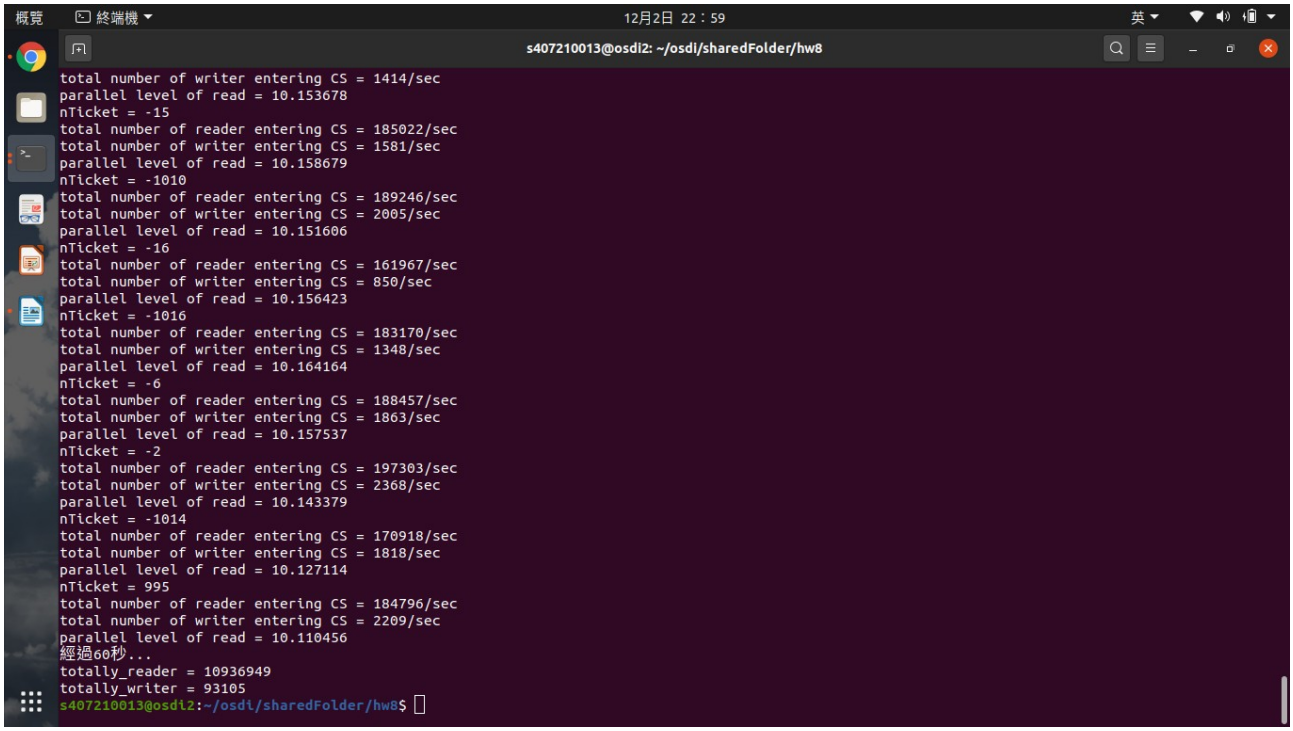
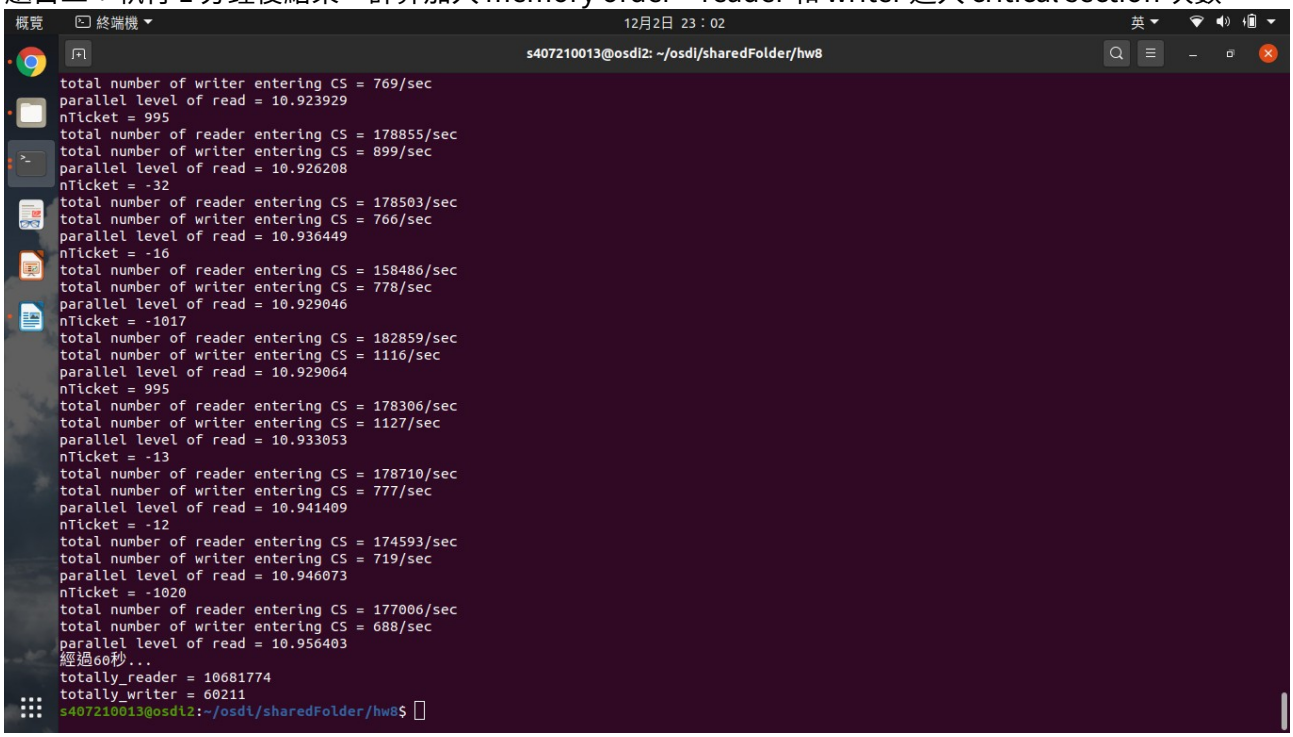
甲

題目一：執行 1 分鐘以後結束執行，並印出以 reader 和 writer 的身份，進入 critical section 的次數。

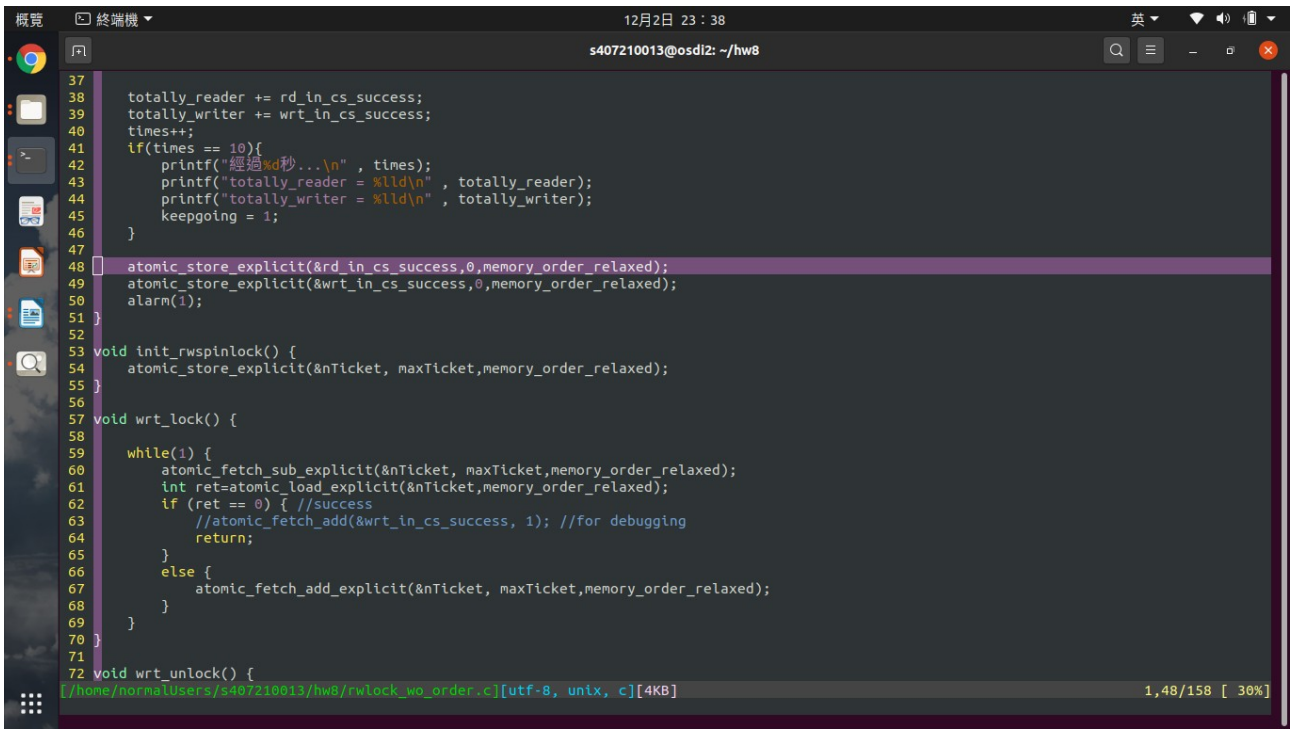


題目二：執行 1 分鐘後結束，計算加入 memory order，reader 和 writer 進入 critical section 次數。



乙

使用 atomic 相關的函數，且「指定 memory_order」，像是 atomic_store_explicit(&rd_in_cs_success , 0 , memory_order_relaxed)，實測出來的結果會是原本進入的次數較多。感覺上因為 memory_order_seq_cst 為其預設值，跟後來使用的 memory_order_relaxed 相比，後者對於資料的順序不加以限制，而前者對於這部份有較強的處理，因此有結果的產生。假設今天有更強的 memory_order，相較此結果，或許就能增加實驗過程中的次數，由此可知可透過 memory order 增加效率。


概覽
終端機
12月2日 23：38
英
s407210013@osdi2: ~/hw8
totally_reader += rd_in_cs_success;
totally_writer += wrt_in_cs_success;
times++;
if(times == 10){
printf("經過%d秒...\n" , times);
printf("totally_reader = %lld\n" , totally_reader);
printf("totally_writer = %lld\n" , totally_writer);
keepgoing = 1;
}
atomic_store_explicit(&rd_in_cs_success,0,memory_order_relaxed);
atomic_store_explicit(&wrt_in_cs_success,0,memory_order_relaxed);
alarm(1);
}
void init_rwspinlock() {
atomic_store_explicit(&nTicket, maxTicket,memory_order_relaxed);
}
void wrt_lock() {
while(1) {
atomic_fetch_sub_explicit(&nTicket, maxTicket,memory_order_relaxed);
int ret=atomic_load_explicit(&nTicket,memory_order_relaxed);
if (ret == 0) { //success
//atomic_fetch_add(&wrt_in_cs_success, 1); //for debugging
return;
}
else {
atomic_fetch_add_explicit(&nTicket, maxTicket,memory_order_relaxed);
}
}
}
void wrt_unlock() {
[/home/normalUsers/s407210013/hw8/rwlock_wo_order.c][utf-8, unix, c][4KB]
1,48/158 [30%]