Autor: Eduardo Murillo

Proyecto Final

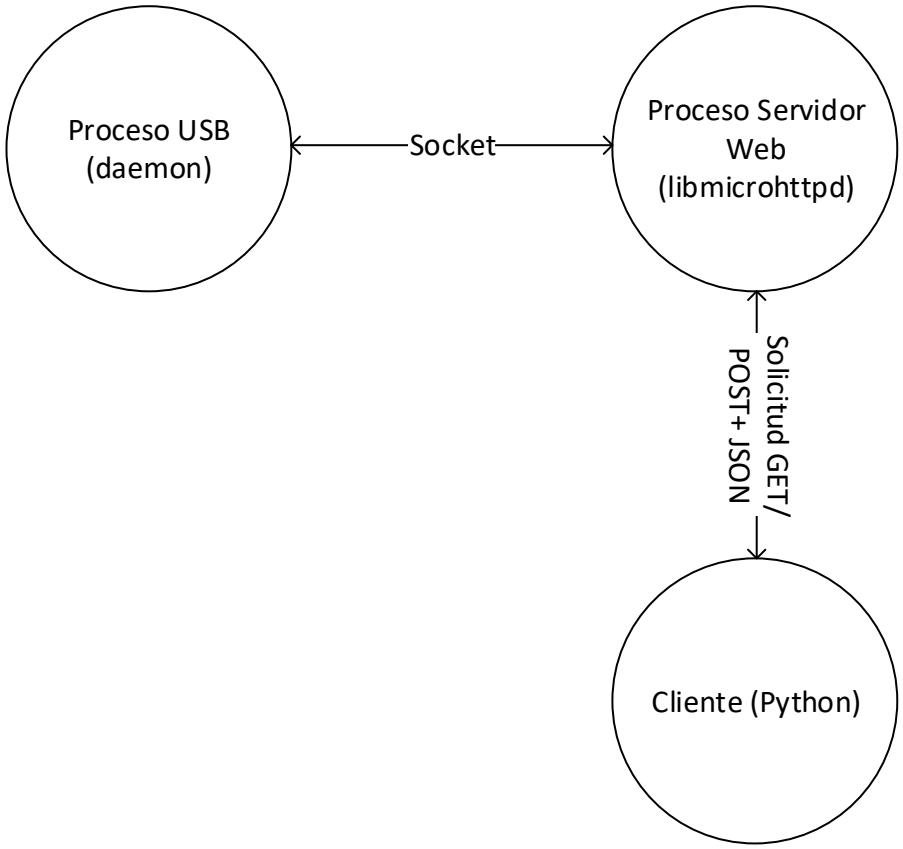
Introducción

En este proyecto usarán todos lo aprendido durante el semestre, para implementar un proyecto con diversa funcionalidad.

Descripción

En este proyecto, crearán un programa que se encargará de monitorear los puertos USB, detectar nuevos dispositivos. El programa usará el formato JSON y un API REST para comunicarse con clientes externos, que darán órdenes a su programa. Su programa ejecutará estas órdenes, y devolverá resultados al cliente. Se explicarán cada una de estas partes a continuación.

Arquitectura General



El proyecto consta de tres partes:

1. **Un proceso** que monitorea los dispositivos USB conectados al PC (como daemon).
2. **Un proceso** que funciona como servidor web.
3. Un cliente escrito en Python.

La premisa es que el cliente pueda enviar comandos al proceso web, usando un API REST (https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) + JSONs (http://www.json.org/), para que el proceso USB pueda manipular archivos en los dispositivos conectados. Por ejemplo:

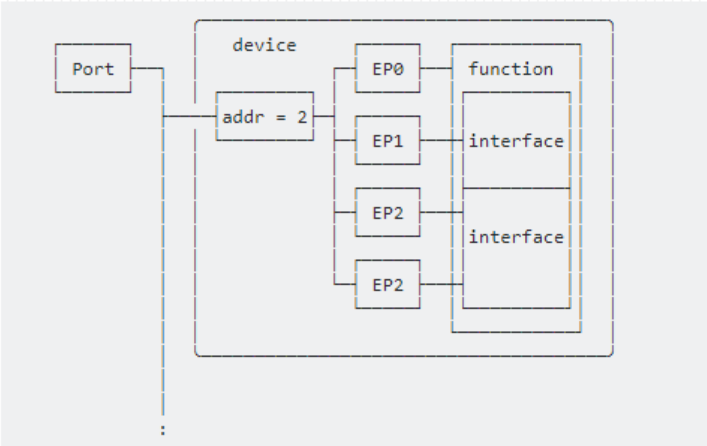
Autor: Eduardo Murillo

1. Listar los dispositivos conectados
2. Escribir un archivo recibido desde cliente y escribirlo en un dispositivo USB
3. Leer un archivo en uno de los dispositivos USB, y enviárselo al cliente.

Cada una de estas partes se explicará a continuación. Ambos procesos se comunicarán mediante un socket. Para comunicar el proceso web, usaran la librería libmicrohttpd.

Parte 1: Proceso USB

En esta parte, Uds. manejarán los dispositivos USB. Un dispositivo USB tiene una dirección, y varios puntos finales (end-points, EP). El EP0 se utiliza para configurar el dispositivo. Un dispositivo USB ofrece una o más interfaces, cada una de las cuales indica una funcionalidad del dispositivo (por ejemplo, una webcam puede presentar una interfaz de video, y otra de recepción de audio, por ejemplo). Los EPs se vinculan a las interfaces:



Las interfaces poseen *clases*, que nos indican el tipo funcionalidad de la misma, por ejemplo:

- HID: dispositivo de interfaz humana (teclado, mouse, etc)
- MSC/UMS: Mass storage device (almacenamiento masivos)
- Otros más.

El proceso de escribir a dispositivos USB directamente es complejo, y existen librerías para ello, como **libusb.** Sin embargo, para nuestros propósitos, usaremos una interfaz de mayor nivel para manejar estos dispositivos.

En esta parte del proyecto, su programa estará constantemente monitoreando los puertos USB y **detectará** los **dispositivos de almacenamiento masivo** y mantendrá una lista actualizada con los dispositivos conectados. Para esta parte, usarán la librería **libudev (**http://presbrey.scripts.mit.edu/doc/libudev/api-index-full.html **).** Abajo se proveen dos funciones que tendrán que modificarlas para cumplir este propósito:

Estas funciones obtienen los dispositivos de almacenamiento masivo, y el **nodo** que hace referencia a ellos.

Autor: Eduardo Murillo

```c
struct udev_device* obtener_hijo(struct udev* udev, struct udev_device* padre, const char* subsistema)
{
    struct udev_device* hijo = NULL;
    struct udev_enumerate *enumerar = udev_enumerate_new(udev);

    udev_enumerate_add_match_parent(enumerar, padre);
    udev_enumerate_add_match_subsystem(enumerar, subsistema);
    udev_enumerate_scan_devices(enumerar);

    struct udev_list_entry *dispositivos = udev_enumerate_get_list_entry(enumerar);
    struct udev_list_entry *entrada;

    udev_list_entry_foreach(entrada, dispositivos) {
        const char *ruta = udev_list_entry_get_name(entrada);
        hijo = udev_device_new_from_syspath(udev, ruta);
        break;
    }

    udev_enumerate_unref(enumerar);
    return hijo;
}

static void enumerar_disp_alm_masivo(struct udev* udev)
{
    struct udev_enumerate* enumerar = udev_enumerate_new(udev);

    //Buscamos los dispositivos USB del tipo SCSI (MASS STORAGE)
    udev_enumerate_add_match_subsystem(enumerar, "scsi");
    udev_enumerate_add_match_property(enumerar, "DEVTYPE", "scsi_device");
    udev_enumerate_scan_devices(enumerar);

    //Obtenemos los dispositivos con dichas caracteristicas
    struct udev_list_entry *dispositivos = udev_enumerate_get_list_entry(enumerar);
    struct udev_list_entry *entrada;

    //Recorremos la lista obtenida
    udev_list_entry_foreach(entrada, dispositivos) {
        const char* ruta = udev_list_entry_get_name(entrada);
        struct udev_device* scsi = udev_device_new_from_syspath(udev, ruta);

        //obtenemos la informacion pertinente del dispositivo
        struct udev_device* block = obtener_hijo(udev, scsi, "block");
        struct udev_device* scsi_disk = obtener_hijo(udev, scsi, "scsi_disk");

        struct udev_device* usb
            = udev_device_get_parent_with_subsystem_devtype(scsi, "usb", "usb_device");

        if (block && scsi_disk && usb) {
            printf("block = %s, usb = %s:%s, scsi = %s\n",
                   udev_device_get_devnode(block),
                   udev_device_get_sysattr_value(usb, "idVendor"),
                   udev_device_get_sysattr_value(usb, "idProduct"),
                   udev_device_get_sysattr_value(scsi, "vendor"));
        }

        if (block) {
            udev_device_unref(block);
        }

        if (scsi_disk) {
            udev_device_unref(scsi_disk);
        }

        udev_device_unref(scsi);
    }

    udev_enumerate_unref(enumerar);
}
```

Ahora bien, para acceder al dispositivo, no basta saber el **nodo (/dev/…)** del dispositivo físico. Para hacer uso del mismo, debemos saber dónde está montado. Para esto, usarán la librería **mtab (https://www.gnu.org/software/libc/manual/html_node/mtab.html).** Con ayuda de esta librería, podrán asociar el nodo con el directorio donde está montado el **dispositivo USB.**

Autor: Eduardo Murillo

Para cada dispositivo deben registrar:

1. El nodo (/dev/…)
2. El punto de montaje (/…)
3. Un id (idVendor:idProduct)
4. Un nombre (originalmente vacío, luego usuario lo podrá configurar)

Este proceso debe crearse como un daemon (https://notes.shichao.io/apue/ch13/). Es decir, correrá como un servicio en el fondo.

Parte 2: Comunicación

El programa se comunicara con clientes externos, que darán órdenes al programa. El mecanismo de comunicación debe serán solicitudes GET. Para esto usaremos la libería **libmicrohttpd (https://www.gnu.org/software/libmicrohttpd/tutorial.html). USEN LA VERSION QUE VIENE POR DEFECTO EN UBUNTU: sudo apt-get install libmicrohttpd***.Las solicitudes irán con información adjunta, que será un JSON con las órdenes del cliente. Así mismo, las respuestas del servidor web serán solicitudes con JSON adjunto. Para hacer *parsing* del JSON, usarán Jsmn (**https://github.com/zserge/jsmn**). Este proceso se comunicará mediante sockets con el proceso USB.

Las solicitudes y respuestas se detallan a continuación:

Tipo Solicitud → GET
Solcitud → <IP>/listar_dispositivos
Código respuesta → 200 OK
JSON Respuesta → { 'solicitud': 'listar_dispositivos', 'dispositivos': [{
'nombre': ...,
'id': 'vendor:device',
'montaje' : '/home/…',
'nodo': '/dev/…'
}, { …}],
'status': …, 'str_error' : …
}

Status 0 cuando fue existoso, -1 en error.

Tipo Solicitud → POST
Solcitud → <IP>/nombrar_dispositivo
Código respuesta → 200 OK
JSON Solicitud → { 'solicitud': 'nombrar_dispositivo', 'nodo': '/dev/…', 'nombre': … }
JSON Respuesta → { 'solicitud': 'nombrar_dispositivo', 'status: 0, 'nombre': ..., 'nodo': '/dev/…' , 'str_error' : … }

Status 0 cuando fue existoso, -1 en error.

Autor: Eduardo Murillo

Tipo Solicitud → GET
Solcitud → <IP>/leer_archivo
Código respuesta → 200 OK
JSON Solicitud → { 'solicitud': 'leer_archivo', 'nombre': …, 'nombre_archivo': ... }
JSON Respuesta → { 'solicitud': 'leer_archivo', 'nombre...', 'nombre_archivo': ... , 'contenido': ..., 'str_error' : …}

Status 0 cuando fue existoso, -1 en error.

Tipo Solicitud → POST
Solcitud → <IP>/escribir_archivo
Código respuesta → 200 OK
JSON Solicitud → { 'solicitud': 'escribir_archivo', 'nombre': ..., 'nombre_archivo': ..., 'tamano_contenido: …, 'contenido': ….}
JSON Respuesta → { 'solicitud': 'escribir_archivo', 'nombre...', 'nombre_archivo': ... , 'status: …, , 'str_error' : …}

Status 0 cuando fue existoso, -1 en error.

El campor str_error debe devolver una explicación CLARA de la causa del error.

Parte 3: Cliente

En esta parte implementaran un programa cliente en **Python**, para probar la funcionalidad de su programa.

Calificación

El proyecto será calificado en base a funcionalidad. **DEBE EXISTIR UN MAKEFILE. Si este no existe o no compila el proyecto, es 0 automático en el mismo.** Su proyecto debe estar organizado en varios archivos y directorios, como hemos venido haciendo durante el semestre.

Rúbrica:

Puntos a favor

Solicitudes realizan la acción correcta:	+25 puntos/solicitud

Puntos en contra

Segmentation fault	-50/segfault
Warnings al compilar	-10/warning
No hay makefile o no funciona	0 en proyecto
Proyecto no compila por error de sintaxis	0 en el proyecto
Cuando hay error no se especifica la causa (JSON)	-10/error
Programa está escrito en un solo archivo	-30 puntos
Errores misceláneos (mal nombre de archivo, Se escribe en dispositivo equivoado, etc)	-5/error

Autor: Eduardo Murillo

Solicitudes no se enván como JSON	-50 puntos

Entregable:

Un repositorio de Git con todos los archivos del proyecto. Debe existir un archivo README con toda la información necesaria para crear y ejecutar el proyecto. Trabajo grupal. FECHA DE ENTREGA: Domingo 3 de Septiembre, 2017.