INSTITUTO POLITECNICO NACIONAL
UNIDAD PROFESIONAL INTERDICIPLINARIA EN INGENIERIA Y TECNOLOGIAS AVANZAS

Sistemas Operativos en Tiempo Real

Practica:
Compilacion de un SOTR con una aplicacion de alarma

Profr.: M. Lamberto Maza Casas

Alumno: Plata Colín César Iván

01-Diciembre-2019

Introduccion

Cuando se trabaja con sistemas operativos en tiempo real es necesario ser capaz de hacer aplicaciones que corran bajo el ambiente del SO correspondiente. Por consiguiente se debe enfrentar a las limitantes del sistema. En estas hojas se explica como se hizo una aplicacion de alarma para el sistema operativo MaRTE OS el cual es de tiempo real.

Prerrequisitos

Como primer paso de debe descargar las fuentes MaRTE OS y el compilador ADA, instalar estas fuentes, y despues instalar MaRTE OS. Para lo anterior se utilizo como base el documento “Guia para la instalcion del Sistema Operativo en Tiempo Real MaRTE OS” creado por Baldemar Martínez Morales y Jonathan Alexis Salas Soto. Debido a que el objetivo de esta practica no es instalar el sistema operativo en sí, resumiremos el documento en sencillos pasos:

1. Descargar las fuentes MaRTE y GNAT (compilador ADA core basado en la infrestructura de GCC) y crear el lugar donde estaran
   ```
   $ wget
   https://marte.unican.es/marte/marte_2.0_22Feb2017_src.tar.gz
   $ wget
   https://community.download.adacore.com/v1/845147a8c6ef6af29a6814
   4d6b3d228fd226268e?filename=gnat-gpl-2016-x86_64-linux-
   bin.tar.gz
   ```
2. Descomprimir las fuentes de MaRTE y ADAcore
   ```
   $ tar -xvf marte_2.0_22Feb2017_src.tar.gz
   $ tar -xvf gnat-gpl-2016-x86_64-linux-bin.tar.gz
   ```
3. Modificar el archivo .bashrc con las lineas
   ```
   export PATH=$HOME/myapps/gnat/bin:$PATH
   export PERL5LIB=$HOME/myapps/marte_2.0_22Feb2017_src.tar.gz
   export PATH=$PATH:$HOME/myapps/marte_2.0_22Feb2017/utils
   #export PATH=/opt/cross-pi-gcc/bin:$PATH
   ```
4. Dentro de la carpeta donde se descomprio GNAT instalar el compilador:
   ```
   $ ./doinstall
   ```
   y seguir las instrucciones, para despues cambiarnos a la carpeta de marte y ejecutar la instalacion de marte:
   ```
   $ ./minstall
   ```
5. Moverse a la carpeta “utils” de marte para seleccionar arquitectura para usar, compilar y construir marte, de hecho esta instrucción es sugerida por el instalador al termino de la instalcion.
   ```
   $ msetcurrentarch x86 i386 && mkrtsmarteuc && mkmarte
   ```

6. Una vez terminado el inciso anterior podemos salir de la carpera util y ubicarnos en la carpeta examples para comenzar a crear programas para marte y compilarlos desde ahi.

Desarrollo

Como primer punto debemos crear el programa. Desafortunadamente muchas de las funciones que comunmente usamos como clock() y sleep() no se encuentran disponibles en las librerias de MaRTE OS. Por esto usaremos las funciones de la librería time.h.

Dentro de la carpeta examples de MaRTE OS escribimos:

```
$ vim alarma.c
```

```c
#include <stdio.h>
#include <time.h>

int main (){
        int h,m,s,ts;
        struct timespec acT, psT;

        printf("Escriba el tiempo que se va temporizar en el formato \n ho
ras minutos segundos \n");
        scanf("%d %d %d",&h,&m,&s);
        /* calculo de cantidad de segundos a contar */

        ts=(h*60*60)+(m*60)+s;
        printf("\n Se contaran %d segundos\n",ts);

        clock_gettime(CLOCK_REALTIME,&acT);
        psT.tv_sec = acT.tv_sec + ts;
        psT.tv_nsec = 0;

        while (acT.tv_sec < psT.tv_sec){

                clock_gettime(CLOCK_REALTIME,&acT);
        }

        printf("\n**************************************************\n");
        printf(" Han pasado %d segundos",ts);
        printf("\n**************************************************\n");

        return 0;
}
```

El programa es sencillo de explicar, en las primeras lineas se crean las variables, y las estructuras que usara el programa, despues se le indica al usuario que introdusca el tiempo que se temporizara en un formato determinado (hrs min seg), despues se calcula el numero de segundos que se temporizaran. Para finalizar se guarda el tiempo

actual en acT y dentro de la estructura while se compara con el tiempo que deberiamos tener cuando la alarma se deba activar. Al final se escribe que el timepo se ha terminado.

Para compilar marte con el programa que acabamos de crear usamos:

```
$ make alarma.exe
```

```
aufasder@aufasder:~/myapps/marte/examples$ make alarma.exe

>> Compiling alarma.exe: Use of uninitialized value $GNAT_LIBS_PATH{"rpi"} in co
ncatenation (.) or string at /home/aufasder/myapps/marte/utils/globals.pl line 1
6.
  [OK]
aufasder@aufasder:~/myapps/marte/examples$
```

Se ha generado un archivo alarma.exe el cual corresponde al kernel de marte con el programa alarma el cual se ejecutara inmediatamente despues del proceso de boot del sistema. Esto se puede comprobar usando qemu, escribimos:

```
$ qemu-system-i386 -kernel alarma.exe
```

Inmediatamente se nos abrira el shell de qemu, y podemos notar que nuestro programa se esta ejecutando correctamente.

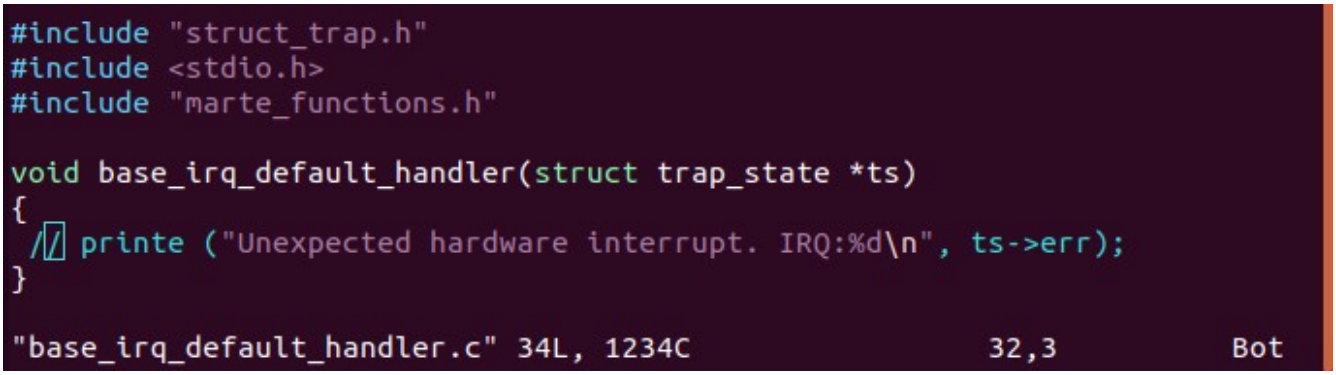
```
                    -=  M a R T E   O S  =-
                        V2.0 2017-02-22
            Copyright (C) Universidad de Cantabria, SPAIN
TLSF 2.3.2 dynamic memory pool: 131514368 bytes
Devices initialization...
   Major Number 1 (stdin)  Keyboard        ...OK
   Major Number 2 (stdout) Text/Serial     ...OK
   Major Number 3 (stderr) Text/Serial     ...OK
Escriba el tiempo que se va temporizar en el formato
 horas minutos segundos
0 1 5

 Se contaran 65 segundos

**************************************************
 Han pasado 65 segundos
**************************************************
 _exit(0) called; rebooting...
Press a key to reboot
```

Se podria pensar que ya estamos listos para correr el programa sobre hardware pero tal como esta ahora el SOTR nos mandara una señal de

solicitud de interrupcion y un warning que nos impedira ver nuestro programa en ejecucion. Ahora en lugar de atender la interrupcion que nos solicita simplemente impediremos que el mensaje se mueste modificando 2 archivos de manejo de interrupciones:

```
$ vim marte/linux_arch/hwi/boot/base_irq_default_handler.c
$ vim marte/x86_arch/hwi/boot/base_irq_default_handler.c
```

En ambos archivos (en teoria deberia bastar con x86_arch pues es la arquitectura seleccionada), comentamos la linea de printe.



Nota: La razon por la que se piden todos los valores de tiempo en una sola linea en nuestro programa de alarma es porque incluso despues de evitar que se muestre el aviso de "Unexpected interrupt request IRQ:7" el error se sigue presentando aunque no se es consiente ello. Si se desea pedir un segundo dato en otro punto del programa el propio error de la interrupcion te impedira excribir el siguiente dato.

Ahora debemos regresar a la carpeta utils para reconstruir marte. Ya dentro de la carpteta utils escribimos

```
$ mkrtsmarteuc && mkmarte
```

Despues regresamos a la carpeta de examples y creamos el kernel que copiaremos a nuestra carpeta donde generamos el archivo iso.

```
$ cd ../examples
$ make clean && make alarma.exe
```

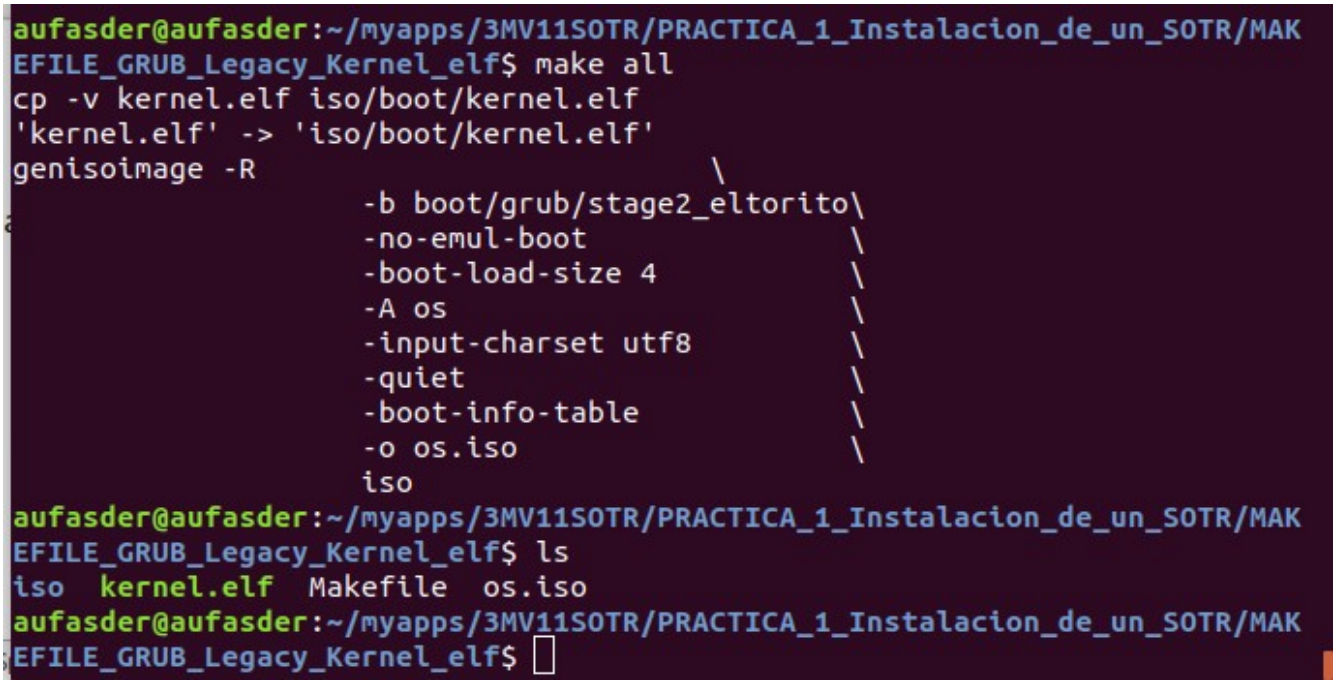
Despues copiaremos nuestro kernel con el nombre de kernel.elf a la carpeta de "MAKEFILE . . . elf" y cambiamos a esa carpeta para generar nuestro archivo iso.

```
$ cp alarma.exe
$HOME/myapps/3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf/kernel.elf
```

$ cd HOME/myapps/3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_kernel_elf/

$ make all

Una vez terminado tenemos el os.iso que necesitamos para instalarlo en una USB booteable



Como se ha hecho anteriormente con el programa LinuxLive USB se creo la usb booteable y esta es capaz de ejecutarse sobre hardware.

Conclusion

Respecto a los resultados fueron parciamente satifastorios, se pudo generar el programa y funciona en el entorno MaRTE OS pero funciona parciamente, pues no tenemos informacion sobre la interrupcion que aqueja a este programa cuando se ejecuta, ademas no sabemos como atender interrupciones de manera correcta pues en clase intentamos modificar la atencion a interrupciones sin éxito. Por el resto de la practica fue muy ilustrativa, se utilizaron funcion nativas de marte y librerias que de otra manera no nos veriamos en la necesidad de utilizarlas pero que contribuyen de manera enorme en el conocimiento de los sistemas operativos y su relacion con el hardware tal como es el MONOLITIC_CLOCK.