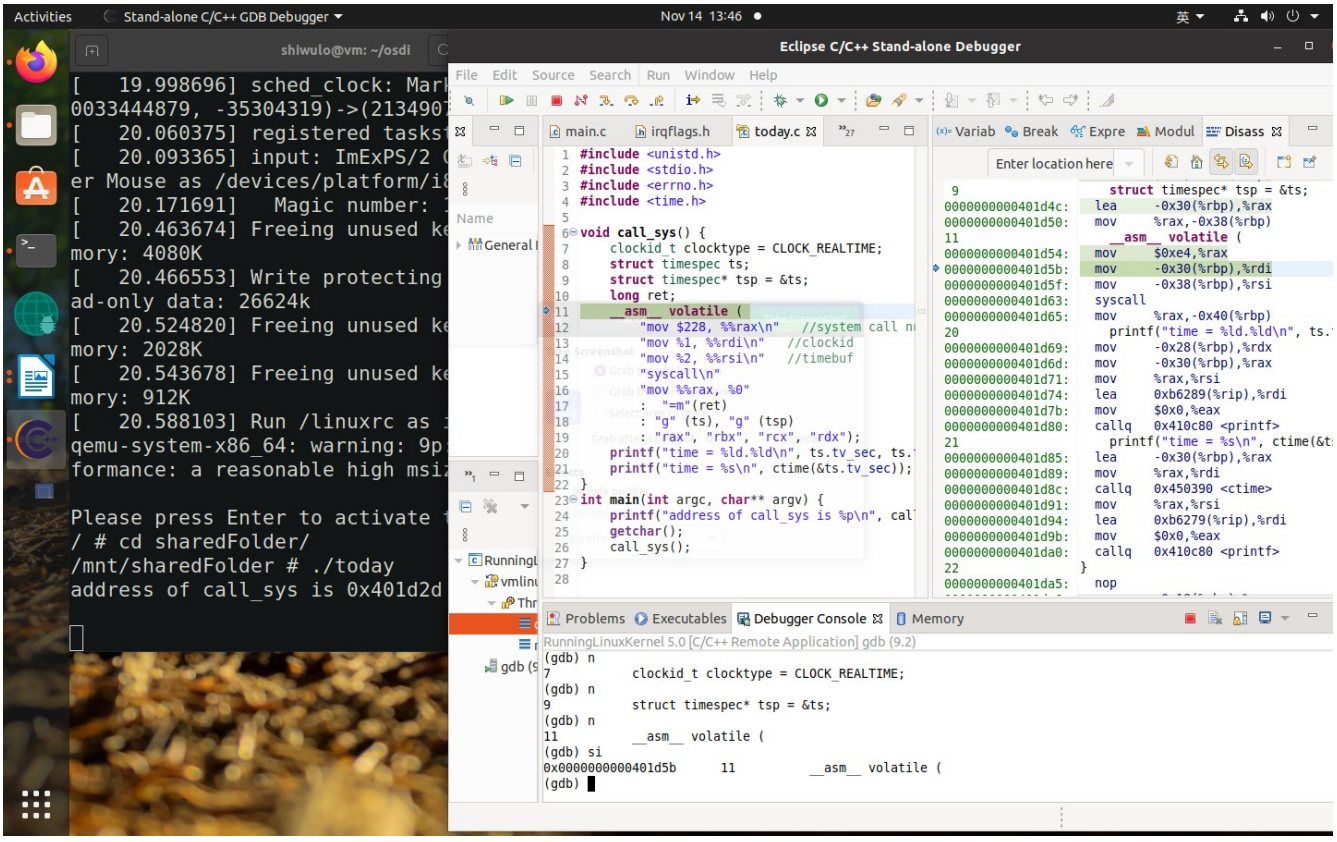
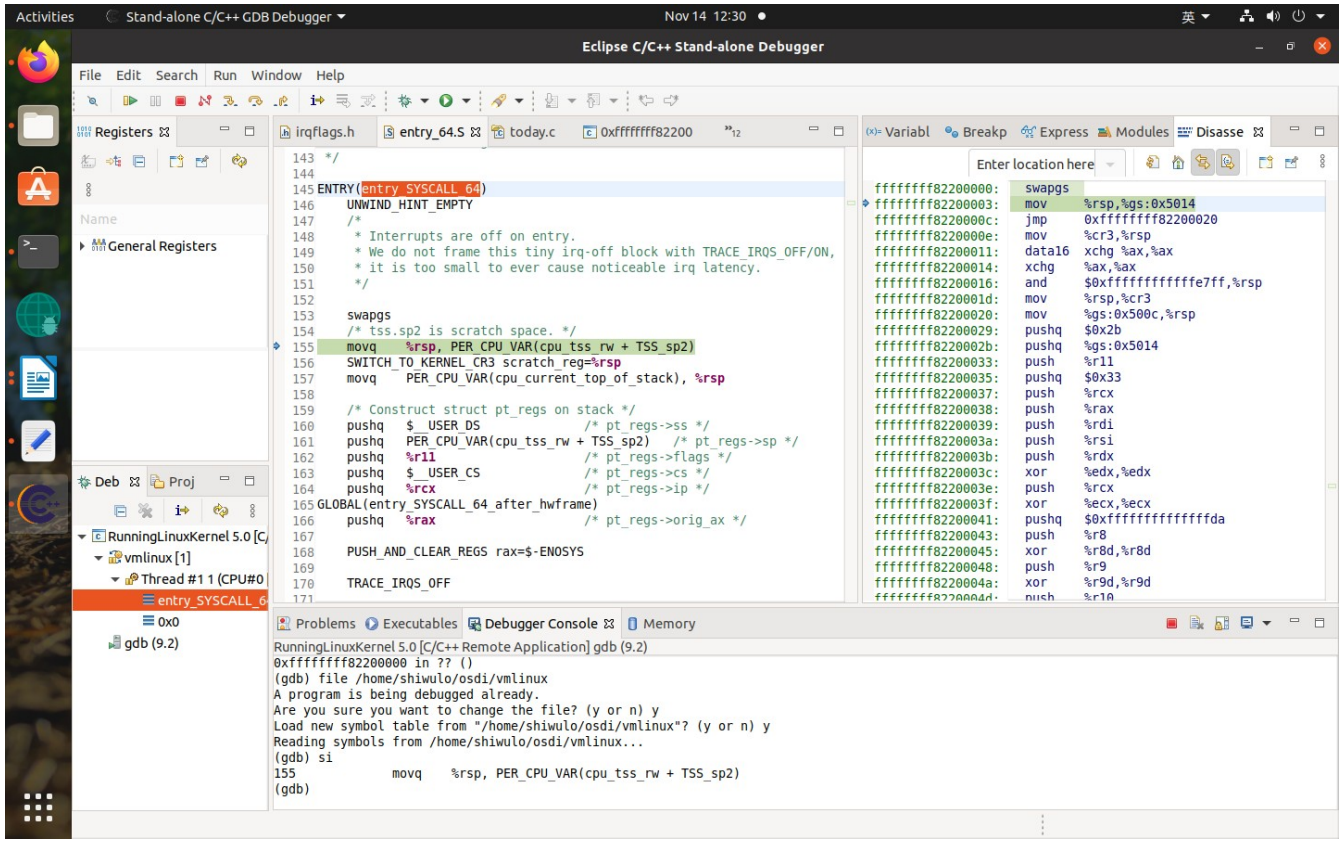
408410007 鄭O云

一:



二:



三:

```
nr = array_index_nospec(nr, NR_syscalls);
            regs->ax = sys_call_table[nr](regs);

SYSCALL_DEFINE2(clock_gettime, const clockid_t, which_clock,struct
__kernel_timespec __user *, tp)
{
      const struct k_clock *kc = clockid_to_kclock(which_clock);
      struct timespec64 kernel_tp;
      int error;

      if (!kc)
            return -EINVAL;

      error = kc->clock_get(which_clock, &kernel_tp);

      if (!error && put_timespec64(&kernel_tp, tp))
            error = -EFAULT;

      return error;
}
```

四:

- SYSCALL_DEFINE2 =>__x64_sys_clock_gettime => __se_sys_clock_gettime => __do_sys_clock_gettime
- clockid_to_kclock(which_clock)應該是檢查which_clock有沒有小於0還是大於posix_clocks陣列的數量，沒有就回傳posix_clocks陣列裡的東西，給kc這個指標指向他
- 如果kc沒有東西，回傳-EINVAL，屬於一種clock的ERROR。
- kc->clock_get(which_clock, &kernel_tp)讓kc拿到clock_realtime。拿到後回傳error=0。（並會WARN，若有延遲）

追出程式碼:

```
//return posix_clocks[array_index_nospec(idx, ARRAY_SIZE(posix_clocks))];
static inline unsigned long array_index_mask_nospec(unsigned long index, unsigned long size)
{
      unsigned long mask;

      asm volatile ("cmp %1,%2; sbb %0,%0;"
                  :"=r" (mask)
                  :"g"(size),"r" (index)
                  :"cc");
      return mask;
}

/* Get clock_realtime */
static int posix_clock_realtime_get(clockid_t which_clock, struct timespec64 *tp)
{
      ktime_get_real_ts64(tp);
      return 0;
}

/**
 * ktime_get_real_ts64 - Returns the time of day in a timespec64.
 * @ts:           pointer to the timespec to be set
 *
 * Returns the time of day in a timespec64 (WARN if suspended).
 */
void ktime_get_real_ts64(struct timespec64 *ts)
```

```
{
	struct timekeeper *tk = &tk_core.timekeeper;
	unsigned long seq;
	u64 nsecs;

	WARN_ON(timekeeping_suspended);

	do {
		seq = read_seqcount_begin(&tk_core.seq);

		ts->tv_sec = tk->xtime_sec;
		nsecs = timekeeping_get_ns(&tk->tkr_mono);

	} while (read_seqcount_retry(&tk_core.seq, seq));

	ts->tv_nsec = 0;
	timespec64_add_ns(ts, nsecs);
}

int put_timespec64(const struct timespec64 *ts,struct __kernel_timespec __user *uts)
{
	struct __kernel_timespec kts = {
		.tv_sec = ts->tv_sec,
		.tv_nsec = ts->tv_nsec
	};

	return copy_to_user(uts, &kts, sizeof(kts)) ? -EFAULT : 0;
}
EXPORT_SYMBOL_GPL(put_timespec64);

static __always_inline bool check_copy_size(const void *addr, size_t bytes, bool is_source)
{
	int sz = __compiletime_object_size(addr);
	if (unlikely(sz >= 0 && sz < bytes)) {
		if (!__builtin_constant_p(bytes))
			copy_overflow(sz, bytes);
		else if (is_source)
			__bad_copy_from();
		else
			__bad_copy_to();
		return false;
	}
	check_object_size(addr, bytes, is_source);
	return true;
}
```