UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

Profesor: Ing. Agustín Martina

TRABAJO PRÁCTICO 4

Sistemas Operativos de Tiempo Real

Mauricio Aguilar

22 de Noviembre de 2016

Contenido

Introducción

Un sistema operativo de tiempo real es un sistema operativo que ha sido desarrollado para ejecutar aplicaciones de tiempo real. Es un entorno en el que no se le da importancia al usuario sino a determinados **procesos**. Como tal, se le exige corrección en sus **respuestas** bajo ciertas restricciones de tiempo. Si no las respeta, se dirá que el sistema ha fallado. Para garantizar el comportamiento correcto en el tiempo requerido se necesita que el sistema sea **predecible**. Respecto de un sistema tradicional, la diferencia radica en que proporciona mayor **prioridad** a los elementos de control y procesamiento que son utilizados para ejecutar los procesos o tareas.

El sistema operativo de tiempo real debe:

- Ser multitarea y permisible
- Poder asignar prioridades a las tareas
- Proporcionar medios de comunicación y sincronización entre tareas
- Poder evitar el problema de inversión de prioridades
- Tener un comportamiento temporal conocido.

Los SOTR pueden clasificarse como **blandos** cuando estos pueden tolerar un excesos en los tiempos de respuestas garantizando que las tareas críticas se ejecuten a tiempo, o como **duros** cuando la respuesta fuera de término no tiene valor alguno.

Descripción del problema

En el presente trabajo nos proponemos:

1) Instalación y configuración del kernel con soporte de tiempo real

2) Instalación y configuración de las API para desarrollar programas que aprovechen el sistema de tiempo real.

3) Ejecución del programa y medición de parámetros.

Se proveen tres programas para realizar las pruebas de respuesta del sistema. Para las pruebas se debe ejecutar un programa productor y un programa consumidor. El programa productor realiza un muestreo cada 12 microsegundos de una función seno que varía en el tiempo. Los puntos muestreados se escriben en una FIFO. El programa consumidor lee la FIFO y muestra la información en pantalla.

Existe un programa productor para un entorno tradicional (“process.c”), donde se utiliza una FIFO común, como las utilizadas en IPC. El otro programa productor para entornos de tiempo real (“rt_process.c”), utiliza mecanismos de IPC provistos por el entorno de tiempo real. El programa consumidor (“scope.c”) lee la FIFO indicada y muestra la información en pantalla.

R1) Describa el funcionamiento del programa “rt_process.c”.

R2) Describa la funciones de la API de RTAI utilizadas.

R3) Ejecute varias veces los programas en un entorno tradicional, y almacene los datos en un archivo, en ambos casos sobrecargando el sistema.

Ejecute varias veces los programas en un entorno de tiempo real, y almacene los datos en un archivo. Para ello debe estar cargado el módulo “rtai_fifos”.

Obtenga gráficos con los puntos muestreados en el entorno tradicional, utilizando distinta cantidad de puntos y distintas bases de tiempo (Sugerencia: utilizar “gnuplot”).

Obtenga gráficos con los puntos muestreados en el entorno de tiempo real, utilizando distinta cantidad de puntos y distintas bases de tiempo (Sugerencia: utilizar “gnuplot”).

Emita conclusiones a partir de la comparación de gráficos en ambos entornos.

R4) Explique porque debe ejecutarse de modo nativo y no en una maquina virtual.

R5) Investigue acerca de cuáles son los Sistemas Operativos de tiempo real más utilizados en la actualidad y compare sus ventajas/desventajas y sus ámbitos de aplicación.

Requerimientos y tareas

Requerimientos Funcionales:

- Un proceso productor deberá generar valores correspondientes a la función sin(x) en función del tiempo, con una aplicación tradicional en un sistema tradicional.
- Un proceso productor deberá generar valores correspondientes a la función sin(x) en función del tiempo, con una aplicación de tiempo real en un sistema de tiempo real.
- Un proceso consumidor deberá guardar los datos del sistema tradicional en un archivo llamado data.txt
- Un proceso consumidor deberá guardar los datos del sistema de tiempo real en un archivo llamado rt_data.txt

Requerimientos No Funcionales:

- El sistema tradicional debe sobrecargarse.
- El sistema de tiempo real debe arrojar tiempo de latencia cero cuando ningún proceso se esta ejecutando.
- El sistema de tiempo real no puede generar cortes en el gráfico de la senoidal.

Diseño y documentación

Para esta implementación se utilizó:

- El sistema operativo Debian Wheezy 7.1.
- La API de RTAI para procesos de tiempo real
- El test de latencia de RTAI
- El graficador de Linux, GNUplot.

.

Implementación

Proceso de montado del sistema paso a paso:

1. Se descargó debian Wheezy desde https://www.debian.org/releases/ optando por la versión Debian Wheezy Net Installer (http://cdimage.debian.org/mirror/cdimage/archive/7.9.0/i386/iso-cd/debian-7.9.0-i386-netinst.iso).

2. Se instaló de forma nativa en una partición de 17GB.

3. Se realizó una actualización de la lista de repositorios y luego una actualización

   a) sudo apt-get update

   b) sudo apt-get dist-upgrade

4. Se realiza la siguiente serie de pasos para obtener el archivo de configuración .config para la compilación del kernel.

   a) Instalar el kernel PreemptRT y sus módulos (dependiendo que versión se haya descargado)

      - sudo apt-get install linux-image-rt-amd64
      - sudo apt-get install linux-image-rt-686-pae (para este caso)

   b) Reiniciar el sistema y asegurarse de que se booteo el kernel correcto. Verificando que diga PREEMPT y RT.

      - uname -v

   c) Agregar la LinuxCNC Archive Signing Key al aptitude

      - sudo apt-key adv --keyserver hkp://keys.gnupg.net --recv-key 3cb9fd148f374fef

   d) Agregar a la lista de repositorios LinuxCNC

      - sudo add-apt-repository "deb http://linuxcnc.org/ wheezy base 2.7-uspace"

e) Realizar una actualización de la lista de repositorios para cargar LinuxCNC

- sudo apt-get update

f) Instalar uspace

- sudo apt-get install linuxcnc-uspace

5. Se llevó a cabo la instalación de las herramientas de compilación (gcc):

- sudo apt-get install build-essential

6. Se llevó a cabo una búsqueda en los fuentes de las **librerías RTAI**, una versión que cuente con un parche para una versión de kernel próxima a la de la distribución Debian instalada. Se decidió utilizar la versión 4.0 la cual puede descargarse del siguiente enlace.: https://www.rtai.org/userfiles/downloads/RTAI/rtai-4.0.tar.bz2

7. Se descargo el código fuente de la versión de **kernel de linux** 3.4.6 desde la página oficial: http://www.kernel.org/pub/linux/kernel/

8. Ambos fuentes fueron descomprimidos bajo la ruta /usr/src.

9. Se aplicó el parche RTAI a los fuentes del kernel a compilar. Esto se llevó a cabo posicionado sobre la carpeta que contiene los fuentes del kernel y mediante el comando:

- patch -p1 -b < /usr/src/rtai-4.0/base/arch/x86/patches/hal-linux-3.4.6-x86-4.patch

10. Instalar las librerías ncurses para hacer uso de menuconfig

- apt-get install libncurses5-dev

11. Configurar la compilación del kernel con:

- make menuconfig CC=/usr/bin/gcc CXX=/usr/bin/g++
- Aquí puede optarse por:
  - Configurar manualmente la compilación del kernel.

- Descargar el archivo de configuración .config desde internet (páginas de LinucCNC)
- Generar el archivo de configuración como se hizo en el paso 4. Se optó por esta opción. Se copió la configuración del kernel de la distribución Debian en ejecución a la carpeta que contiene los fuentes del kernel con el parche de RTAI a compilar:
  - cp /boot/config-xxx-generic .config (forma genérica posicionado dentro de /usr/src/rtai-4.0/
  - cp /boot/config-3.4.6-rtai /usr/src/rtai-4.0/.config
  - Se carga el archivo .config con la opción Load… Se sale y se guarda.

12. Se instala fakeroot para compilar el kernel

- apt-get install kernel-package fakeroot

13. Se llevó a cabo la compilación mediante el comando:

- make-kpkg --append-to-version -rtai --revision 1 --initrd --config menuconfig kernel_image kernel_headers

14. Finalizada la compilación se generan paquetes DEB para la instalación, en la ruta /usr/src. Se procede a la instalación del kernel compilado y sus headers mediante el comando:

a) cd /usr/src/

b) sudo dpkg -i *.deb

15. Se inicia el sistema con el kernel con soporte RTAI

16. Se lleva a cabo la compilación de las librerías RTAI, para ello desde la carpeta que contiene los fuentes de las librerías RTAI se ejecuta:

a) cd /usr/src/rtai-4.0/

b) mkdir build

c) cd build

d) Aplicar permisos a estos archivos y a todos los que tenga el acceso denegado.

   i. chmod 777 /usr/src/rtai-4.0/base/config/autoconf/config.guess

   ii. chmod 777 /usr/src/rtai-4.0/base/config/autoconf/arch2host.sh

   iii. chmod 777 /usr/src/rtai-4.0/configure

17. Y configurar RTAI con

   a) make -f ../makefile CC=/usr/bin/gcc CXX=/usr/bin/g++
   Si todo salio bien se tiene que ver el menu de configuración de Rtai.

   b) Cambiar nombre a la carpeta con las fuentes del linux, de "Linux-3.4.6" a "linux"

   c) Configurar RTAI

      i. Asegurarse de que los directorios de instalación (/usr/realtime) y kernel (/usr/src/linux) estén configurados adecuadamente en el primer menú.

      ii. También se activó en el primer menú, la opción easter inlining.

      iii. Se seteó en 4 el numero de procesadores, de acuerdo a la pc.
      Machine (x86) ---> (4) Number of CPUs (SMP-only)

      iv. Desactivar Comedi en Add-Ons
      Add-ons ---> [ ] Real Time COMEDI support in user space

18. Setear las variables de entorno:

   a) export PATH=$PATH:/usr/realtime/bin:/usr/realtime/include

   b) export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/realtime/lib

19. Cada vez que se vaya a ejecutar un programa de tiempo real deben insertarse los módulos:

```
sudo insmod /usr/realtime/modules/rtai_hal.ko
sudo insmod /usr/realtime/modules/rtai_lxrt.ko
sudo insmod /usr/realtime/modules/rtai_fifos.ko
```

20. Se realizó una prueba de latencia con ningún proceso ejecutando, la cual debe ser cero (columna de la derecha):

- /usr/realtime/testsuite/kern/latency/run

```
* Type ^C to stop this application.

## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot

RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH|    lat min|    ovl min|    lat avg|    lat max|    ovl max|    overruns
RTD|       -1792|       -1792|       -1360|       41712|       41712|          0
RTD|       -1792|       -1792|       -1430|       37572|       41712|          0
```

21. Se realizó la compilación de los programas productores y consumidores con los siguientes comandos:

```
gcc -o rtp rt_process.c -pthread -lm -I /usr/realtime/include/
gcc -o rts rt_scope.c -pthread -lm -I /usr/realtime/include/
gcc -o p process.c -lm -lrt
gcc -o s scope.c -lm -lrt
```

23. Se ejecutó los programas tradicionales y de tiempo real en sus respectivos sistemas, con los comandos:

- ./rtp
  ./rts

- ./p
  ./s

- Siempre se debe ejecutar primero el productor y luego el scope, ya que sino este último intentará acceder a una fifo que aún no existe. El scope debe ser el primero en detenerse ya que si se detiene primero el productor, el scope quedara copiando el ultimo valor en el archivo.

Resultados y Respuestas

R1) Describa el funcionamiento del programa “rt_process.c”.

R2) Describa la funciones de la API de RTAI utilizadas.

Main_Task = rt_task_init_schmod Esta máscara sirve para controlar cual procesador usa cada tarea. Para el primer asignamiento, el uso de un solo procesador es obligatorio.

nam2num convierte un string en un long, sirve para name de tarea para la máscara, y permite usar letras fáciles de recordar en vez de numeros.

mlockall(MCL_CURRENT | MCL_FUTURE);

Esta función se usa para cumplir con el requisito de que la memoria para un proceso esté bloqueada (libre de paginación).

if (!(Main_Task)) si no se puede inicializar la tarea, imprime error y sale.

rt_is_hard_timer_running()) se utiliza para saber si el temporizador de tiempo real duroe está corriendo.

nano2count convierte nanosegundos en unidades de cuenta interna.

rt_set_periodic_mode() setea el modo periódico para el temporizador.

start_rt_timer(0) inicia el temporizador con período cero.

pthread_create crea un hilo que ejecuta la función main_loop.

getchar(); crea una pausa

rt_task_delete(Main_Task); borra la tarea principal

Cuando se ejecuta la función main_loop, nuevamente se crea una máscara, se bloquea la memoria, se obtiene el tiempo actual con **rt_get_time()** y se le suma 100 intervalos, cuyo tiempo pasa a ser el tiempo de inicio de la tarea periódica generada por **rt_task_make_periodic()**. Se convierte a la tarea de tiempo real en una dura, ya que por defecto es blanda, con la función **rt_make_hard_real_time().**

Se elimina el fifo, si es que existe de una ejecución anterior con **unlink**. Se lo vuelve a crear con **mkfifo()**. Se lo abre con **open()**.

Es entonces recién cuando se ingresa en un loop que escribe pares de valores en el fifo, un contador y un valor de sin(x) en función del tiempo.

Entre cada loop, **rt_task_wait_period()** suspende la tarea que actualmente se está ejecutando en tiempo real hasta que el siguiente período sea alcanzado.

R3) Ejecute varias veces los programas en un entorno tradicional, y almacene los datos en un archivo, en ambos casos sobrecargando el sistema.

Ejecute varias veces los programas en un entorno de tiempo real, y almacene los datos en un archivo. Para ello debe estar cargado el módulo “rtai_fifos”.

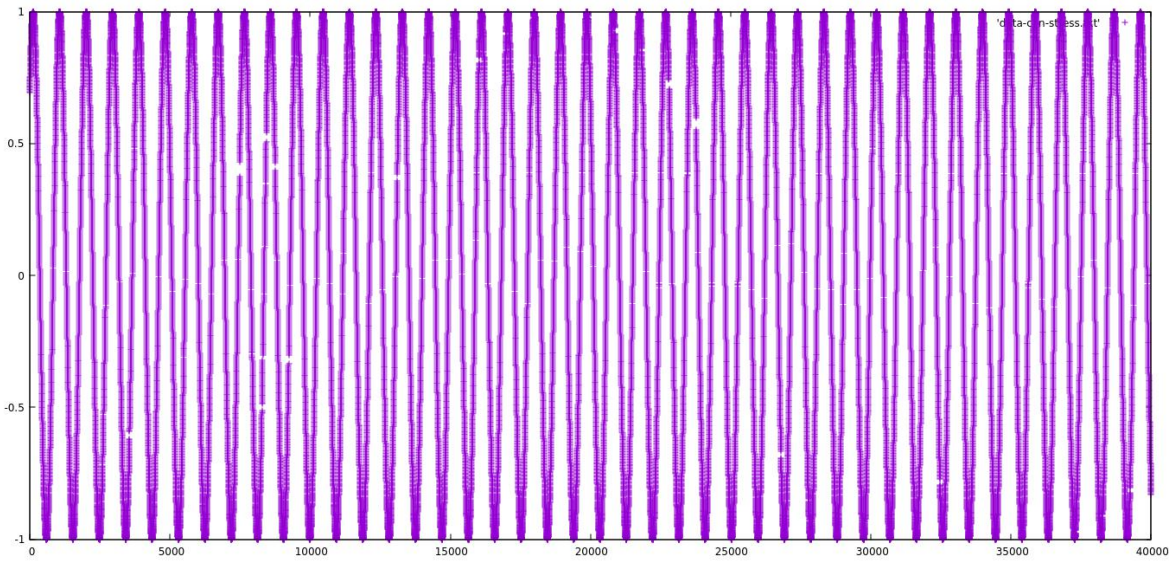
Obtenga gráficos con los puntos muestreados en el entorno tradicional, utilizando distinta cantidad de puntos y distintas bases de tiempo (Sugerencia: utilizar “gnuplot”).

Obtenga gráficos con los puntos muestreados en el entorno de tiempo real, utilizando distinta cantidad de puntos y distintas bases de tiempo (Sugerencia: utilizar “gnuplot”).

Emita conclusiones a partir de la comparación de gráficos en ambos entornos.

Programas de Entorno Tradicional:

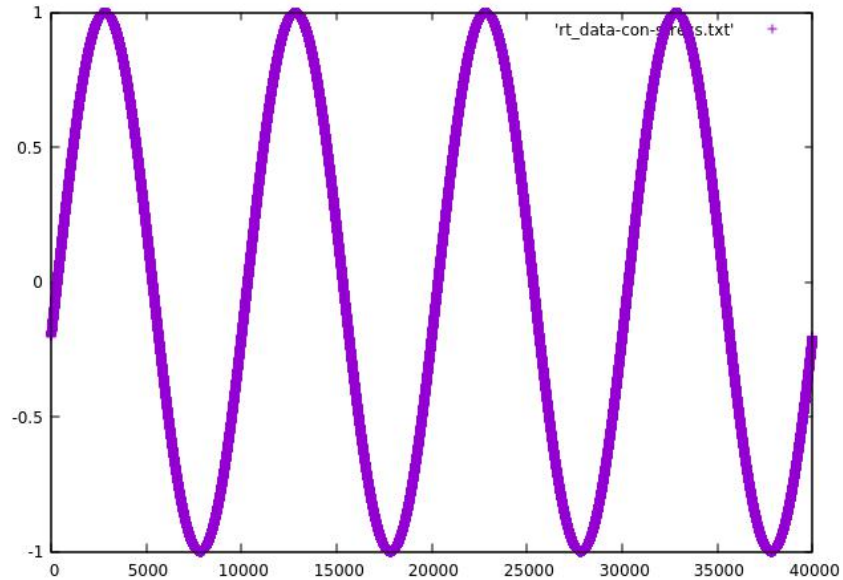
Para una ejecución de 40.000 puntos y TICKS de 1.000us=1.000.000ns se obtuvo el siguiente gráfico:



En él podemos apreciar la onda senoidal con microcortes generados por el stress programado.

Programas de Entorno de Tiempo Real:

Para una ejecución de 40.000 puntos y TICKS de 1.000.000ns se obtuvo el siguiente gráfico:



Puede observarse una gran diferencia en la frecuencia respecto del anterior. Sin embargo, no existen cortes y el sistema es determinístico.

R4) Explique porque debe ejecutarse de modo nativo y no en una maquina virtual.

Porque en modo nativo se puede detectar las interrupciones de tiempo real rápidamente, mientras que en una máquina virtual al estar emulados los componentes físicos, no se tendrá velocidad. Tampoco se tendrá determinismo ya que este sistema en modo nativo coloca un núcleo de tiempo real debajo del núcleo Linux, el cual le da prioridad a las tareas de tiempo real y deja en segundo plano las tareas de usuario. De este modo, con una máquina virtual se pierde esa ventaja, ya que se está poniendo un nuevo núcleo y sistema operativo debajo del núcleo de tiempo real, perdiendo así el determinismo en las tareas.

R5) Investigue acerca de cuáles son los Sistemas Operativos de tiempo real más utilizados en la actualidad y compare sus ventajas/desventajas y sus ámbitos de aplicación.

Utilizando Google Trends, se estudió la tendencia de todos los sistemas operativos de tiempo real encontrados, para observar cuales son los sistemas más estudiados en la actualidad y con mas tendencia a estudiarse a futuro: [1]

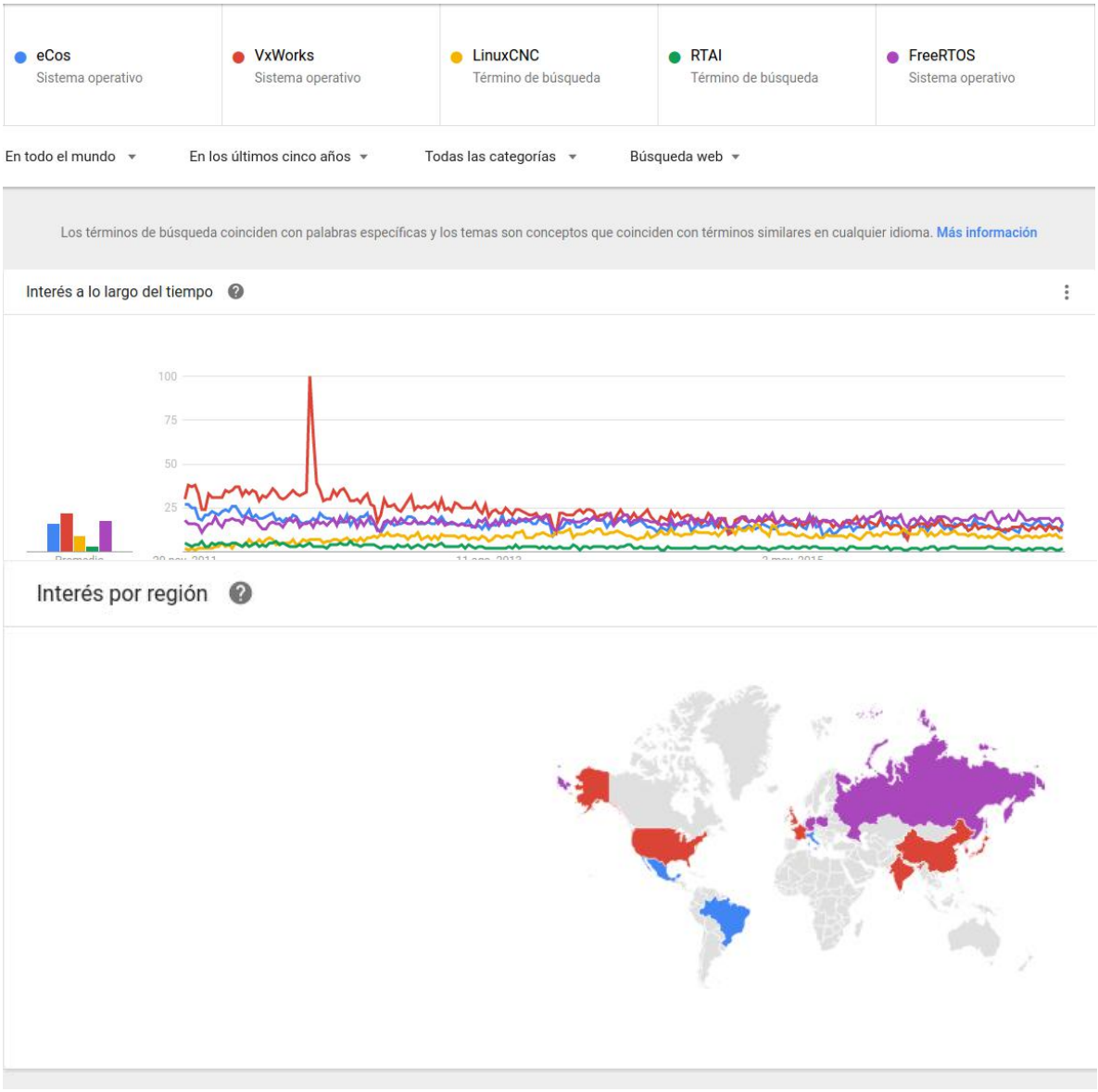
Vemos una fuerte tendencia en búsquedas sobre QNX, que descrece lentamente.

A continuación analizamos estas búsquedas por región:

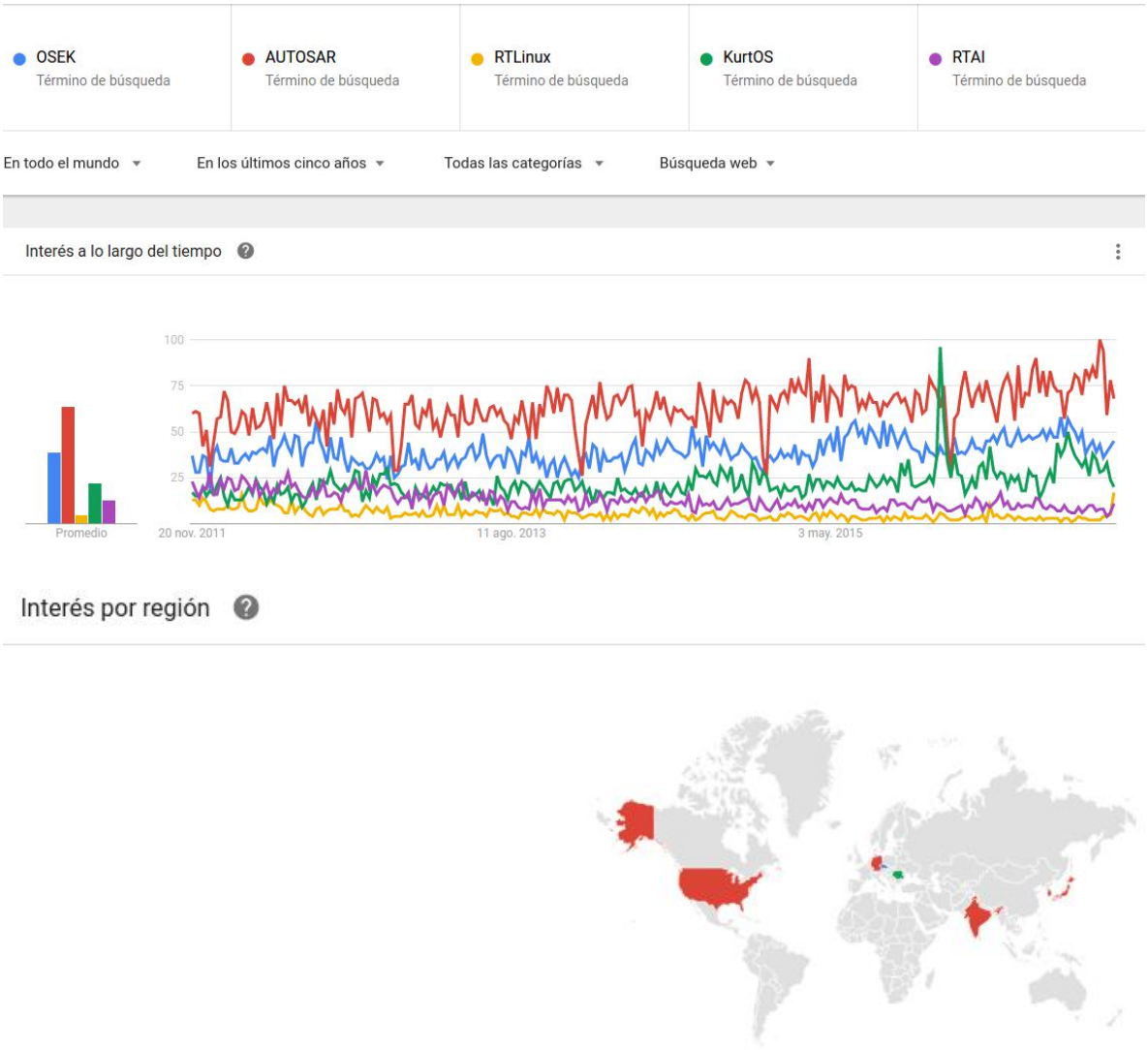
Haiku

QNX

Analizamos las tendencias de otros 5 sistemas operativos en los últimos 5 años: [2]



Por último se analizan otros 5 sistemas en los últimos 5 años: [3]



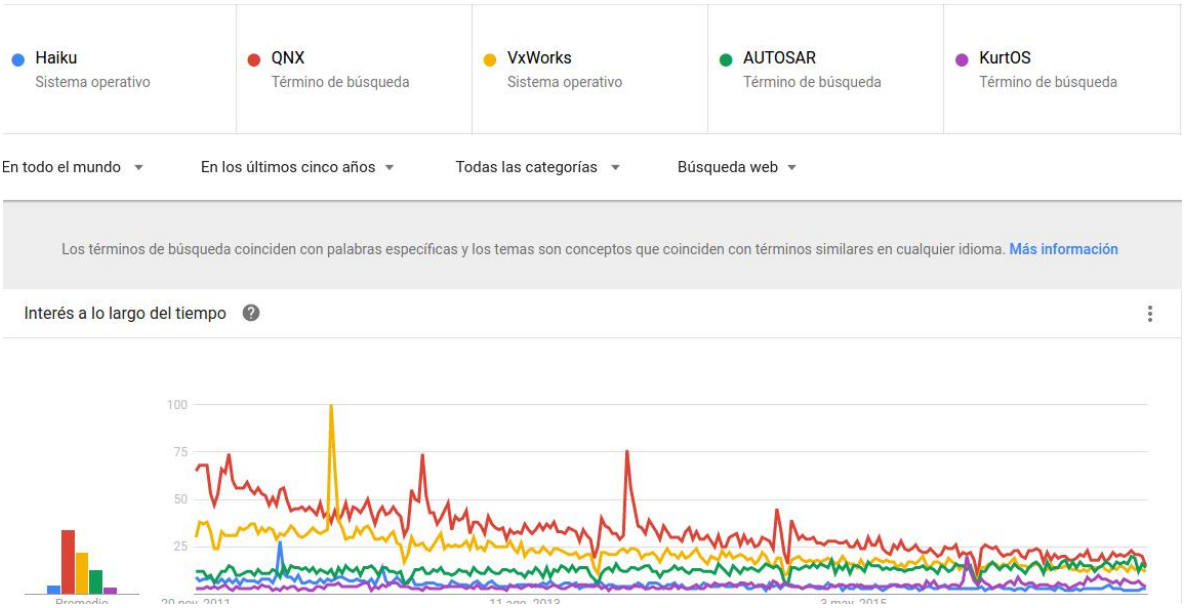
Los países mas industrializados del mundo [4], en orden de mas a menos, son:

1. Estados Unidos
2. China
3. Japón
4. Alemania
5. Reino Unido
6. Francia
7. India
8. Italia
9. Brasil
10. Rusia

Es de esperar que en estos países se utilicen y estudien las mejores herramientas de software, por lo que si los tenemos en cuenta a los 5 primeros, podemos ver que según los gráficos anteriores, los sistemas más utilizados son:

- QNX
- Haiku
- VxWorks
- eCos
- AUTOSAR (sucesor de OSEK)

Si graficamos solo estos sistemas, obtenemos las siguientes tendencias: [5]



Vemos un incremento en la tendencia al uso de AUTOSAR, mientras que QNX y VxWorks tienen también una gran cuota de interés.

AUTOSAR es el sucesor de OSEK para la industria automovilística.. OSEK es una estandarización de donde parten diferentes distribuciones como **FreeOSEK** que es utilizada y mantenida por el proyecto CIAA, aunque la documentación puede ser escasa. Otra opción es openOSEK.

VxWorks es sin duda uno de los mejores y más utilizados, pero hay que pagar una licencia de uso.

QNX, producido por BlackBerry, así como LynxOS son interesantes opciones aunque también están bajo licencia comercial.

RTAI, el núcleo utilizado para este trabajo posee licencia LGPL y buena documentación, **RTLinux** posee licencia GPL2, sobre KurtOs no se encontró información relevante, mientras que **Haiku** posee licencia MIT lo cual lo convierte en una interesante opción. **FreeRTOS** se muestra con un nivel de utilización constante en lo gráficos, es de licencia GPL (modificada) y parece tener un nivel de madurez adecuado, lo que lo convierte en una opción a tener muy en cuenta.

De esta manera, suponiendo que a la mayoría de los proyectos les conviene tener como base un sistema operativo por el cual no se tenga que pagar y por tanto depender de ella, creemos que las opciones mas

interesantes a tener en cuenta son FreeOSEK, RTAI, RTLinux, Haiku y FreeRTOS.

De acuerdo a un estudio[9] de mercado de embebidos en 2014, los siguientes sistemas operativos de tiempo real son los 10 mas usados del mercado de embebidos:[10]

- InHouse/Custom (18%)
- Android (17%)
- FreeRTOS (17%)
- Ubuntu (15%)
- Debian Linux (12%)
- Green Hills Software INTEGRITY (3%)
- Wind River VxWorks (8%)
- QNX Neutrino (5%)
- Micrium µC/OS-II, III (10%)
- Windows CE (8%)
- TI-RTOS Kernel (7%)
- RTEMS open source RTOS diseñado para sistemas embebidos, principalmente usados para control de sondas espaciales y misiles.

Continuando con la investigación se encontró una tabla con todos los sistemas operativos de tiempo de real conocidos (o núcleos), con sus características importantes, como licencia, modelo de fuentes, usuario objetivo, estado, plataformas, sitio oficial.

Ver tabla en Anexo 1.pdf. [6]

DMOZ es un directorio de contenido abierto mantenido por la comunidad. En él hay una sección para sistemas operativos de tiempo real, en el que se puede buscar sistemas de acuerdo a ciertas características. [7]

En la página web de RTAI aparecen una serie de links relacionados a aplicaciones industriales y académicas de RTAI que pueden resultar muy interesantes. [8]

Conclusión

En este trabajo se logró implementar un sistema de tiempo real, con el determinismo necesario para generar y guardar una función sin(x.t) sin cortes.

Se utilizó RTAI como núcleo debajo del núcleo de Debian Wheezy, el cual tiene buena documentación aunque un poco desordenada y repartida en distintas páginas de Internet. Su licencia es LGPL, por lo que nos permite utilizar y desarrollar software sin pagar por ello (siempre siguiendo las normas de dicha licencia).

Se encontró una amplia variedad de sistemas operativos de tiempo real, por lo que seleccionar el sistema adecuado podría parecer una tarea difícil. Sin embargo, siempre debe iniciarse la búsqueda pensando qué se va a realizar. Una vez que se tenga claro el qué, lo próximo es elegir entre sistemas con la licencia mas adecuada al proyecto, y que sirva para la plataforma en la cual se va a utilizar. También es muy importante que el sistema elegido tenga una buena documentación con la cual guiarse, y que se encuentre activo, sobretodo si el proyecto a realizar se trata de un desarrollo a largo plazo en el cual deba actualizarse a las nuevas tendencias tecnológicas.

Referencias

[1] https://www.google.com.ar/trends/explore?q=%2Fm%2F02pd5y,%2Fm%2F0hbbd,RT-11,LynxOS,EasyTasks

[2] https://www.google.com.ar/trends/explore?q=%2Fm%2F027tff,%2Fm%2F014n2t,LinuxCNC,RTAI,%2Fm%2F06qmwf

[3] https://www.google.com.ar/trends/explore?q=OSEK,AUTOSAR,RTLinux,KurtOS,RTAI

[4] http://www.actividadeseconomicas.org/2012/06/top-10-de-los-paises-mas.html

[5] https://www.google.com.ar/trends/explore?q=%2Fm%2F02pd5y,QNX,%2Fm%2F014n2t,AUTOSAR,KurtOS

[6] https://en.wikipedia.org/wiki/Comparison_of_real-time_operating_systems

[7] https://www.dmoz.org/Computers/Software/Operating_Systems/Realtime

[8] https://www.rtai.org/?Links___Application_Links&print

[9] http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf

[10] https://en.wikipedia.org/wiki/Real-time_operating_system