x00000000000011b5 <+12>:    movl   $0x0,-0x4(%rbp)
   0x00000000000011bc <+19>:    jmp    0x11d1 <increment+40>
   0x00000000000011be <+21>:    mov    0x2e50(%rip),%eax        # 0x4014 <counter>
   0x00000000000011c4 <+27>:    add    $0x1,%eax
   0x00000000000011c7 <+30>:    mov    %eax,0x2e47(%rip)        # 0x4014 <counter>
   0x00000000000011cd <+36>:    addl   $0x1,-0x4(%rbp)
   0x00000000000011d1 <+40>:    cmpl   $0xf423f,-0x4(%rbp)
   0x00000000000011d8 <+47>:    jle    0x11be <increment+21>
   0x00000000000011da <+49>:    mov    $0x0,%eax
   0x00000000000011df <+54>:    pop    %rbp
   0x00000000000011e0 <+55>:    ret
End of assembler dump.
(gdb) quit
(base) kolin@mosaic:~/col7001/concurrency$ gcc -g -O6 sync.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Final counter = 2000000
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Final counter = 2000000
(base) kolin@mosaic:~/col7001/concurrency$ gcc syncL.c
```



```
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./a.out...
(gdb) list
1       #include <stdio.h>
2       #include <pthread.h>
3
4       int counter = 0;  // shared variable
5
6       void* increment(void* arg) {
7           for (int i = 0; i < 1000000; i++) {
8               counter++;  // data race: multiple threads update at the same time
9           }
10          return NULL;
(gdb) disassemble increment
Dump of assembler code for function increment:
   0x0000000000001250 <+0>:     endbr64
   0x0000000000001254 <+4>:     addl   $0xf4240,0x2db6(%rip)        # 0x4014 <counter>
   0x000000000000125e <+14>:    xor    %eax,%eax
   0x0000000000001260 <+16>:    ret
End of assembler dump.
(gdb)
```



Constant Folding

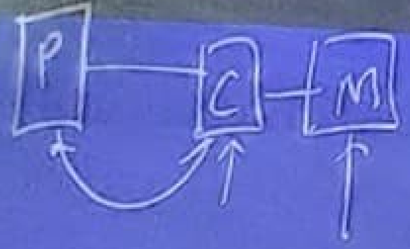
Dead Code Elim

for i = 1 — ~~N~~

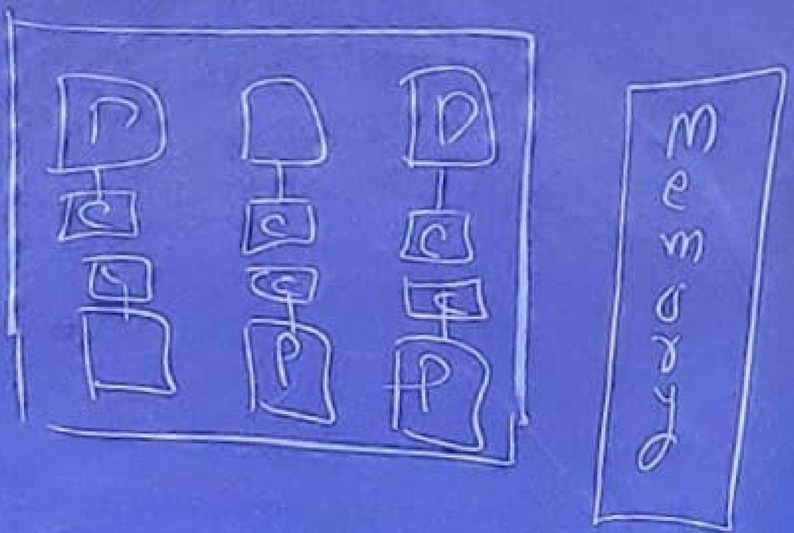
Count = Count + 1

Count = Count + N

CPU



multi Processors



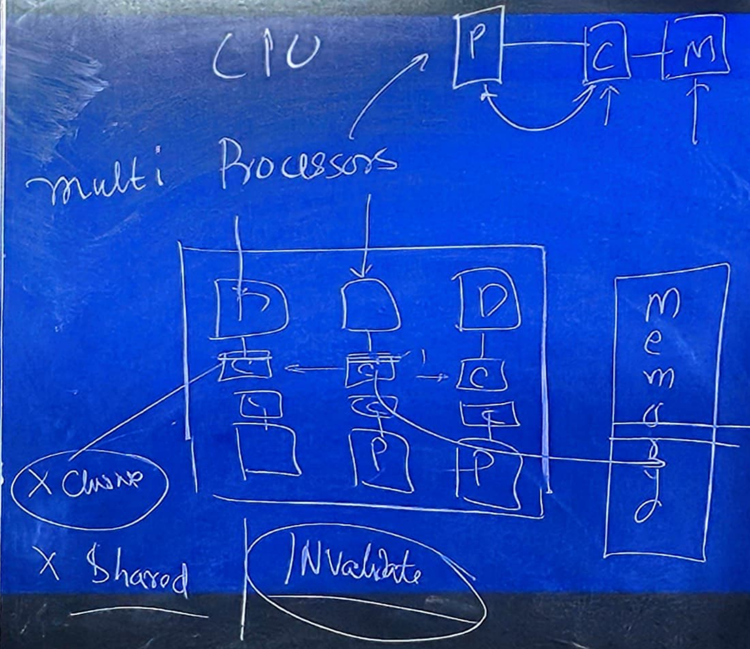
Memory Model

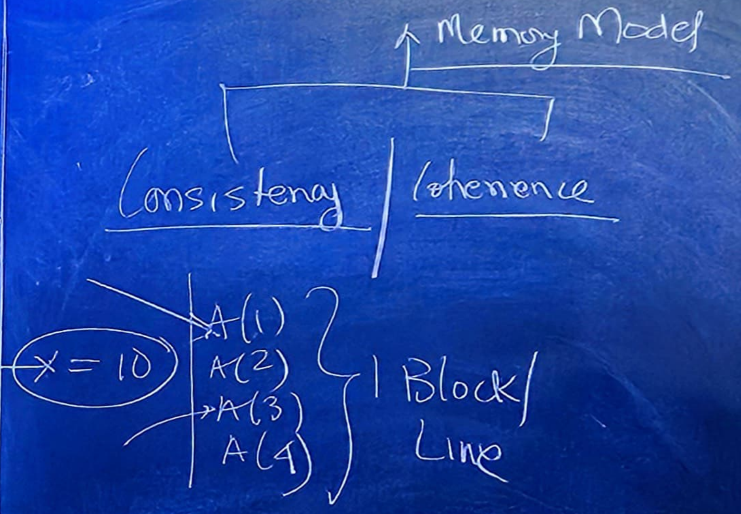
Consistency / Coherence

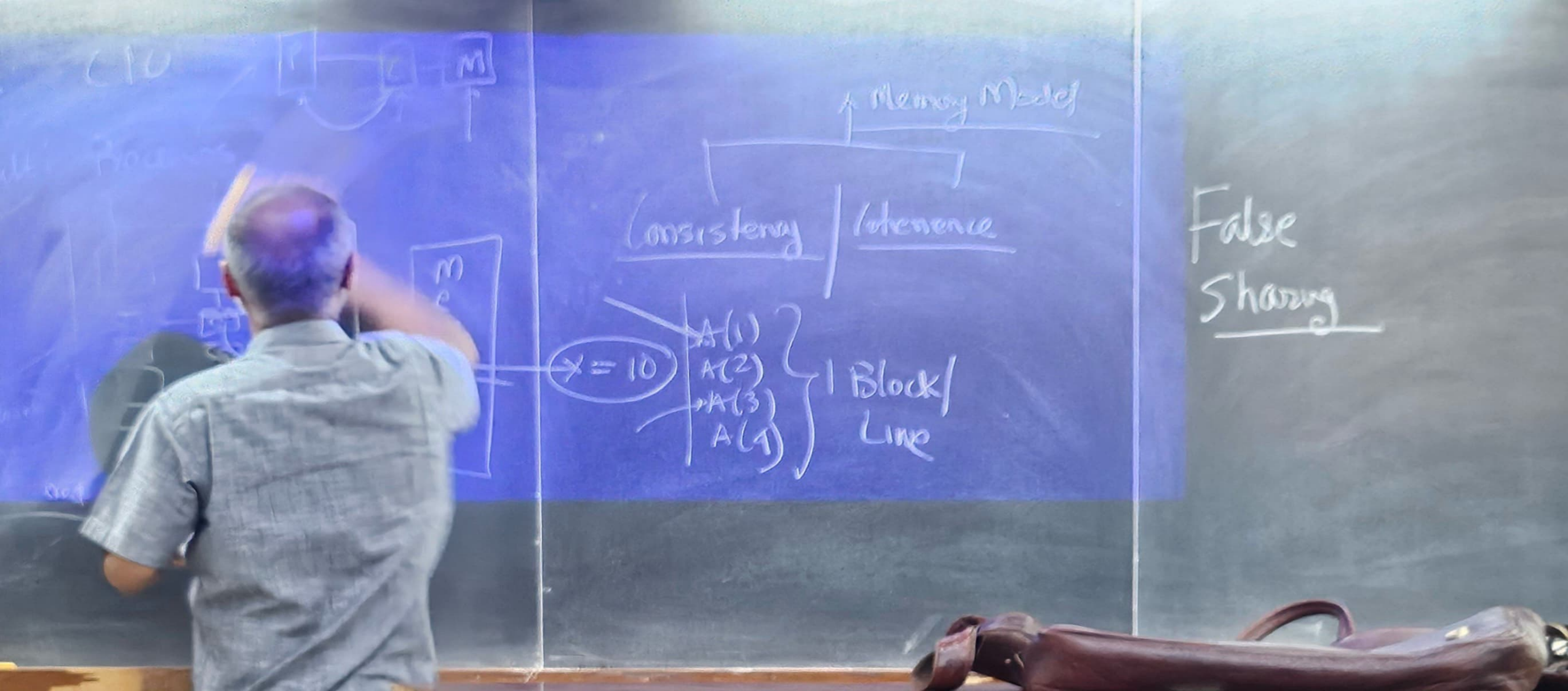
Constant Folding

Dead Code Elim

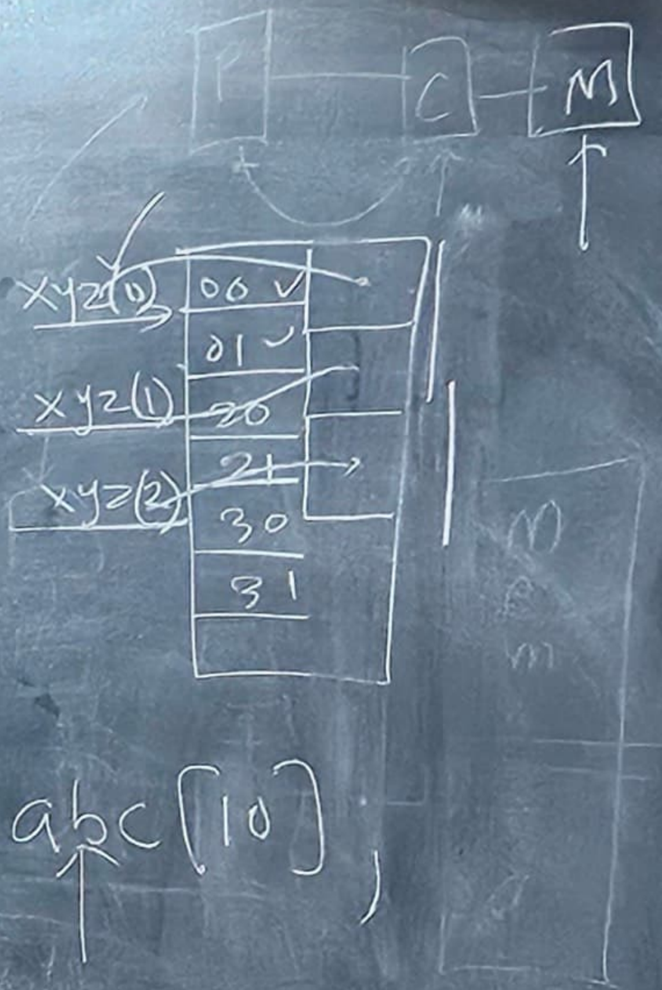
```
for i = 1 — ~~N~~
    Count = Count + 1
Count = Count + N
```






P
C
M
M
x = 10
Memory Model
Consistency
Coherence
A(1)
A(2)
A(3)
A(4)
1 Block/
Line
False
Sharing

CPU

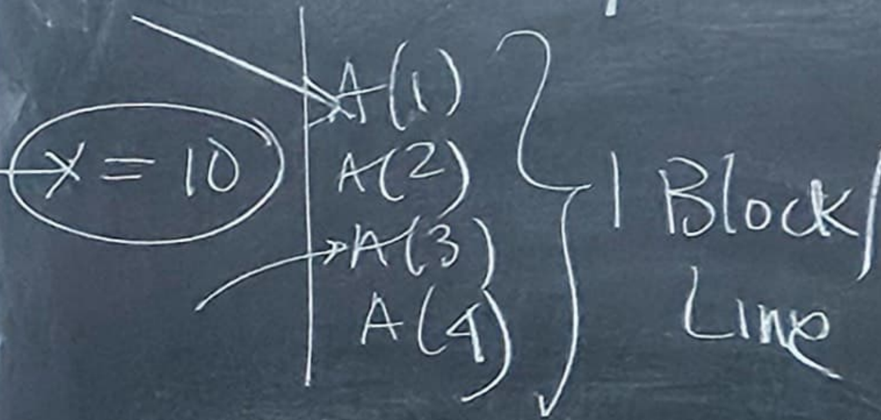


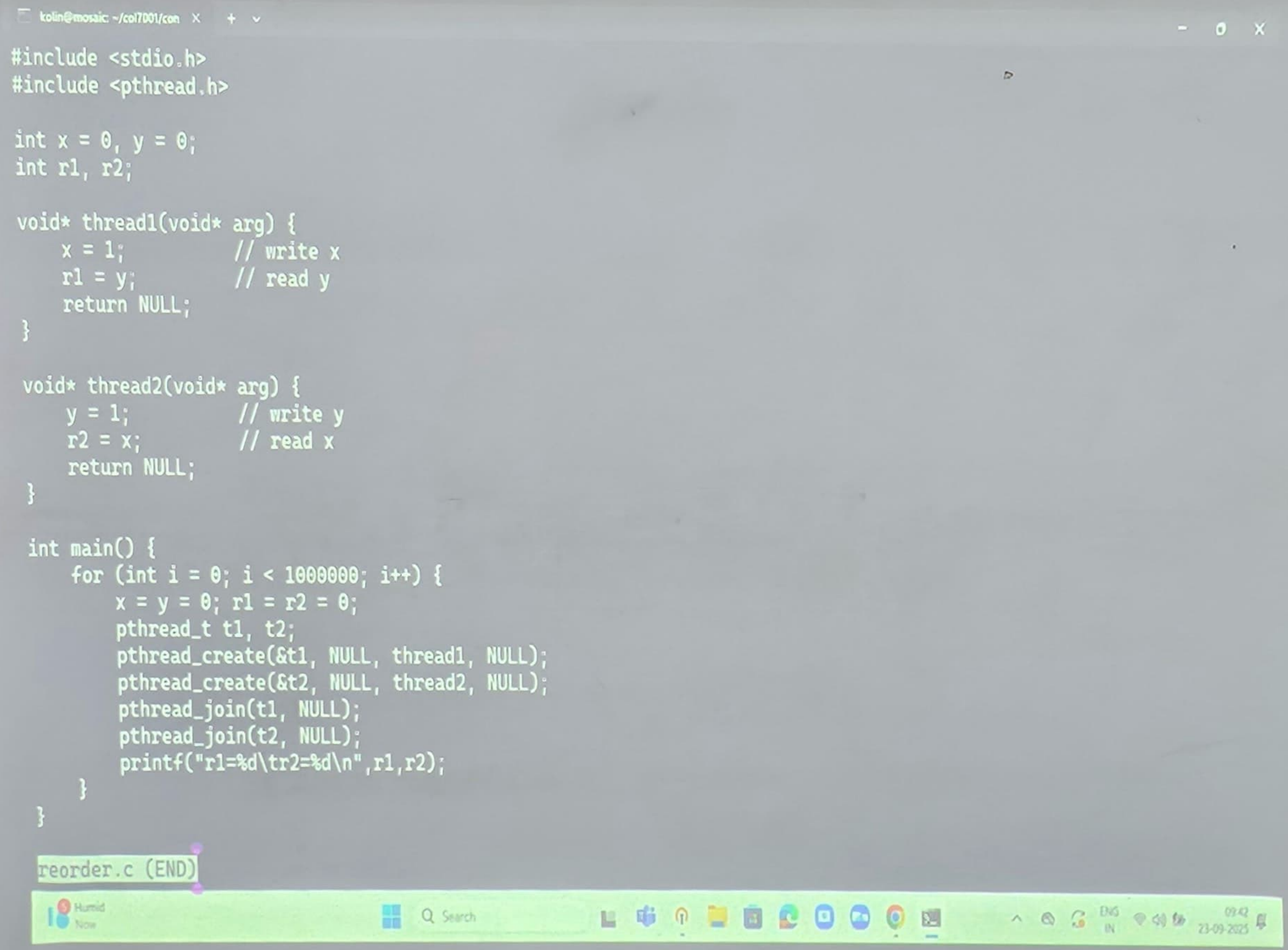
$\frac{5}{8}$

```
struct {
    int a
    char b
} xyz

xyz abc[10];
```

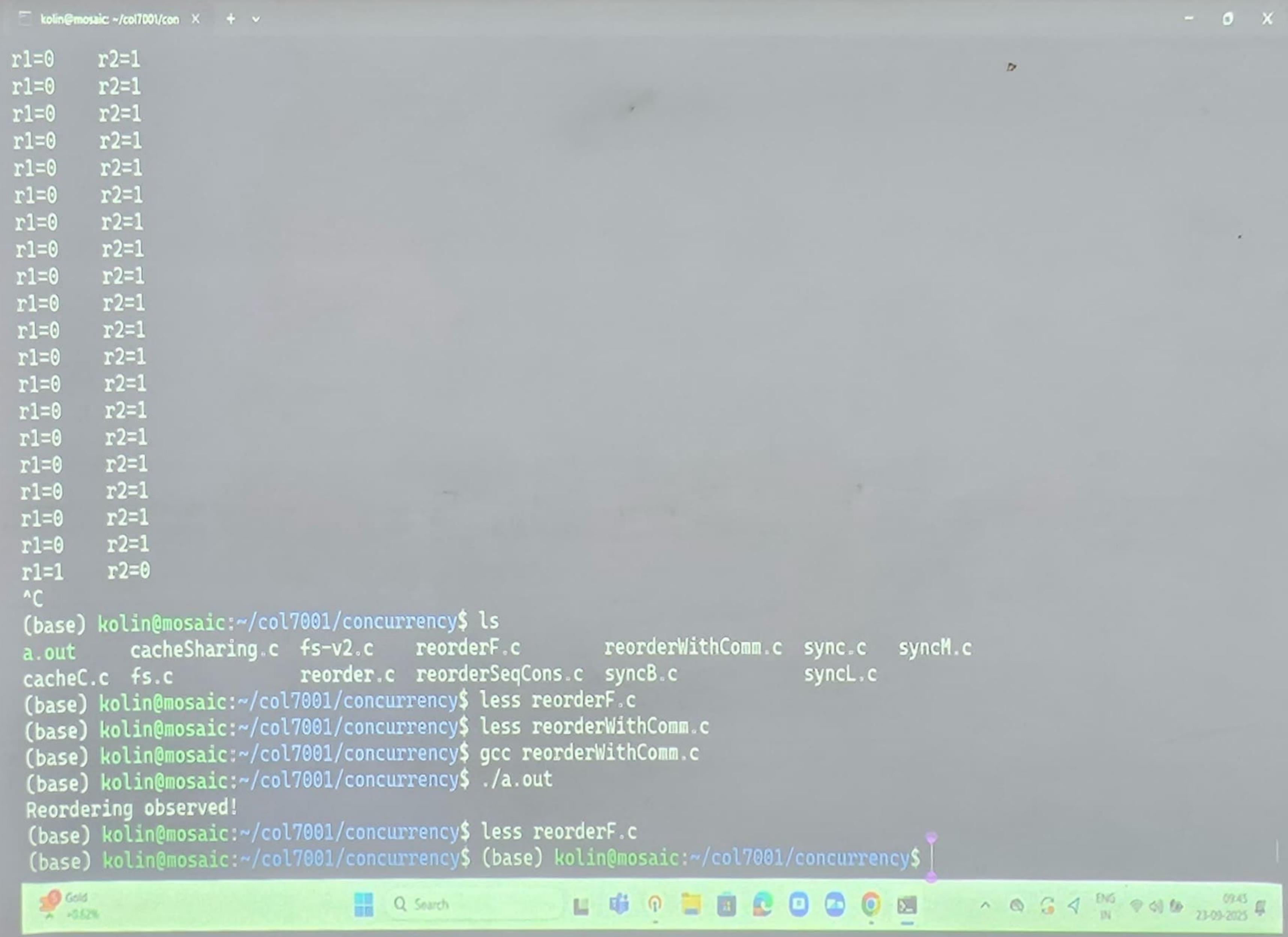
Memory M

Consistency | Coherence

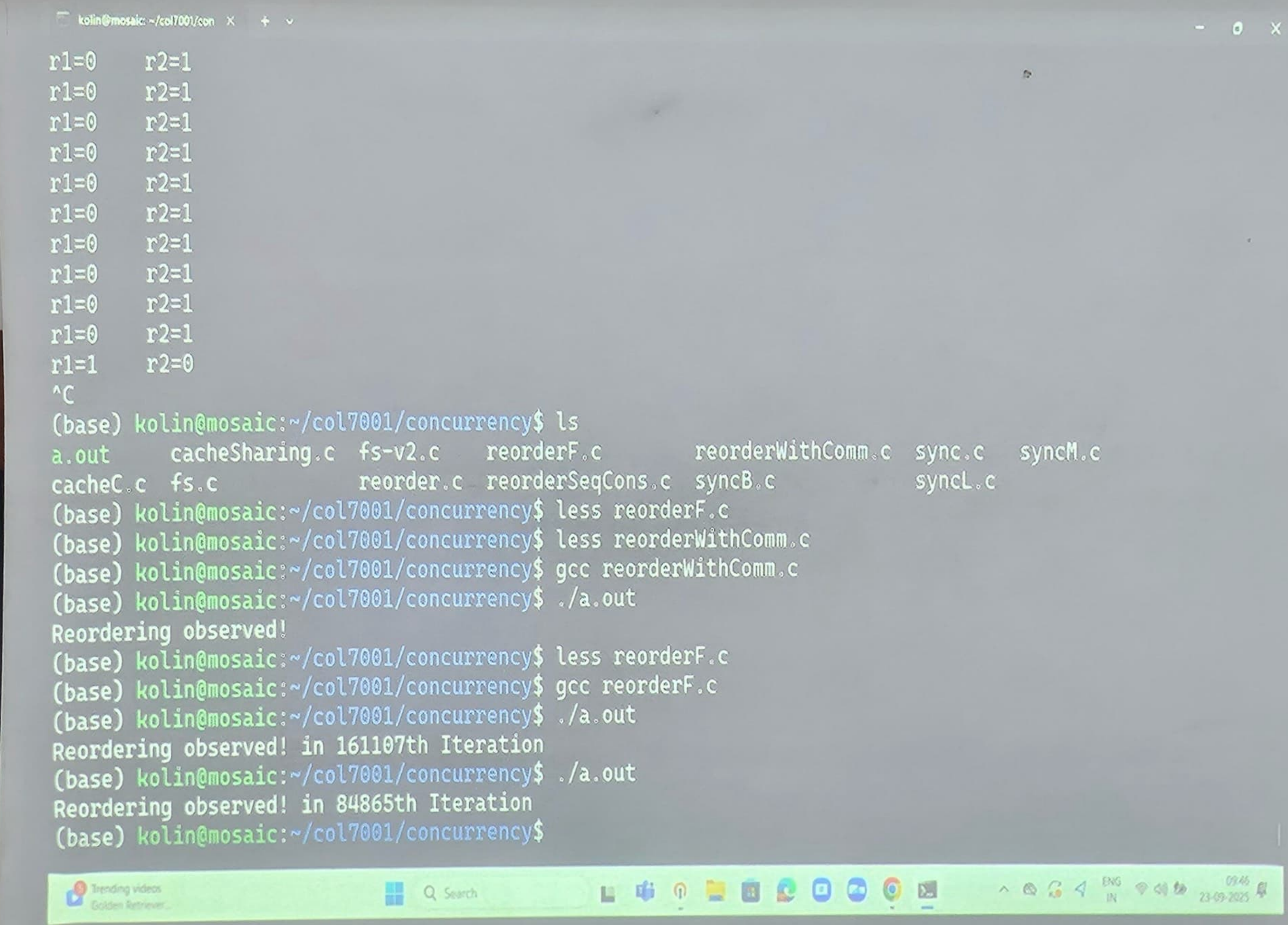




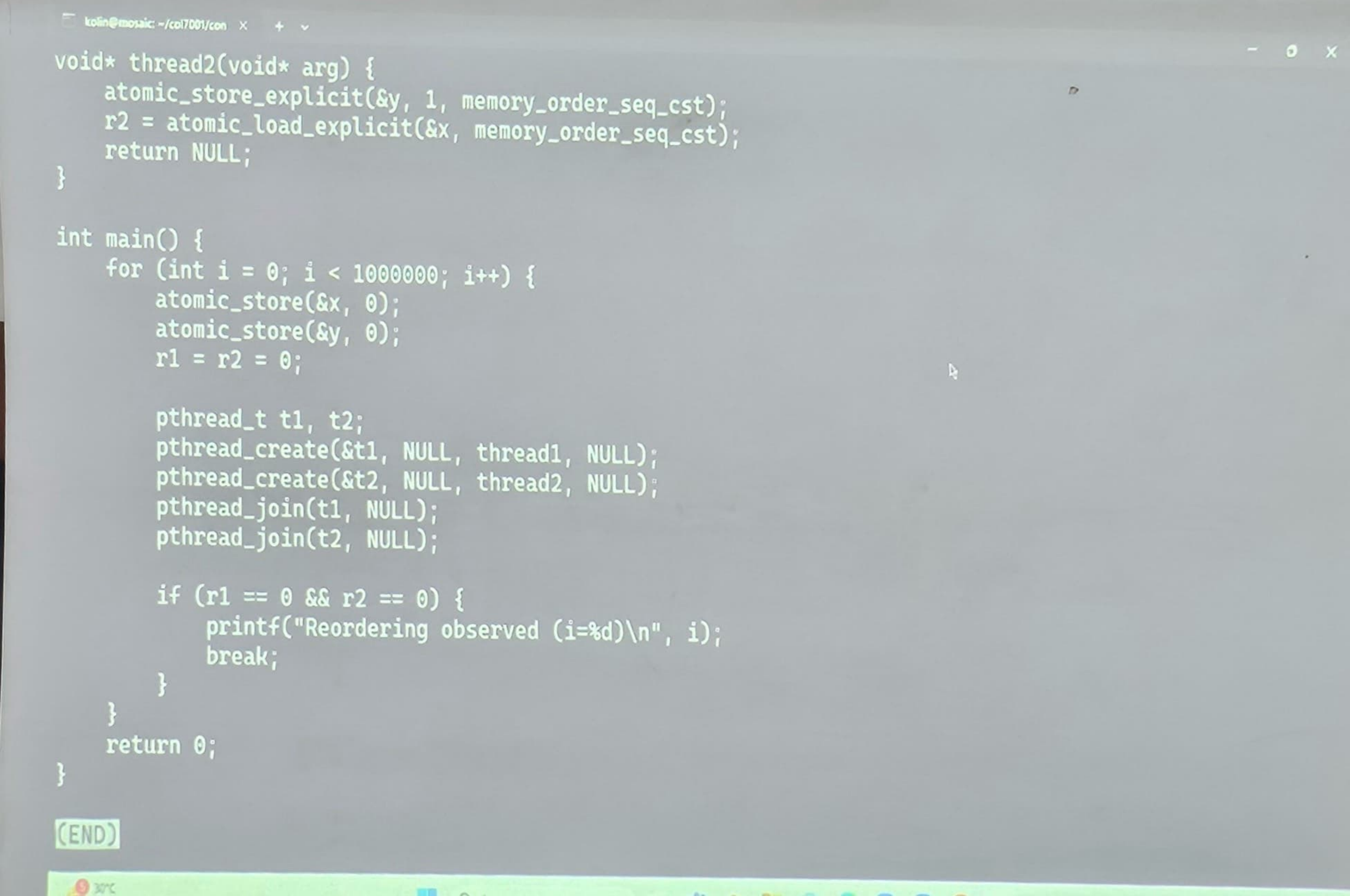
```c
#include <stdio.h>
#include <pthread.h>

int x = 0, y = 0;
int r1, r2;

void* thread1(void* arg) {
    x = 1;          // write x
    r1 = y;         // read y
    return NULL;
}

void* thread2(void* arg) {
    y = 1;          // write y
    r2 = x;         // read x
    return NULL;
}

int main() {
    for (int i = 0; i < 1000000; i++) {
        x = y = 0; r1 = r2 = 0;
        pthread_t t1, t2;
        pthread_create(&t1, NULL, thread1, NULL);
        pthread_create(&t2, NULL, thread2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        printf("r1=%d\tr2=%d\n",r1,r2);
    }
}

reorder.c (END)
```



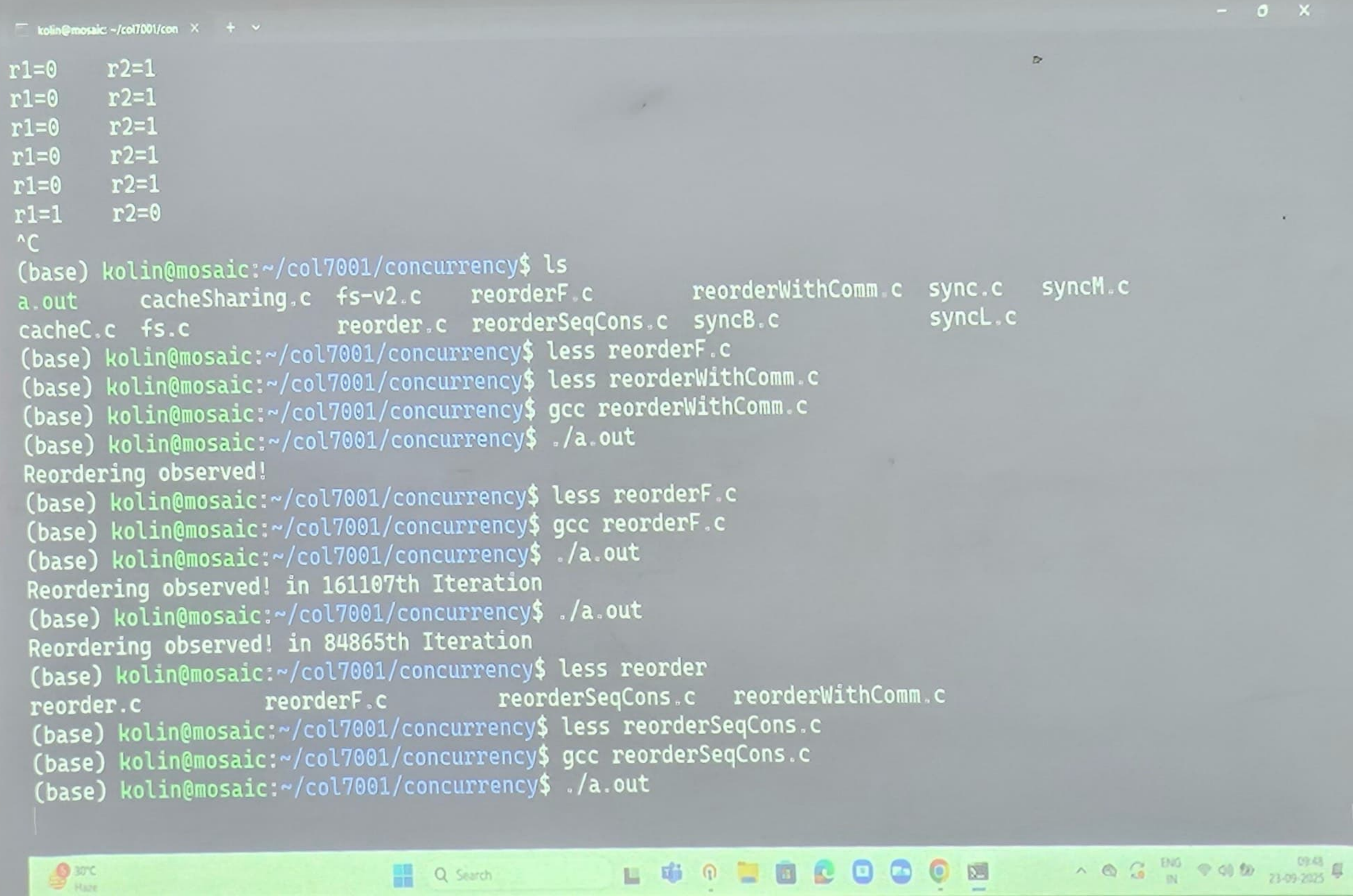
```
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=1    r2=0
^C
(base) kolin@mosaic:~/col7001/concurrency$ ls
a.out     cacheSharing.c  fs-v2.c    reorderF.c        reorderWithComm.c  sync.c   syncM.c
cacheC.c  fs.c            reorder.c  reorderSeqCons.c  syncB.c            syncL.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed!
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ (base) kolin@mosaic:~/col7001/concurrency$
```



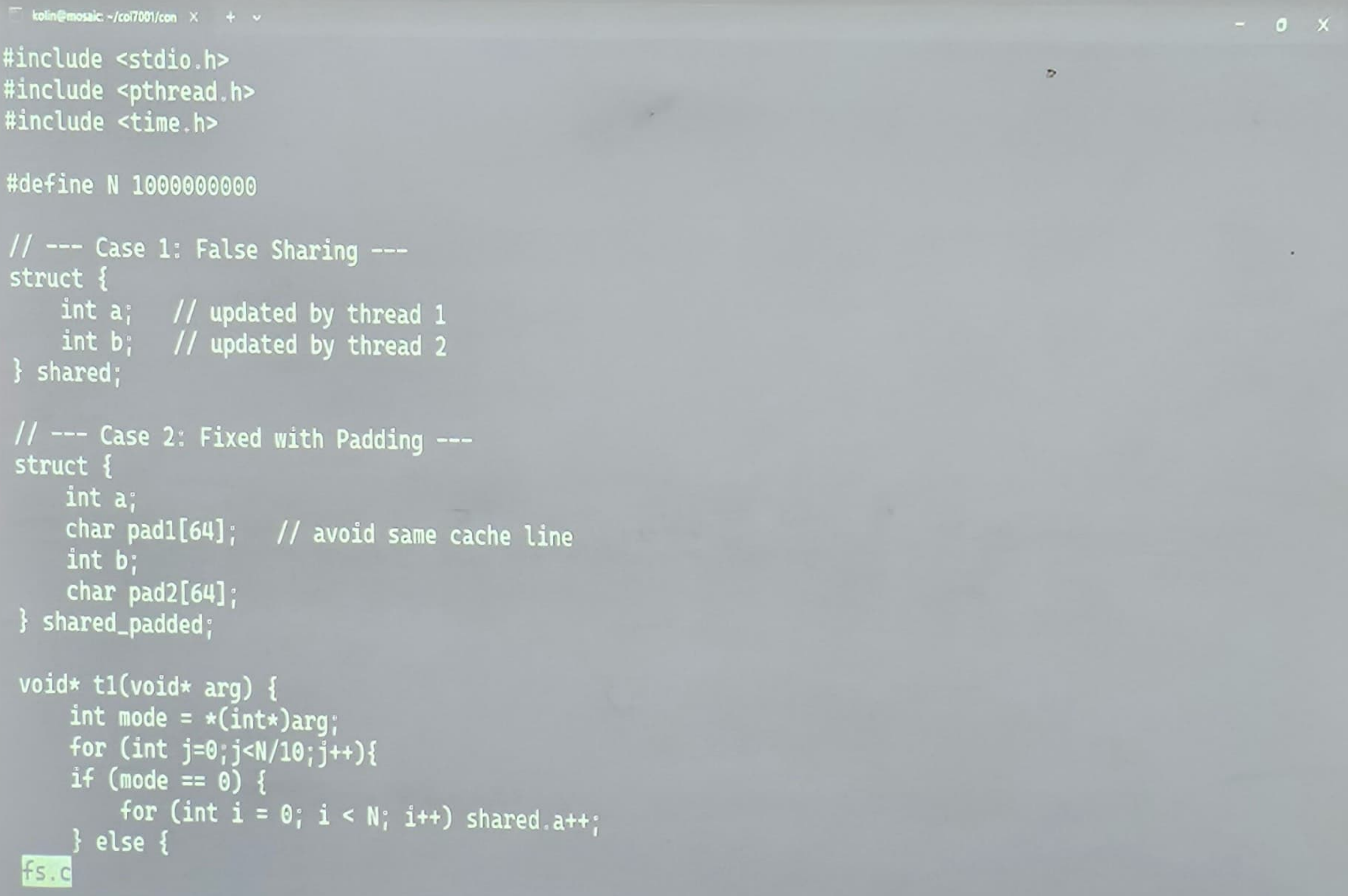
```
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=1    r2=0
^C
(base) kolin@mosaic:~/col7001/concurrency$ ls
a.out     cacheSharing.c  fs-v2.c   reorderF.c        reorderWithComm.c  sync.c   syncM.c
cacheC.c  fs.c            reorder.c reorderSeqCons.c  syncB.c            syncL.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed!
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed! in 161107th Iteration
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed! in 84865th Iteration
(base) kolin@mosaic:~/col7001/concurrency$
```



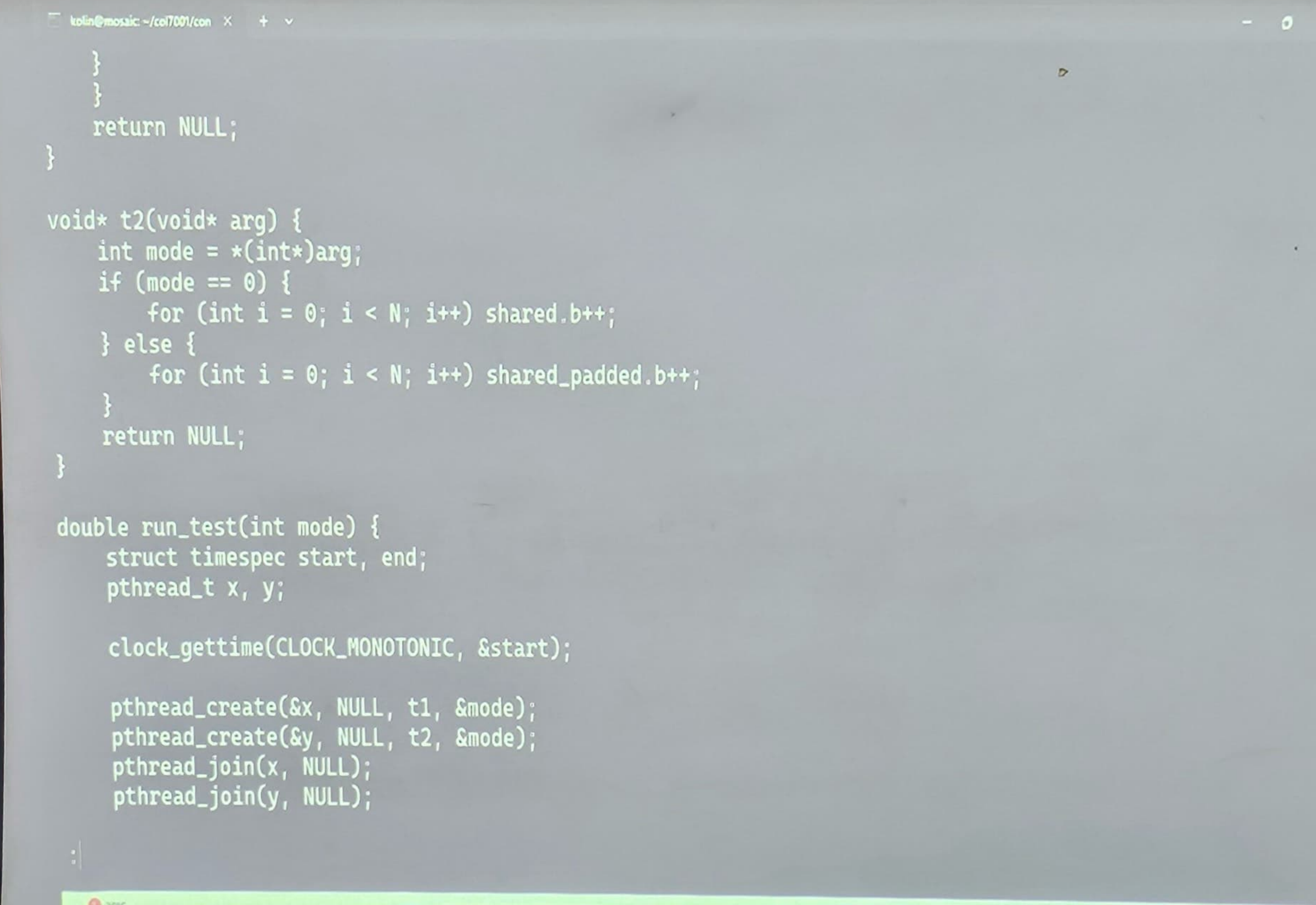
```
void* thread2(void* arg) {
    atomic_store_explicit(&y, 1, memory_order_seq_cst);
    r2 = atomic_load_explicit(&x, memory_order_seq_cst);
    return NULL;
}

int main() {
    for (int i = 0; i < 1000000; i++) {
        atomic_store(&x, 0);
        atomic_store(&y, 0);
        r1 = r2 = 0;

        pthread_t t1, t2;
        pthread_create(&t1, NULL, thread1, NULL);
        pthread_create(&t2, NULL, thread2, NULL);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);

        if (r1 == 0 && r2 == 0) {
            printf("Reordering observed (i=%d)\n", i);
            break;
        }
    }
    return 0;
}

(END)
```



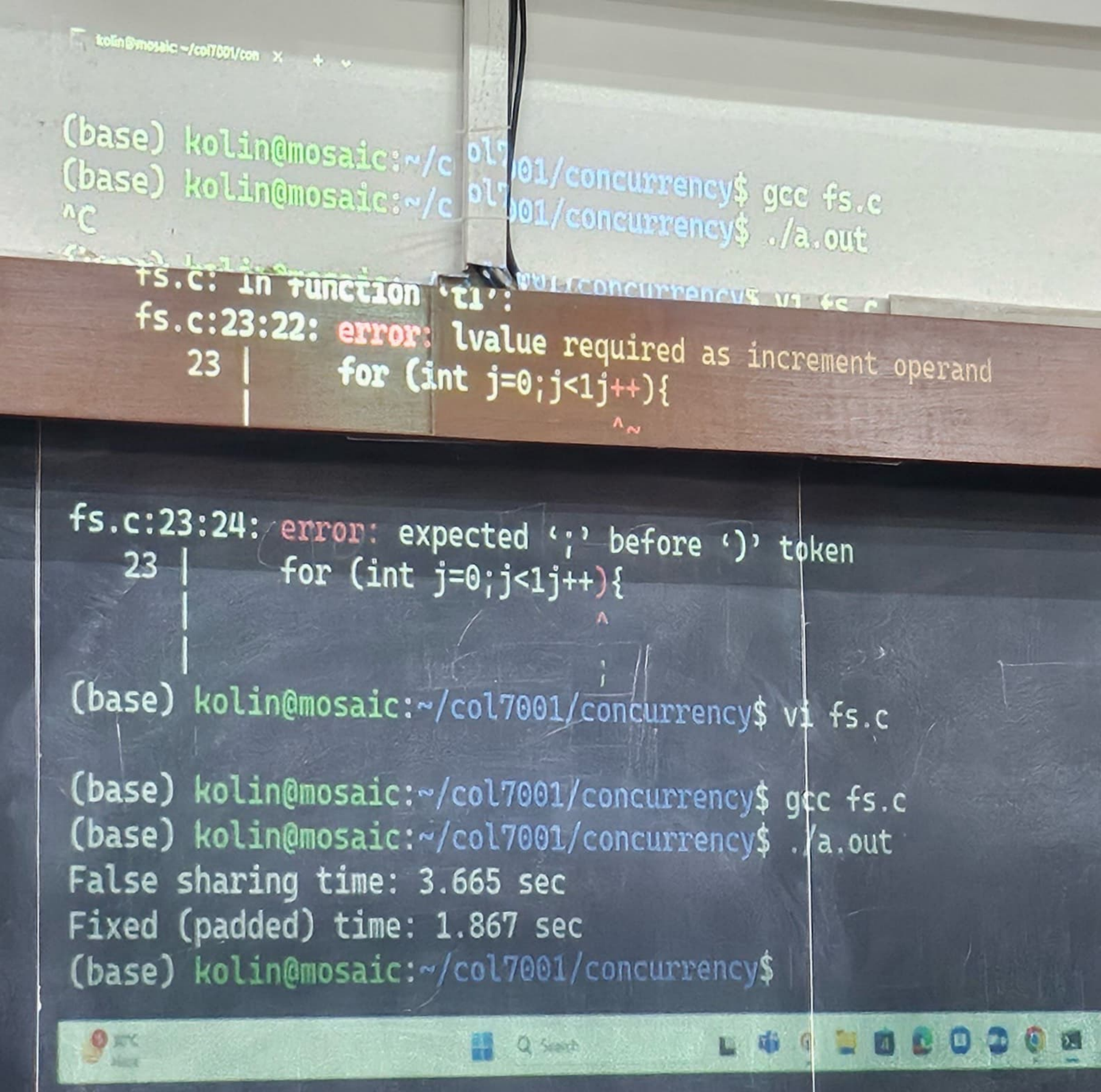
```
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=0    r2=1
r1=1    r2=0
^C
(base) kolin@mosaic:~/col7001/concurrency$ ls
a.out     cacheSharing.c  fs-v2.c    reorderF.c        reorderWithComm.c  sync.c   syncM.c
cacheC.c  fs.c            reorder.c  reorderSeqCons.c  syncB.c            syncL.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed!
(base) kolin@mosaic:~/col7001/concurrency$ less reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderF.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed! in 161107th Iteration
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
Reordering observed! in 84865th Iteration
(base) kolin@mosaic:~/col7001/concurrency$ less reorder
reorder.c          reorderF.c         reorderSeqCons.c   reorderWithComm.c
(base) kolin@mosaic:~/col7001/concurrency$ less reorderSeqCons.c
(base) kolin@mosaic:~/col7001/concurrency$ gcc reorderSeqCons.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
```



```
#include <stdio.h>
#include <pthread.h>
#include <time.h>

#define N 1000000000

// --- Case 1: False Sharing ---
struct {
    int a;   // updated by thread 1
    int b;   // updated by thread 2
} shared;

// --- Case 2: Fixed with Padding ---
struct {
    int a;
    char pad1[64];   // avoid same cache line
    int b;
    char pad2[64];
} shared_padded;

void* t1(void* arg) {
    int mode = *(int*)arg;
    for (int j=0;j<N/10;j++){
    if (mode == 0) {
        for (int i = 0; i < N; i++) shared.a++;
    } else {
fs.c
```



```
        }
    }
    return NULL;
}

void* t2(void* arg) {
    int mode = *(int*)arg;
    if (mode == 0) {
        for (int i = 0; i < N; i++) shared.b++;
    } else {
        for (int i = 0; i < N; i++) shared_padded.b++;
    }
    return NULL;
}

double run_test(int mode) {
    struct timespec start, end;
    pthread_t x, y;

    clock_gettime(CLOCK_MONOTONIC, &start);

    pthread_create(&x, NULL, t1, &mode);
    pthread_create(&y, NULL, t2, &mode);
    pthread_join(x, NULL);
    pthread_join(y, NULL);

:
```



```
(base) kolin@mosaic:~/col7001/concurrency$ gcc fs.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
^C
(base) kolin@mosaic:~/col7001/concurrency$ vi fs.c
fs.c: In function ‘t1’:
fs.c:23:22: error: lvalue required as increment operand
   23 |         for (int j=0;j<1j++){
      |                          ^~
fs.c:23:24: error: expected ‘;’ before ‘)’ token
   23 |         for (int j=0;j<1j++){
      |                            ^
      |                            ;
(base) kolin@mosaic:~/col7001/concurrency$ vi fs.c

(base) kolin@mosaic:~/col7001/concurrency$ gcc fs.c
(base) kolin@mosaic:~/col7001/concurrency$ ./a.out
False sharing time: 3.665 sec
Fixed (padded) time: 1.867 sec
(base) kolin@mosaic:~/col7001/concurrency$
```