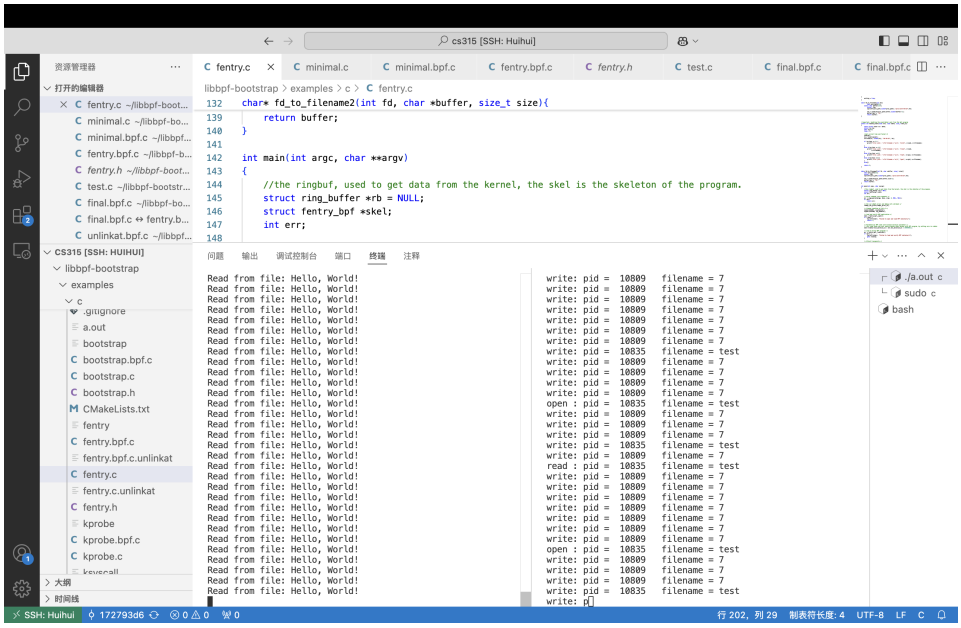
Lab 11

SID: 12110644

Name: Sicheng Zhou

Task 1



task 1 result

```c
static int handle_event(void *ctx, void *data, size_t data_sz)
{
    const struct event *e = data;
    struct tm *tm;
    char ts[32];
    time_t t;

    //get current time and format it
    time(&t);
    tm = localtime(&t);
    strftime(ts, sizeof(ts), "%H:%M:%S", tm);

    if(e->func ==1){
        printf("open : pid =  %-7d filename = %s\n", e->pid, e->filename);
    }
    else if(e->func ==2){
        printf("read : pid =  %-7d filename = %s\n", e->pid, e->filename);
    }
    else if (e->func == 3) {
        printf("write: pid =  %-7d filename = %s\n", e->pid, e->filename);

    }
    else if(e->func == 4){
        printf("close: pid =  %-7d filename = %s\n", e->pid, e->filename);
    }
    else{}

    return 0;
}

```

trigger program handle_event

```c
int main(int argc, char **argv)
{
    //the ringbuf, used to get data from the kernel, the skel is the skeleton of the program.
    struct ring_buffer *rb = NULL;
    struct fentry_bpf *skel;
    int err;

    /* Parse command line arguments */
    err = argp_parse(&argp, argc, argv, 0, NULL, NULL);
    if (err)
        return err;

    /* Set up libbpf errors and debug info callback */
    libbpf_set_print(libbpf_print_fn);

    /* Cleaner handling of Ctrl-C */
    signal(SIGINT, sig_handler);
    signal(SIGTERM, sig_handler);

    /* Load and verify BPF application */
    skel = fentry_bpf__open();
    if (!skel) {
        fprintf(stderr, "Failed to open and load BPF skeleton\n");
        return 1;
    }
    skel->rodata->min_duration_ns = env.min_duration_ms * 1000000ULL;
    err = fentry_bpf__load(skel);
    if (err) {
        fprintf(stderr, "Failed to load and verify BPF skeleton\n");
        goto cleanup;
    }
    err = fentry_bpf__attach(skel);
    if (err) {
        fprintf(stderr, "Failed to attach BPF skeleton\n");
        goto cleanup;
    }
    rb = ring_buffer__new(bpf_map__fd(skel->maps.rb), handle_event, NULL, NULL);
    if (!rb) {
        err = -1;
        fprintf(stderr, "Failed to create ring buffer\n");
        goto cleanup;
    }

```

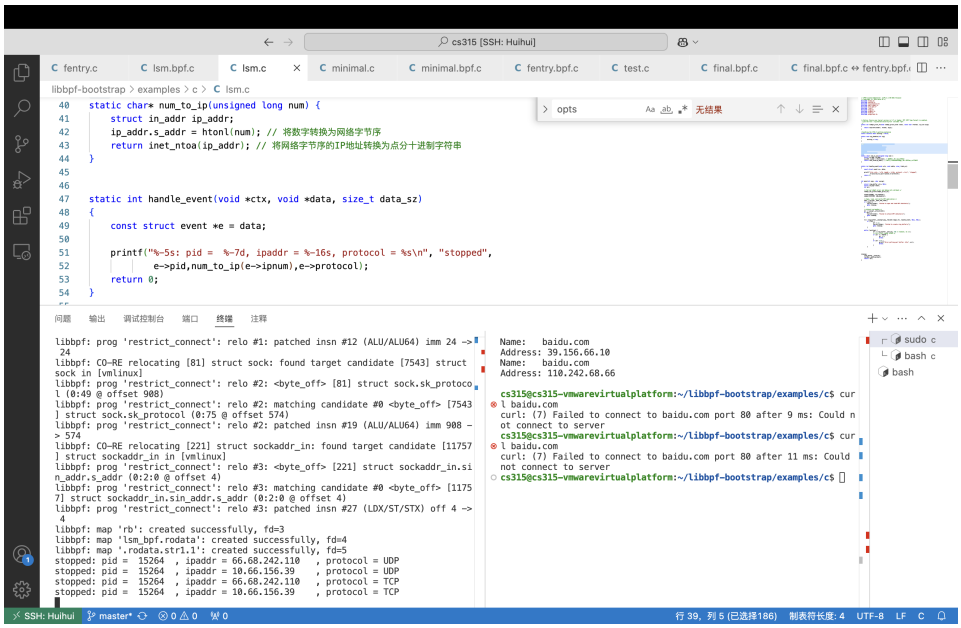
load, verify, attach

```c
    /* Process events */
    while (!exiting) {
        err = ring_buffer__poll(rb, 100 /* timeout, ms */);
        if (err == -EINTR) {
            err = 0;
            break;
        }
        if (err < 0) {
            printf("Error polling perf buffer: %d\n", err);
            break;
        }
    }

```

process events

Task 2



task 2 result