**Đồ án Hệ điều hành**

1. **Thông tin nhóm**
2. **Phân công và đánh giá mức độ hoàn thành**
3. **Chi tiết đồ án**
4. **Tổng quan**

* **Linux Kernel là gì?**
* Kernel module là một cơ chế giúp thêm hoặc loại bỏ code ra khỏi Linux kernel ở thời gian thực. Cơ chế này thích hợp cho những driver các thiết bị, kích hoạt kernel giao tiếp với phần cứng mà không cần phải biết rõ nguyên lý hoạt động của phần cứng.
* Kernel module chạy ở Kernel Space và các ứng dụng chạy ở User Space (hình 1). Cả kernel space và user space đều có những vùng địa chỉ nhớ riêng biệt mà không bị trùng lắp nhau. Phương pháp này đám bảo các ứng dụng chạy ở user space có một cái nhìn nhất quán đối với phần cứng, bất kể nền tảng nào. Các dịch vụ ở kernel được cung cấp cho user space thông qua system calls. Kernel ngăn các ứng dụng ở user space có xung đột với nhau hoặc truy cập vào tài nguyên không cho phép thông qua các mức bảo vệ.

* **Làm sao để xây dựng một kernel module?**

Một phương pháp để xây dựng một kernel module là sử dụng kernel code. Khác với việc viết một chương trình ứng dụng, một kerel module không có hàm main() và có một số điểm lưu ý như sau:

* Không được thực thi một cách tuần tự: một kernel module sẽ tự đăng kí để xử lý những yêu cầu sử dụng các hàm khởi tạo của chính nó. Các hàm đó sẽ chạy và dừng sau đó. Các loại yêu cầu mà nó xử lý có thể được định nghĩa bên trong module code.
* Không tự động dọn dẹp: bất cứ tài nguyên nào được cấp phát ở module code đều phải được giải phóng một cách thủ công khi mà module dừng. Nếu không nó sẽ vẫn chiếm một vùng tài nguyên cho đến khi hệ thống khởi động lại.
* Không có printf(): kernel code không thể truy cập những thư viện code được viết ở tầng user space. Kernel module chỉ chạy ở kernel space và có một vùng nhớ riêng. Có một sự giao tiếp giữa kernel space và user space được định nghĩa. Do vậy, ta sẽ dùng printk() thể xuất thông tin từ user space.
* Có thể bị gián đoạn.
* Có đặc quyền thực thi ở cấp độ cao hơn.
* Không hỗ trợ chấm động: kernel code sử dụng traps để chuyển đổi từ số nguyên sang số chấm động cho các ứng dụng ở user space.

Cần phải có Makefile để xây dựng một kernel module. Cấu trúc một Makefile như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | obj-m+= modulename.o    all:  make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) modules  clean:  make -C /lib/modules/$(shell uname -r)/build/ M=$(PWD) clean |

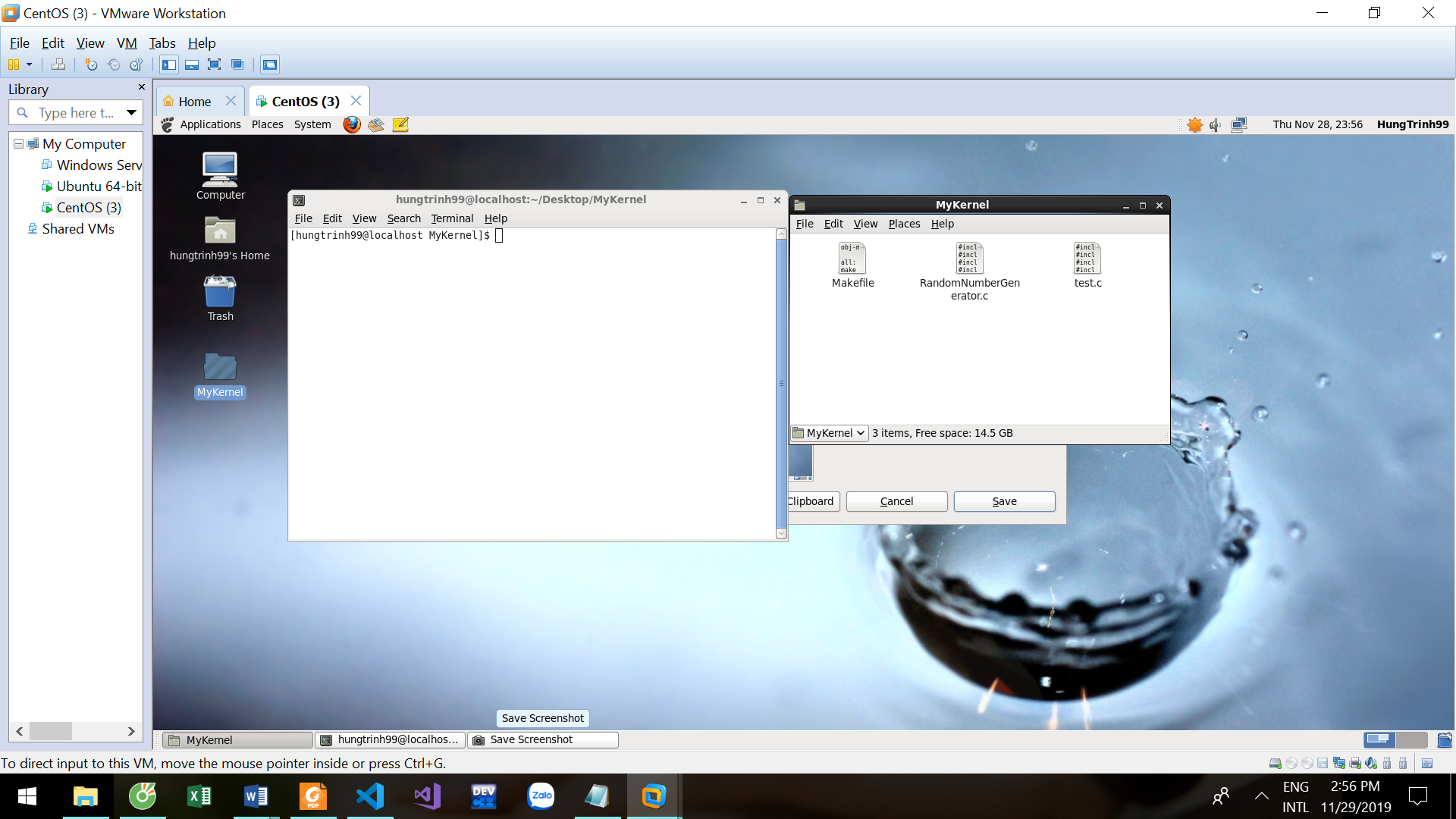
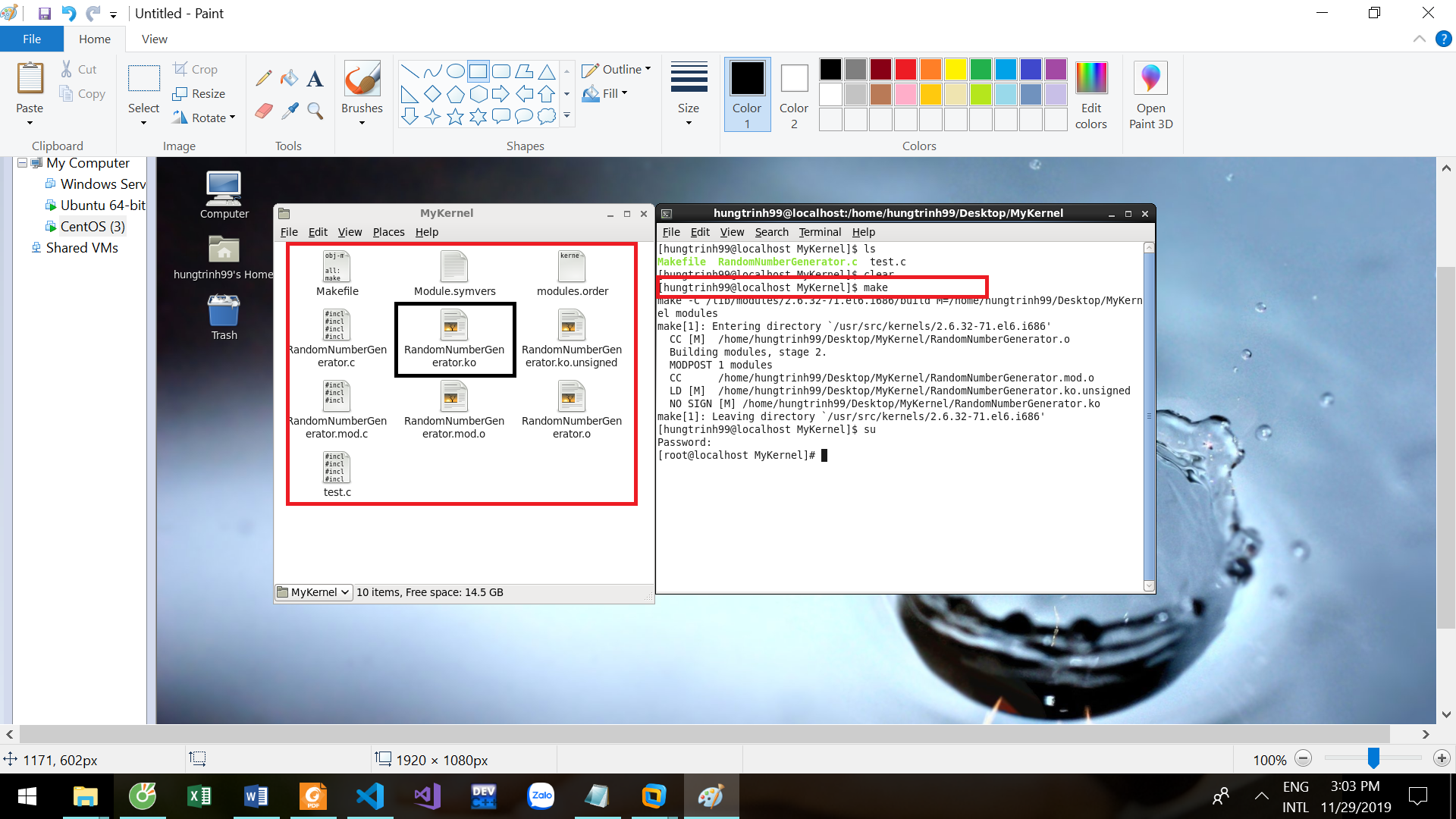
Dòng đầu tiên là goal definition. Nó sẽ khai báo tên module để xây dựng. Sử dụng obj-m để khai báo một module. $shell uname -r là một lời gọi để trả về phiên bản kernel hiện tại. Quá trình cài đặt module được gọi qua **make**.

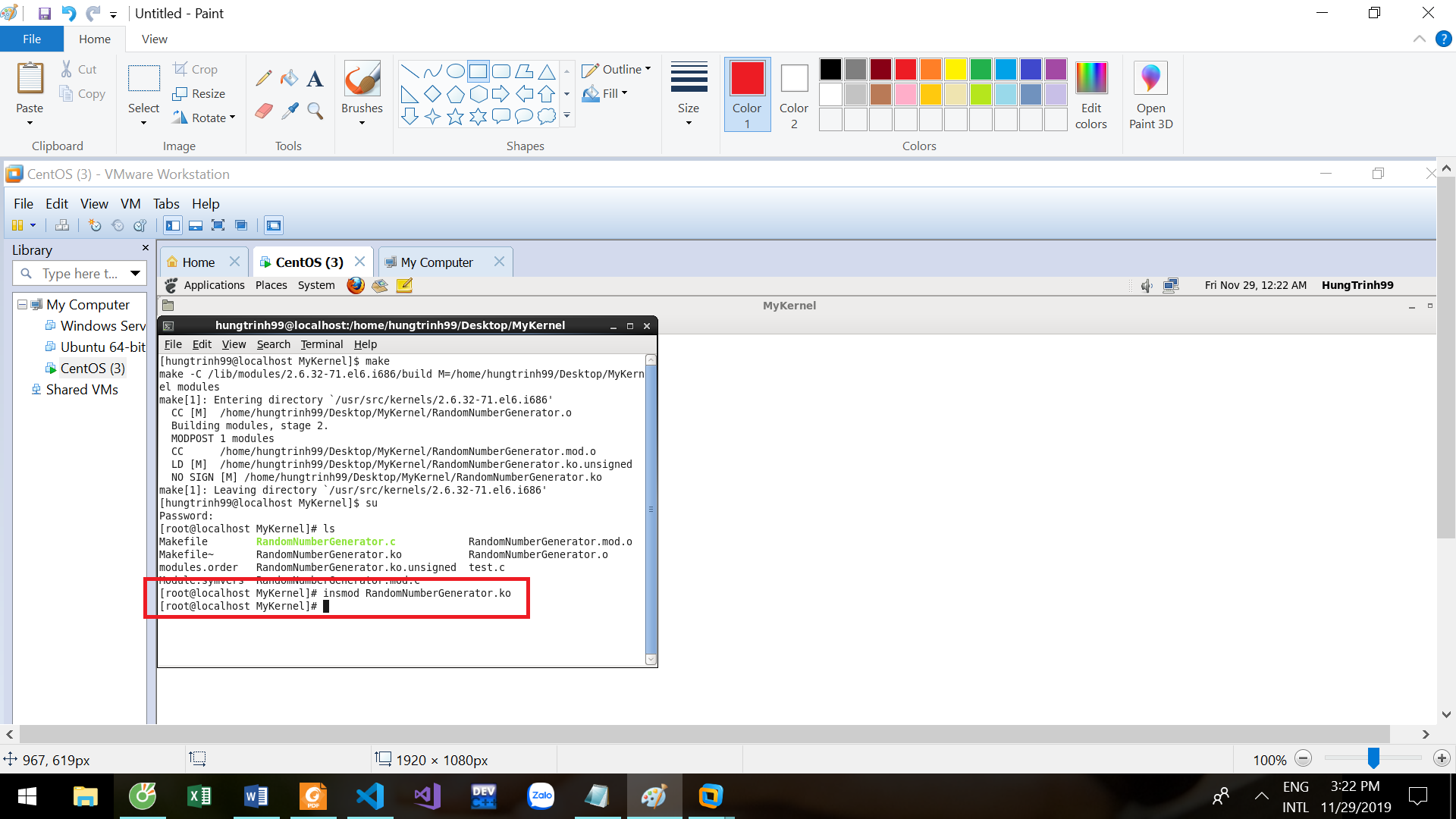
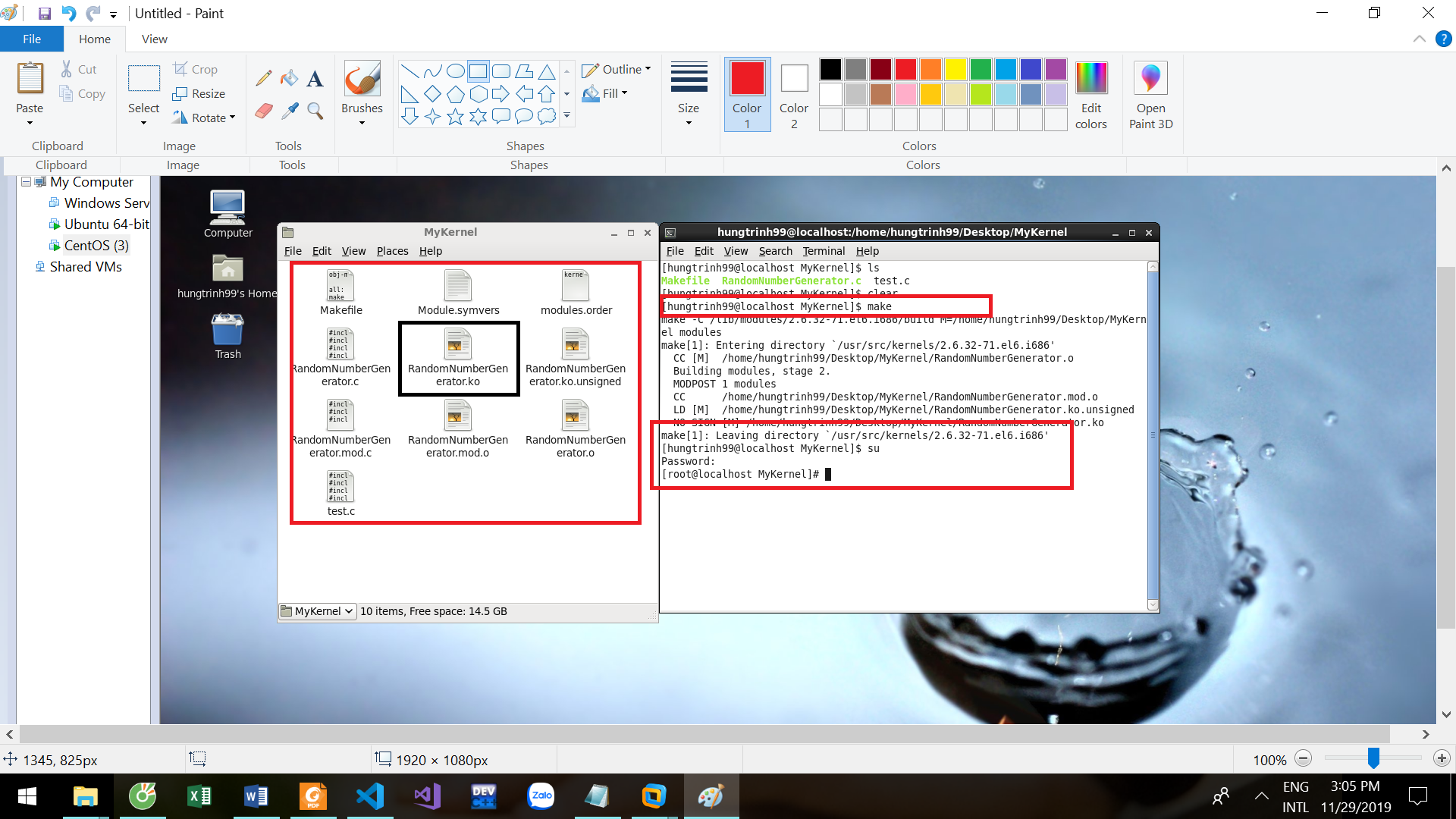
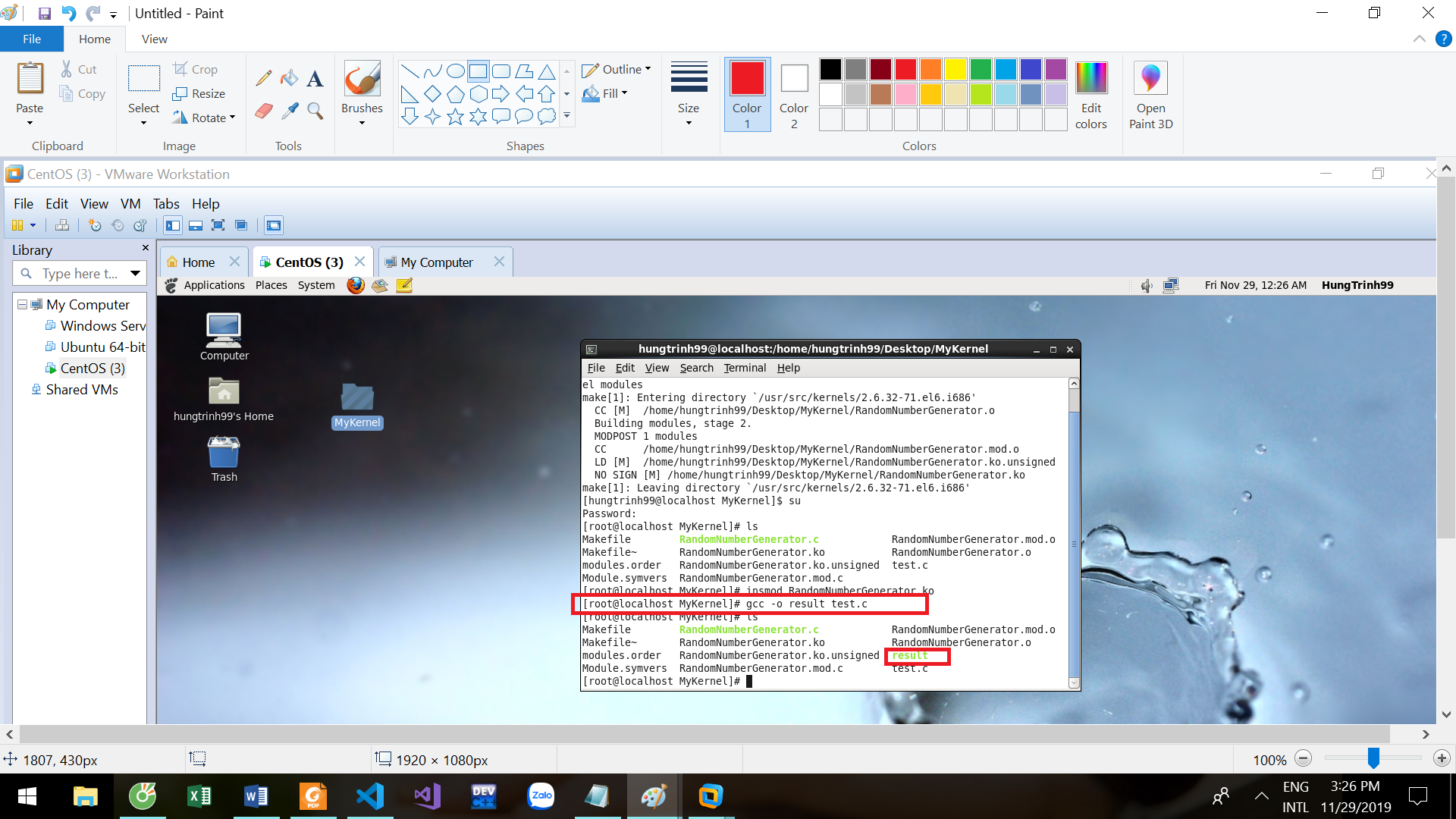
1. **Yêu cầu 1**

Xây dụng một kernel module dùng để sinh ra một số ngẫu nhiên, và module này cho phép các tiến trình ở User space có thể open và read các số ngẫu nhiên đó.

1. **Các hàm chức năng RandomNumberGenerator.c**

* **static int \_\_init generDev\_init(void)**
  + **Chức năng:** Hàm được dùng để khởi tạo khi xây dựng một driver. Nó như là một constructor các giá trị khởi tạo sẽ được thực hiện trong hàm này, nó được gọi lúc ban đầu khi module được load lên.
  + **Quy trình thực hiện:**
    - Đầu tiên hàm này sẽ đăng kí với hệ thống để hệ thống cấp cho một majorNumber. Nếu thành công (majorNumber < 0) thì đăng kí thất bại.
    - Sau khi đăng kí thành công. Tiếp theo ta tiến hành khởi tạo một lớp thiết bị (device class) với câu lện genClass = class\_create(THIS\_MODULE, CLASS\_NAME); Hàm này sẽ trả về một con trỏ hàm đến class device được khởi tạo nếu thành công. Ngược lại chúng ta phải hủy bỏ đi số majorNumber đã đăng kí. Sau đó thông báo ra màn hình việc đăng kí thất bại.
    - Cuối cùng sẽ đăng kí một driver qua lệnh: genDevice = device\_create(genClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE\_NAME); Hàm này sẽ trả về con trỏ kiểu device-driver device struct được định nghĩa ở "driver.h" nếu thành công. Ngược lại chúng ta sẽ hủy bỏ genClass và majorNumber vừa mới đăng kí.
* **static void \_\_exit generDev\_exit(void)**
  + **Chức năng:** Giúp xác định được hàm nào được thực thi trước khi tháo module ra khỏi kernel, hàm này có chức năng giống như một Destructor
  + **Quy trình thưc hiện:** Hàm sẽ được gọi trong quá trình unload module, nó sẽ giải phóng những con trỏ được cấp phát và hủy đăng kí majorNumber.
* **static ssize\_t my\_read(struct inode \*inodep, struct file filep,char buffer, size\_t len loff\_t \*offset)**
  + **Chức năng:** Hàm được gọi khi User space muốn đọc dư liệu vào từ device.
  + **Quy trình thực hiện** 
    - Lệnh: get\_random\_bytes để sinh ra một số ngẫu nhiên.
    - Sau đó gửi dữ liệu này đến User space thông qua lệnh copy\_to\_user(buffer,numbergen,sizeof(int)); Hàm này sẽ trả về 0 nếu thành công.
* **static ssize\_t my\_release(struct inode \*inodep, struct file \*filep)**
  + **Chức năng**: Hàm này được thực thi khi đóng 1 device.
  + **Quy trình thực hiện:** trong hàm này chỉ xuất một thông báo màn hình thiết bị đã đóng bằng **printk**
* **static ssize\_t my\_open(struct inode \*inodep, struct file \*filep)**
  + **Chức năng:** hàm được gọi khi nhận yêu cầu mở từ các file test.c chức các dòng lệnh yêu cầu mở.
  + **Quy trình thực hiện:** Hàm sẽ có một biến toàn cục numberOpens để đếm số lần module này được yêu cầu mở và xuất ra thông tin này khi nhận được yêu cầu open

1. **Hàm test.c** 
   * Đây là một chương trình C, được thực hiện ở User space. Chương trình này gọi lệnh open(“dev/generDev”,O\_RDWR), để yêu cầu mở một devcice và random số ngẫu nhiên. Nếu mở thành công, tiến trình sẽ đọc được số ngẫu nhiên mà device trả về và xuất ra số nhận đươc.
2. **Chạy demo**
   * Mở terminal và cd đến thực mục cần make file:
   * Thư mục lúc này gồm 3 file: Makefile, RandomNumberGenerator.c, test.

* Make file: load một devic: **$ insmod RandomNumberGenerator.ko**
* Chuyển về quyền root: $ su -> nhập password
* Chạy file **test.c: $ gcc –o result test.c**
* Chạy file result để nhận được kết quả từ driver trả về là các số ngẫu nhiên được sinh ra: **./result**



* Ngắt kết nối: **$ rmmod RandomNumberGenerator**

