UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

CARRERA DE INGENIERÍA CIENCIAS Y SISTEMAS

LABORATORIO DE SISTEMAS OPERATIVOS 2

“SÉPTIMO SEMESTRE”

ING: FRANCISCO ROJAS

ESTUDIANTE: Mario Moisés Ramírez Tobar - 201830007

Trabajo: “PROYECTO 1”

FECHA: 22 de septiembre de 2,021

INTRODUCCIÓN

La siguiente investigación e informe es sobre una práctica basada en visualizar las interrupciones de la computadora, y estas mostrarlas en pantalla, esto utilizando lo que es el lenguaje assembler, para la lectura de la interrupción y cálculo de pulsaciones en un determinado tiempo, para contar hasta tres, donde también se compartirá a la memoria este número, donde otra aplicación en C, estará escuchando, leyendo para escribir lo encontrado.

Para esto necesitaremos lo que es un listener para que ocurran las interrupciones, ya que por sí solo no ocurren, y se necesitará también de un módulo, que esté atento a las interrupciones para mandar una bandera, la cuál escuchará el sender, este compartirá en memoria la información y el sender la leerá y la compartirá en pantalla.

OBJETIVOS

Objetivos Generales

- Realizar una interrupción de un botón con el cable serial de usb a rs232
- Obtener la lectura de la interrupción y mostrar cuántas veces se ha pulsado, hasta un número de tres veces.

Objetivos Específicos

- Realizar circuito para pulsar un botón conectándolo con un rs232
- Realizar módulo de lectura de interrupciones.
- Realizar lectura de cantidad de pulsaciones hasta tres en Assembler.
- Visualizar las pulsaciones por medio de un programa C

MARCO TEÓRICO

KERNEL

El Kernel o núcleo, es una parte fundamental del sistema operativo que se encarga de conceder el acceso al hardware de forma segura para todo el software que lo solicita, el Kernel es una pequeña e invisible parte del sistema operativo, pero la más importante, ya que sin esta no podría funcionar. Todos los sistemas operativos tienen un Kernel, incluso Windows 10, pero quizá el más famoso es el Kernel de Linux, que ahora además está integrado en Windows 10 con sus últimas actualizaciones.

Este núcleo de los sistemas operativos se ejecuta en modo privilegiado con acceso especial a los recursos del sistema para poder realizar las peticiones de acceso que le va pidiendo el software que lo necesita, además como los recursos no son ilimitados, también hace de árbitro a la hora de asignarlos, decidiendo el orden de las peticiones recibidas según la prioridad e importancia de estas. Una gestión muy importante y fundamental que en la mayoría de las ocasiones pasa desapercibida aún siendo un trabajo esencial para coordinar todo el hardware con el software.

El kernel de Windows es privado y su código lo tienen a salvo en el que solamente Microsoft puede realizar modificaciones para mejorar sus próximas versiones de Windows 10, al igual pasa con el de macOS, basado en Unix pero con licencia privada se reserva al equipo de desarrollo encargado en Apple de realizar estas modificaciones, pero Linux tiene un Kernel público bajo licencia GPL v2 y su código (o la mayor parte) está disponible para poder descargarlo, examinarlo o incluso realizar aportes y modificaciones útiles para los demás usuarios.

Se ha publicado en GitHub el código fuente del sistema operativo MS-DOS para poder examinarlo, este sistema operativo lo constituye un núcleo básico en el que solo se puede ejecutar una instrucción a la vez, si eres un curioso y te gustaría saber como está

programado un sistema operativo en el que no existe el multiusuario, puedes descargar su código fuente y examinarlo o incluso modificarlo a tu gusto para hacerlo funcionar en un PC.

¿Para qué sirve?

El Kernel o núcleo de un sistema operativo sirve para administrar los recursos de hardware solicitados por los diferentes elementos de software y hacer de intermediario decidiendo a qué y cuando se concede este acceso evitando así sobrecarga del sistema, recursos innecesarios y acceso a software malicioso al propio Kernel y llegar a poder controlar así todo el sistema. De este modo el Kernel sirve como elemento de seguridad teniendo que pasar por varias capas antes de poder tener acceso, además tiene que distribuir los recursos de manera eficiente y ordenada para que el Hardware trabaje junto al Software de la mejor manera posible.

Aunque usualmente relacionamos un Kernel o un núcleo del sistema operativo a un PC, también está presente y sirve para hacer funcionar todos los computadores que podemos encontrar hoy en día, como por ejemplo un ordenador de a bordo de un coche o un barco, una raspberry PI que ejecuta una versión adaptada de Linux Debian o los dispositivos móviles con Android e iOS, que también disponen de un Kernel basado en Linux / Unix.

Tiene también como trabajo conceder acceso a todos los periféricos que tengamos conectados e interactuar con el software que los solicite, aunque no sean los usuales con los que trabajamos. Por ejemplo si ocasionalmente conectamos un móvil para usarlo como webcam con DroidCam, este Kernel se encarga de conceder los permisos necesarios al software para gestionar y poder tener la imagen y el audio para poder usarla en algún software de videoconferencia o reuniones si por ejemplo teletrabajar desde casa o cualquier otro lugar.

El Kernel es el encargado de hacer funcionar básicamente todo, tiene que ser capaz de arrancar, por ejemplo, un PC desde que lo encendemos hasta que vemos visible el escritorio, todo esto comunicándose con los elementos hardware que dispone el PC y que

también son necesarios para hacerlo funcionar, una vez que tengamos el escritorio deberá ser capaz de hacer funcionar los programas que nosotros queramos abrir y hacerlos funcionar en nuestro PC.

INTERRUPCIONES

En el contexto de la informática, una interrupción (del inglés *interrupt request*, en español «petición de interrupción») es una señal recibida por el procesador de una computadora, que indica que debe «interrumpir» el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Una interrupción es una suspensión temporal de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa, sino que pertenece al sistema operativo o al BIOS. Una vez finalizada dicha subrutina, se reanuda la ejecución del programa.

Las interrupciones son generadas por los dispositivos periféricos habilitando una señal del CPU (llamada IRQ del inglés "interrupt request") para solicitar atención del mismo. Por ejemplo. Cuando un disco duro completa una lectura solicita atención al igual que cada vez que se presiona una tecla o se mueve el ratón.

La primera técnica que se empleó para esto fue el *polling*, que consistía en que el propio procesador se encargará de sondear los dispositivos periféricos cada cierto tiempo para averiguar si tenía pendiente alguna comunicación para él. Este método presentaba el inconveniente de ser muy ineficiente, ya que el procesador consumía constantemente tiempo y recursos en realizar estas instrucciones de sondeo.

El mecanismo de interrupciones fue la solución que permitió al procesador desentenderse de esta problemática, y delegar en el dispositivo periférico la responsabilidad de comunicarse con él cuando lo necesitara. El procesador, en este caso, no sondea a

ningún dispositivo, sino que queda a la espera de que estos le avisen (le "interrumpan") cuando tengan algo que comunicarle (ya sea un evento, una transferencia de información, una condición de error, etc.).

Funcionamiento del Mecanismo

Todos los dispositivos que deseen comunicarse con el procesador por medio de interrupciones deben tener asignada una línea única capaz de avisar al CPU cuando le requiere para realizar una operación. Esta línea se denomina *IRQ*.

Las IRQ son líneas que llegan al controlador de interrupciones, un componente de hardware dedicado a la gestión de las interrupciones, y que puede estar integrado en el procesador principal o ser un circuito separado conectado al mismo. El controlador de interrupciones debe ser capaz de habilitar o inhibir las líneas de interrupción y establecer prioridades entre las mismas. Cuando varias líneas de petición de interrupción se activan a la vez, el controlador de interrupciones utilizará estas prioridades para escoger la interrupción sobre la que informará al procesador principal. También puede darse el caso de que una rutina de tratamiento de interrupción sea interrumpida para realizar otra rutina de tratamiento de una interrupción de mayor prioridad a la que se estaba ejecutando; aunque hay interrupciones que no se pueden deshabilitar (conocidas como interrupciones no enmascarables o NMI).

Un procesador principal que no tenga un controlador de interrupciones integrado, suele tener una única línea de interrupción llamada habitualmente INT. Esta línea es activada por el controlador de interrupciones cuando tiene una interrupción que servir. Al activarse esta línea, el procesador consulta los registros del controlador de interrupciones para averiguar cual IRQ hay que atender. A partir del número de IRQ busca en la tabla de vectores de interrupción la dirección de la rutina a la que debe llamar para atender la petición del dispositivo asociado a dicha IRQ.

Procesamiento

1. Terminar la ejecución de la instrucción máquina en curso.
2. Salvar el estado del procesador (valores de registros y flags) y el valor del contador de programa, IP, en la pila, de manera que en la CPU, al terminar el proceso de interrupción, pueda seguir ejecutando el programa a partir de la última instrucción.
3. La CPU salta a la dirección donde está almacenada la rutina de servicio de interrupción (Interrupt Service Routine, o abreviado ISR) y ejecuta esa rutina que tiene como objetivo atender al dispositivo que generó la interrupción.
4. Una vez que la rutina de la interrupción termina, el procesador restaura el estado que había guardado en la pila en el paso 2 y retorna al programa que se estaba usando anteriormente.

Mecanismo y líneas de petición de interrupción

El bus de control de la placa base dispone de líneas específicas para el sistema de interrupciones. Un PC típico dispone en su placa base de un controlador de interrupciones 8259 de Intel o de un circuito integrado análogo. Este dispositivo electrónico dispone de hasta 16 líneas IRQ, numeradas desde el 00 hasta el 15. En las nuevas placas base este circuito está integrado junto con el resto del *chipset* y permite hasta 24 interrupciones.

En el IBM PC y XT existían 8 líneas de petición de interrupción manejadas por el controlador de interrupciones Intel 8259. Estas líneas están numeradas del 0 al 7, las dos primeras están asignadas al *timer tick* del temporizador Intel 8253, y al teclado. Solo quedaban 6 líneas para otros dispositivos, que aparecen como tales en el bus de control (IRQ2 - IRQ7). A partir del modelo AT se añadieron otras 8 líneas, numeradas del 8 al 15, mediante un segundo controlador de interrupciones (PIC), aunque la tecnología empleada exigió colgarlo de la línea IRQ2 del primero, de forma que esta línea se dedica a atender las interrupciones del segundo controlador a través de la línea 9 de este último, y la línea 8 se dedicó al reloj de tiempo real, un dispositivo que no existía en los modelos XT.

Aunque internamente se manejan 16 líneas, no todas tienen contacto en los zócalos del bus externo (son las marcadas con asterisco en la tabla que sigue). La razón de esta ausencia en los zócalos de conexión es que son de asignación fija, y solo son usadas por ciertos dispositivos instalados en la propia placa base. En concreto la línea NMI está asignada al mecanismo de control de paridad de la memoria, la línea 0 está asignada al cronómetro del sistema y la línea 1 al chip que controla el teclado (dispositivos que pueden requerir atención urgente por parte del procesador). Es costumbre denominar IRQx a las que tienen prolongación en el bus.

Teóricamente las restantes líneas podrían ser asignadas a cualquier nuevo dispositivo, pero en la práctica algunas están reservadas a dispositivos estándar. Por ejemplo, IRQ3 está casi siempre asignado al puerto serie COM2 y el IRQ4 al COM1; IRQ6 al controlador estándar de disquetes y IRQ7 al puerto de impresora LPT1. La tabla 1 muestra las asignaciones clásicas para el XT y el AT.

En sistemas más modernos utilizan la arquitectura APIC de Intel con 24 líneas y 8 extra para enrutar las interrupciones PCI.

Cuando se instala un dispositivo de entrada o de salida que puede necesitar muchísima atención del procesador, debe asignársele una IRQ adecuada. Dicho en otras palabras, cuando un dispositivo periférico requiera atención, debe enviar una señal en la línea *IRQ* especificada. Inicialmente esta asignación se efectuaba de forma manual y automática, por medio de puentes (*jumpers*) en la placa o dispositivo móvil, pero actualmente esta selección puede hacerse mediante software.

Memoria Compartida

En informática, la memoria compartida es aquel tipo de memoria que puede ser accedida por múltiples programas, ya sea para comunicarse entre ellos o para evitar copias redundantes. La memoria compartida es un modo eficaz de pasar datos entre aplicaciones.

Dependiendo del contexto, los programas pueden ejecutarse en un mismo procesador o en procesadores separados. La memoria usada entre dos hilos de ejecución dentro de un mismo programa se conoce también como memoria compartida.

Software

En el software, memoria compartida puede referirse a:

- Un método de comunicación entre procesos, por ejemplo: el intercambio de datos entre dos programas ejecutándose al mismo tiempo. Uno de los procesos creará un área en RAM a la que el otro pueda acceder, o
- Un método para conservar espacio en la memoria, usando mapeos virtuales o bien soporte explícito del programa en cuestión, dirigiendo los accesos a una sola instancia de datos que normalmente serían duplicados. Comúnmente destinado para bibliotecas de enlace dinámico dinámicas y el espacio de usuario (XIP, "execute in place").

Dado que ambos procesos pueden acceder al área de memoria compartida como memoria de trabajo regular, esta es una forma de comunicación veloz (al contrario de otros mecanismos de comunicación entre procesos como tuberías nombradas, socket de dominio UNIX o CORBA. En cambio, este sistema es menos potente, si, por ejemplo, los procesos que se comunican deben ejecutarse en la misma máquina (en cuanto a otros métodos de comunicación entre procesos, solo los sockets del Internet Domain (no los sockets de UNIX), pueden usar una red de computadoras). Esto se debe a que se requiere mayor atención y cuidado si los procesos que comparten memoria corren en CPUs separadas y la arquitectura subyacente no soporta coherencia de caché.

La comunicación entre procesos por memoria compartida se usa en UNIX, para transferir imágenes entre una aplicación y un XServer, o en la biblioteca COM de Windows dentro del objeto *IStream* devuelto por la función *CoMarshalInterThreadInterfaceInStream*.

Las bibliotecas de enlace dinámico se copian una sola vez en la memoria y son "mapeadas" para múltiples procesos. Sólo las páginas que están personalizadas se duplican, normalmente con un mecanismo conocido como *copy-on-write* que de manera transparente copia la página cuando se intenta escribir en ésta, y después permite que la escritura se realice en la copia privada.

Hardware

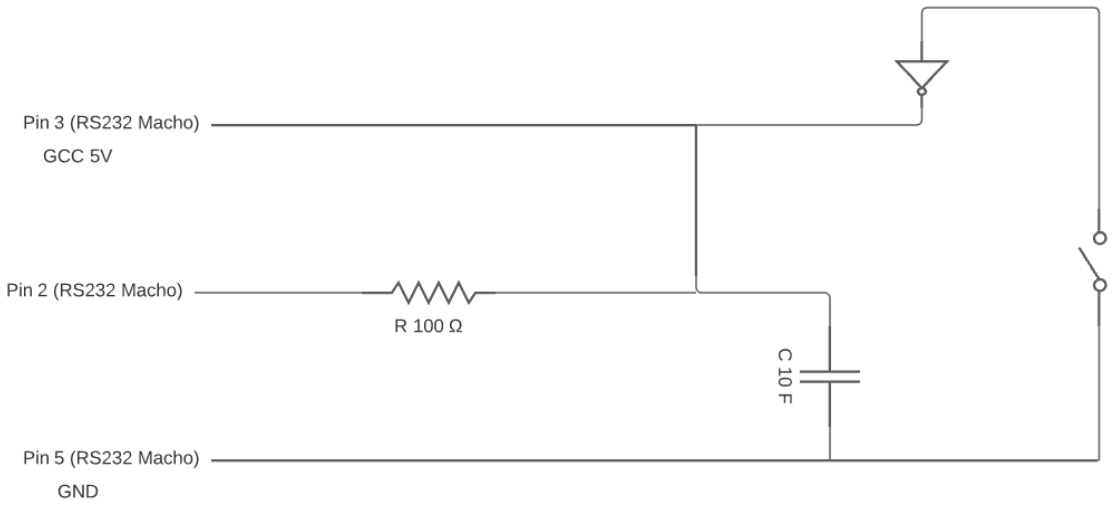
Se refiere a la situación dónde cada nodo (computadora) comparte sus tablas de páginas, su memoria virtual y todo lo que habita en la memoria RAM con otros nodos en la misma red de interconexión. Un ejemplo de memoria compartida es DSM.

En otras palabras, una computadora convencional tiene una memoria RAM y un CPU (entre otras partes de hardware), las cuales están comunicadas de manera local. Cuando se habla de multicomputadoras es necesario entenderlas como un conjunto de máquinas convencionales trabajando coordinadamente con el objetivo de compartir recursos de software y hardware para tareas muy específicas, como por ejemplo la investigación. El hecho de que cada CPU sólo pueda acceder a la RAM local no beneficia mucho y agrega complejidad a los programadores de multicomputadoras, con la compartición de memoria se mitiga esta complejidad ya que todas las CPUs pueden acceder a la misma RAM.

PRÁCTICA

Serial

La conexión de RS232, normalmente está controlada por el dispositivo ttyUSB0 o ttyUSB1, pero esta no genera interrupciones directamente, por más que generemos una conexión, y pase energía no genera interrupción, por lo que hay que correr un tipo de serial port, para que escuche el dispositivo y genere la interrupción. Primero vamos a ver la conexión creada:



Pero como antes decíamos, este conector no genera interrupciones de por sí solo, y el módulo que vamos a utilizar, y hablar en el siguiente momento, solo escucha interrupciones, por lo que hay que generar estas interrupciones, en una especie de listener del puerto en especifico. Primero utilice lo que fue un archivo que encontré, que mandaba señal y recibía señal del mismo, y lo mostraba en la pantalla, pero cambié lo que fue el código, para que fuera un while, que escuche a este dispositivo, pero este dispositivo en vez de conectarse en el pin 3, que es el que manda info, lo cambié al pin 4, y ya no manda info, solo recibe información, lo que es 1 bit, y esto genera una interrupción, en el caso de la

máquina virtual, se ejecutó en el IRQ 18. Aquí mostraremos la parte del código que se cambió.

Por lo que se le da make, y este generará un ejecutable, llamado **send_receive**, donde hay que ejecutarlo con la siguiente instrucción: *sudo ./send_receive /dev/ttyUSB0*, para ejecutarlo.

```
58  int main (int argc, char* argv[])
59  {
60    int serial_fd;           //-- Descripción del puerto serial
61    char data[CMD_LEN+1];    //-- La data recivida
62
63    //-- Verificar si el nombre del dispositivo esta disponible
64    if (argc<2) {
65      printf ("No serial device name is given\n");
66      exit(0);
67    }
68
69    //-- Abrir el puerto serial
70    //-- La rapidez esta configurada en 9600 baudios
71    serial_fd=serial_open(argv[1],B9600);
72
73    //-- Chequeo de errores
74    if (serial_fd==-1) {
75      printf ("Error opening the serial device: %s\n",argv[1]);
76      perror("OPEN");
77      exit(0);
78    }
79  while(1==1){
80
81    //-- Esperar por la data a recibir
82    int n;
83    n=serial_read(serial_fd,data,CMD_LEN,TIMEOUT);
84
85    if (n>0) {
86    //-- Mostrar la data recibida
87      printf ("(%d bytes)\n",n);
88      usleep(TIMEOUT);
89    }
90   }
91
92    //-- Cerrar el puerto serial
93    serial_close(serial_fd);
94
95    return 0;
96  }
97
```

Como visualizamos, en el while es el ciclo, y el serial_read es el que lee la información.

Módulo

Este es el driver, o módulo, el que se va a instalar con el comando **insmod**, el nombre del módulo es driver, por lo que el módulo va a tener el nombre de **driver.ko**, lo cual lo insertamos en el módulo, y se agrega a los módulos, en este archivo, solo se cambia el IRQ que se esté utilizando en la computadora, y en mi caso pues se cambió el IRQ a 18, ya que este es el que usa como lo mencioné anteriormente, al finalizar este cambio, se realiza el make, y se ejecuta el insmod driver.ko, y este ya está debidamente agregado y escuchando las interrupciones.

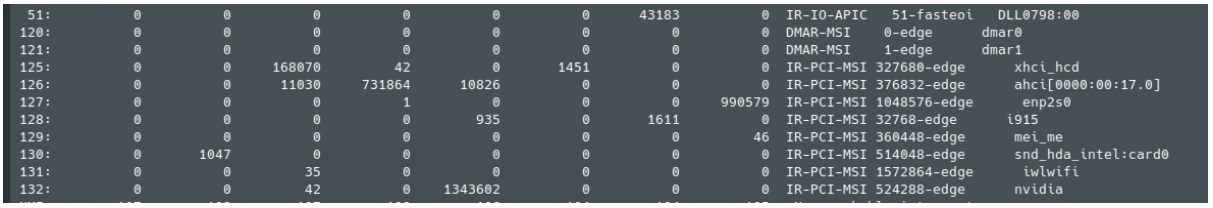
El problema del módulo, es que no se puede compartir memoria directamente, ya que no es aconsejable, y puede ocasionar errores en el sistema, por lo que este solo manda una bandera, que seguidamente nos vamos a dar cuenta, que es el sender el que lo recibe. Aquí un pequeño ejemplo del cambió que se realizó.

```
17  #define SIGETX 44
18
19  #define REG_CURRENT_TASK _IOW('a','a',int32_t*)
20
21  #define IRQ_NO 18                       //IRQ 18 es el de mi commputadora! :D
22
23  /* Signaling to Application */
24  static struct task_struct *task = NULL;
25  static int signum = 0;
26
27  int32_t value = 0;
28
29  dev_t dev = 0;
30  static struct class *dev_class;
31  static struct cdev etx_cdev;
32
33  static int __init etx_driver_init(void);
34  static void __exit etx_driver_exit(void);
35  static int etx_open(struct inode *inode, struct file *file);
36  static int etx_release(struct inode *inode, struct file *file);
37  static ssize_t etx_read(struct file *filp, char __user *buf, size_t len,loff_t * off);
38  static ssize_t etx_write(struct file *filp, const char *buf, size_t len, loff_t * off);
39  static long etx_ioctl(struct file *file, unsigned int cmd, unsigned long arg);
40
```

Donde podemos visualizar en la línea 21 el #define IRQ_NO 18, que este es el que se debe cambiar, por sí es diferente el irq.

¿Cómo saber qué IRQ es el que usa mi serial?

Esto es fácil, al correr el primer archivo **send_receive**, con el comando dicho, en la parte del serial, este ya empieza a escuchar interrupciones, por lo que nos vamos a una terminal y ejecutamos **watch -n1 cat /proc/interrupts**, con este comando vamos a ver las interrupciones, y a la izquierda van a ver lo que es el número de IRQ, por lo que cuando presionemos el botón, este IRQ va a subir, y así sabremos qué número es, al ver cuál es el que sube, también nos ayuda que debe ser uno que tenga un modulo de USB.



Sender

En este archivo, vamos a recibir la señal del módulo, y vamos a mandar a la memoría compartida lo que es el número de pulsaciones que hizo en un cierto tiempo, menor a dos segundos, el cuál el receptor va a estar verificando a cada cierto tiempo. Vamos a mostrar un poco del código hecho para la memoria compartida, y como se maneja con el sistema.

```
82      if(check){
83          int i;
84          void *shared_memory; //memoria de cuantas veces se ha precionado
85          void *shared_memory2; //memoria de fecha de la ultima vez que se presiono
86          int presionado; //numero de veces que se ha presionado
87          int shmid;
88          int bool1=0;
89          /* Obtener el tiempo de ahora */
90          end_t=clock();
91          //Obtener la memoria compartida
92          shmid=shmget((key_t)2222, 1024, 0666);
93          printf("Key of shared memory is %d\n",shmid);
94          if(shmid==-1){
95              shmid=shmget((key_t)2222, 1024, 0666|IPC_CREAT);
96              bool1=1;
97          }
98          if(bool1){ //primera vez que se presiona
99              shared_memory=shmat(shmid,NULL,0); //proceso adjunto al segmento de memoria compartida
100             presionado = 1;
101             char a[2] ;
102             *a= presionado+'0';
103             strcpy(shared_memory,a);
104             printf("You wrote : %s\n",(char *)shared_memory);
105         }else{ // cuando ya se haya presionado una vez.
106             shared_memory=shmat(shmid,NULL,0);
107             printf("Found: %s\n",(char *)shared_memory);
108             // Visualizar tiempo transcurrido
109             double sec = ((double) (end_t - start_t)) / CLOCKS_PER_SEC;
110             printf("Execution time = %f\n", sec);
111             
112             if(sec >= 2 || sec < 0){
113                 presionado = 1;
114             } else if (sec >= 0.1){
115                 if(strcmp(shared_memory, "1") == 0){
116                     presionado = 2;
117                 } else if (strcmp(shared_memory, "2") == 0){
118                     presionado = 3;
119                 } else {
120                     presionado = 1;
121                 }
122             } else {
123                 goto end;
124             }
125             char a[2] ;
126             *a= presionado+'0';
127             strcpy(shared_memory,a);
128             printf("You wrote : %s\n",(char *)shared_memory);
129 
130         }
131         end:
132         printf("end");
133         check=!check;
134         start_t = end_t;
135     }
```

Receiver

En este programa, lo único que revisamos es en la memoria compartida, las veces que se han presionado en cierto tiempo, esto yendo a la memoria compartida, donde vamos a revisar esto, y vamos a mostrar lo que está ingresado, de parte del sendero, aquí el código.

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>

int main()
{
    int i, b;
    void *shared_memory;
    int shmid;
    i=-1;
    while(1){
        //Memoria compartida con el id 2222
        shmid=shmget((key_t)2222, 1024, 0666);
        if(shmid==-1){
            if(i==-1){
                printf("El numero ha sido presionado 0 veces\n");
                i=0;
            }
        }else{
            shared_memory=shmat(shmid,NULL,0); //proceso adjunto al segmento de memoria compartida
            // compartir la información
            b = atoi((char *) shared_memory);
            if(i != b){
                if(b==1){
                    printf("El boton ha sido presionado %d vez\n",b);
                } else {
                    printf("El boton ha sido presionado %d veces\n", b);
                }
                i=b;
            }
        }
        usleep(100000);
    }
}

```

CONCLUSIONES

- Las interrupciones del kernel, son de varios dispositivos, y es interesante ver cómo estos se manejan, y cómo estos se ven visualizados en el mismo, si uno mira con el comando **watch** en la sección de ¿Cómo saber qué IRQ es el que se usa?, podemos visualizar como sube este irq, cuando se mueve el mouse o cuando clickea, y es muy interesante esto mismo.
- La memoria compartida es muy importante para compartir memoria entre diferentes aplicaciones, para que estas aplicaciones, puedan leer información que manda la otra aplicación, por lo que es interesante, lo que falla, es que esto no se puede utilizar en los módulos, por lo que no se puede hacer de una manera directa.
- El uso de assembler fue muy complicado, por lo que se prescindió de lo mismo, por que el uso de assembler se vuelve muy complicado, y extraño, y con muy poca información, por lo que se encuentra más información en c, aún así siendo complicado.

RECOMENDACIONES

- Hacer esta práctica primero en C, y tal vez luego pasar a assembler, ya que con el bajo conocimiento de assembler, básico por así decir, pues es muy complicado, por que información de manejar IRQS en assembler, es muy complicado, casi nula, y la única información es teórica, y casi no se encuentran codificaciones en assembler, para tomar de referencia.
- No manejar la memoria compartida en la parte del módulo, ya que puede provocar fallos, y errores con la computadora, ya que no es aconsejable hacerlo.
- El serial, en sí el send_receive, es importante, ya que el aparato no crea lo que es interrupciones por sí solo, por lo que es primordial, el tener este programa corriendo para que se escuchen las interrupciones.

BIBLIOGRAFÍA

- https://embetronicx.com/tutorials/linux/device-drivers/sending-signal-from-linux-device-driver-to-user-space/
- https://es.wikipedia.org/wiki/Memoria_compartida
- https://es.wikipedia.org/wiki/Interrupción
- https://www.geeknetic.es/Kernel/que-es-y-para-que-sirve
- https://www.monografias.com/docs112/gestion-interrupciones-sistemas-operativos/gestion-interrupciones-sistemas-operativos2.shtml
- https://static.lwn.net/images/pdf/LDD3/ch10.pdf
- https://0xax.gitbooks.io/linux-insides/content/Interrupts/linux-interrupts-8.html
- https://en.wikibooks.org/wiki/X86_Assembly/Programmable_Interrupt_Controller
- https://tldp.org/LDP/lkmpg/2.6/html/x1256.html
- https://www.sciencedirect.com/topics/computer-science/programmable-interrupt-controller
- https://developer.arm.com/documentation/dui0056/d/handling-processor-exceptions/interrupt-handlers/example-interrupt-handlers-in-assembly-language
- https://forums.sifive.com/t/beginner-trying-to-set-up-timer-irq-in-assembler-how-to-print-csrs-in-gdb/2764
- https://www.youtube.com/watch?v=GwENkX_6bVg
- https://stackoverflow.com/questions/34766167/simple-usb-button-using-c-sharp
- http://what-when-how.com/8051-microcontroller/programming-the-serial-communication-interrupt/
- https://what-when-how.com/8051-microcontroller/serial-port-programming-in-c/
- http://www.todopic.com.ar/foros/index.php?topic=25178.0
- https://w3.ual.es/~rguirado/so/practica5