Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Departamento de Electrónica

Cátedra: Técnicas Digitales III - Plan 95A

GUIA DE TRABAJOS PRACTICOS

Ciclo Lectivo 2010

Indice

Trabajos Prácticos de Técnicas Digitales III

Introducción y Régimen de aprobación

La presente guía de Trabajos Prácticos tiene por objeto llevar a la práctica los contenidos vistos en las clases teóricas. De este modo se espera una realimentación entre la aplicación y la lectura de los diferentes conceptos teóricos que permita desarrollar en el alumno un enfoque metodológico para resolver problemas de Ingeniería utilizando como herramientas los diferentes componentes digitales, subsistemas y sistemas aprendidos a lo largo del presente ciclo lectivo.

El grado de complejidad irá creciendo a través de los diferentes ejercicios planteados para cada Unidad Temática.

Cada alumno deberá presentar aquellos ejercicios que lleven la indicación ***Entrega Obligatoria***. La entrega de cada ejercicio se efectuará sin excepciones en las fechas estipuladas en el cronograma de clase que se entregará en la primera clase del ciclo lectivo.

Los calendarios de entrega de los prácticos obligatorios estarán diseñados para que todos los Trabajos Prácticos correspondientes a los contenidos que se incluyen en cada parcial sean revisados por los docentes auxiliares antes del examen. De este modo los alumnos tendrán una devolución con las correcciones de los errores detectados, como forma de realimentación necesaria para el examen parcial.

La no entrega de un ejercicio en la fecha establecida equivale a considerar al alumno o al grupo **Ausente** en ese práctico. En consecuencia se considerará **No Aprobado** dicho práctico. De acuerdo con el reglamento vigente, la aprobación de los Trabajos Prácticos requiere el 80% de los Prácticos Aprobados. En el caso de esta guía de Trabajos Prácticos se requiere la aprobación del 80% de los estipulados de ***Entrega Obligatoria***.

Formato de presentación

- Los archivos fuentes deben tener en todos los casos los comentarios necesarios para clarificar su lectura.

- Deben llevar por cada subrutina / función, un encabezado con la descripción de la operación que realiza, los parámetros que espera como entrada, y en que forma y donde entrega sus resultados.

- Como encabezado del programa, debe haber un comentario que explique claramente que hace dicho programa, y las instrucciones detalladas (comandos) para su compilación y linkeo.

IMPORTANTE:

FORMA DE ENTREGA DE LOS TRABAJOS PRACTICOS

La entrega se realizará en primer lugar en el repositorio de versiones SVN de la cátedra.

El aumno deberá ir volcando en ella los prácticos intermedios que lo conducirán al entregable, con la frecuencia en que vaya dedicándose a las tareas de ejercitación previstas en la asignatura.

La versión final del Tranajo práctico se compondrá de los programas fuentes necesarios, el makefile que permita su compilación y un archivo de texto plano readme con las instrucciones adicionales que el alumno considere pertinente para la ejecución en bochs de su programa.

La no presencia de la versión final completa del TP en la fecha estipulada por parte del alumno en el repositorio indicado se considerará ausente.

Cada equipo de docentes auxiliares establecerá si lo desea como alternativa el envío de las prácticas vía e-mail, sin que esto signifique el relevo e responsabilidad para el alumno de subir su práctica al server SVN de la cátedra.

Solo bajo prueba fundada de indisponibilidad del servicio de internet de la Facultad se aceptarán entregas via e-mail

T.P. Nº 1.Herramientas de Desarrollo

Ejercicio 1.1. Instalar el Bochs, habilitando las opciones de Debugging, Disassembly y SIMD.

Utilizaremos el entorno de emulación x86 Bochs: http://bochs.sourceforge.net/

Bochs es una máquina virtual del tipo vmware aunque algo mas rústico, pero free, y tiene una virtud mayúscula: permite debuggear la PC. Es decir que podremos arrancar en modo debug y avanzar la ejecución de la máquina desde la dirección 0xFFFF:0xFFF0, es decir la primer dirección en donde empieza la rutina del POST pudiendo si lo deseamos examinar el código hasta que se cargue el sistema operativo y una vez cargado seguir dentro del sistema, hasta que se cargue una aplicación a la que también obviamente se podrá debuggear. Es posible establecer breakpoints, y tracear a través de rutinas o pasarlas de un paso como en cualquier debugger normal.

Los comandos del debugger de Bochs están basados en el debugger gdb (GNU Debugger). Podríamos decir que es un subset del gdb.

El manual detallado de comandos del debugger de bochs está en el siguiente link: http://bochs.sourceforge.net/doc/docbook/user/internal-debugger.html

Existen versiones para Windows y Linux. Esta práctica al igual que el resto de la guía se desarrollará íntegramente en Linux.

En la distribución de Debian, el paquete que está en el repositorio es el emulador sin opción de debugging. Para habilitar la función de debugging, entonces es necesario bajar la última versión de fuentes del repositorio dado al principio de esta sección, y compilarlo con las opciones adecuadas. Por lo tanto, hay que compilarlo, pasándole una opción correspondiente al shell script ./configure...

1. Bajar el paquete de archivos fuentes del repositorio correspondientes a la última versión (2.4.2 al momento).

2. Extraer su contenido y descomprimirlo usando el comando tar.

3. Ejecutar en el directorio en el que se descomprimió el siguiente shell script:

./configure --enable-debugger --enable-disasm --prefix=/home/<poner el usuario>/ --enable-mmx –-enable-sse=4 –-enable-fpu –-enable-vbe –-enable-sse-extension ––enable-cpu-level=6

Estos parámetros establecen el tipo de CPU a emular y habilitación de emulación de extensiones SIMD y fpu, entre otras cosas.

VBE se refiere al modo grafico emulado por Bochs (VESA Bios Extension)

Para mas información sobre estos parámetros consultar la página de bochs mencionada anteriormente.

Ejercicio 1.1. Configuración de bochs.

El Bochs se configura mediante el archivo ***.bochsrc***. En la instalación base se incluye un archivo ejemplo que detalla muchas de las opciones disponibles.

Con motivo de testear la instalación, también se provee de una imagen de DOS para verificar que la instalación de Bochs funciona correctamente: http://www.electron.frba.utn.edu.ar/~afurfaro/files/dosboot.rar

Existe una lista de imágenes con un SO ya pre-instalado en http://bochs.sourceforge.net/diskimages.html.

Bajar alguna de las imágenes de Bochs disponibles, y armar el archivo de configuración ***.bochsrc*** que arranque la máquina virtual con dicha imagen.

Una vez configurado el Bochs, debería verse una pantalla similar a:

```
% bochs
========================================================================
                        Bochs x86 Emulator
                          April 14, 2008
========================================================================
00000000000i[     ] reading configuration from .bochsrc
------------------------------
Bochs Configuration: Main Menu
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate. Bochs has already searched for a
configuration file (typically called bochsrc.txt) and loaded it if it
could be found. When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Begin simulation
6. Quit now

Please choose one: [5]
```

entonces Bochs leyó el archivo de configuración ***.bochsrc*** que describe la PC virtual x86, cargando la imagen correspondiente. Está detenido para permitir cambiar opciones antes de iniciar la ejecución. Para iniciar, presionar Enter.

Tal como indica la pantalla anterior de Bochs, puede saltearse este paso e ingresar directamente al programa ejecutándolo con la opción -q.

Una vez arrancada la imagen, Bochs inicia la consola de debugging:

```
Next at t=0
(0) [0x000ffff0] f000:fff0 (unk. ctxt): jmp f000:e05b         ; ea5be000f0
<bochs:1>
```

Ejercicio 1.2. Uso de bochs

Leer la documentación de debugging del Bochs, y realizar un seguimiento de la inicialización de la máquina virtual, ejecutando paso a paso, examinando los registros, memoria, colocando breakpoints, etc.

T.P. Nº 2. BIOS y Boot

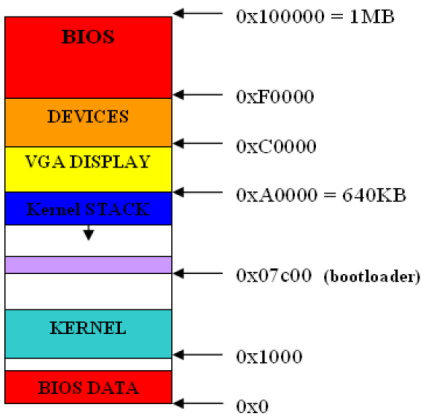
Introducción. Breve síntesis de carga del Sistema Operativo.

El proceso de boot comienza ejecutando el código de la BIOS ubicado en la posición física 0xFFFF0. Allí comienza el POST (Power On Self Test) que es parte de la ROM BIOS. El POST realiza la inicialización básica del hardware (por ejemplo, la placa de video) y su verificación. Luego busca algún dispositivo de booteo: Disco Rígido, Floppy, USB, etc. El orden de búsqueda está determinado en una pequeña memoria RAM estática que se alimenta con una pila de Li. Es lo que se conoce como setup de la PC al que se ingresa mediante alguna tecla específica disponible para tal fin durante algunos instantes cuando recién se enciende la PC. Transcurrido ese breve intervalo se desactiva la opción de ingreso al Setup.

Una vez localizado el dispositivo de arranque, **carga el primer sector de 512 bytes** (excepto en el caso de CDROM, cuyo tamaño es 2048) en la posición de memoria 0x07C00 y salta a esa dirección. Las imagen de arranque es la encargada de cargar el kernel y luego pasarle el control.

IMPORTANTE: Las imágenes de arranque deben ocupar exactamente 512 bytes (excepto en el CDROM), y estar firmada en los últimos dos bytes con 0x55AA.

A continuación se muestra un esquema posible de la distribución de la memoria al arrancar la máquina.



Link interesante: http://en.wikibooks.org/wiki/X86_Assembly/Bootloaders

Ejercicio 2.1. Debuggeando BIOS

Iniciar el Bochs, y debuggear el POST de la BIOS. Seguir el código tratando de ver (a grandes rasgos) que operaciones realiza el POST.

Ejercicio 2.2. Boot de un OS

Iniciar el Bochs, y cargar alguna imagen de Linux. Seguir el código e indicar el momento exacto en que el Bootloader pasa el control al Kernel.

Ejercicio 2.3. Programa autobooteable

En el link http://en.wikibooks.org/wiki/X86_Assembly/Bootloaders, se presenta una versión "bootloader" del programa Hello World y las líneas necesarias para compilarlo usando NASM. Descárguelo, compile y pruebe su funcionamiento, debuggeando paso a paso.

T.P. Nº 3. Compilación, formato ELF y creación de imágenes para Bochs.

Introducción: Guía de herramientas de desarrollo

Para compilar utilizaremos las herramientas estándar de Linux: gcc, make, objdump, objcopy, ld, etc.

El compilador gcc utiliza por default el formato ELF para los archivos ejecutables.

Se dispone del siguiente material para comprender como está organizado este formato:

Tool Interface Standard (TIS), Executable and Linking Format (ELF) Specification, Version 1.2, TIS Committee, May 1995.
http://www.electron.frba.utn.edu.ar/~afurfaro/descargas/Ejemplos%20programaci%f3n/Boot/elf.pdf

Executable and Linkable Format (ELF), Princeton University.
http://www.electron.frba.utn.edu.ar/~afurfaro/descargas/Ejemplos%20programaci%f3n/Boot/ELF_Format.pdf

Para extraer el código assembler de un archivo ejecutable, podemos usar la herramienta *objdump*:

```
objdump -S a.out > a.asm
```

También podemos usar la herramienta *objcopy* para extraer el código máquina de un archivo ejecutable:

```
objcopy -S -O binary a.out a.bin
```

Para compilar sin linkear, generando únicamente archivos objeto usar:

```
gcc -c hello.c
```

Luego es posible linkear los distintos archivos .o utilizando ld. Lo mas conveniente es pasar las opciones para el linker a través del gcc y dejar que éste lo invoque.

Ejercicio 3.1. Práctica asistida de construcción de un programa booteable.

Para armar la imagen para Bochs, lo escribimos en Assembler de 16 bits, compilamos los fuentes y generamos el ejecutable. La BIOS, luego de realizar el POST, salta a la posición 0x7C00 en modo real. Entonces para que las etiquetas del código que estamos compilando estén correctamente resueltas debemos indicar esto al linker. Luego extraemos el código y armamos la imagen. La imagen de booteo debe ocupar exactamente 512 bytes y debe estar firmada con un "AA55" obligatoriamente.

Para eso se provee con un *target* para armar el Makefile. Una vez generados los archivos objeto, puede utilizarse el siguiente *target*:

```
ld -N -e start -Ttext 0x7C00 -o bootloader.out bootloader.o
objdump -S bootloader.out >bootloader.asm
objcopy -S -O binary bootloader.out bootloader
sign bootloader
```

El archivo sign genera el archivo binario de 512 bytes, firmado, listo para incluir en el archivo de configuración de Bochs. Puede bajarse de http://www.electron.frba.utn.edu.ar/~afurfaro/files/sign

Una vez generados el bootloader y el resto del código, se puede armar la imagen bochs de la siguiente manera:

```
dd if=/dev/zero of=bochs.img~ count=10000 2>/dev/null
dd if=bootloader of=bochs.img~ conv=notrunc 2>/dev/null
dd if=kernel of=bochs.img~ seek=1 conv=notrunc 2>/dev/null
mv bochs.img~ bochs.img
```

Ejercicio 3.2. Segundo programa booteable

Escribir un código que no haga uso de bibliotecas externas (por ejemplo, una serie de instrucciones ASM inútiles), compilarlo, linkearlo y generar una imagen bochs que bootee ese código. Hacer un debugging para verificar que el código cargado esté correcto.

Ejercicio 3.3. Desarrollo de un bootloader

Siguiendo el tutorial:

http://osguru.net/index.php/(Tutorial-OS)_LBA_HDD_Access_via_PIO,

escribir un bootloader que lea el segundo sector del disco, y lo ejecute. Guardar el hello_world usando la BIOS en dicho sector.

Ejercicio 3.4. Combinación de C y ASM

Verificar la potencia y flexibilidad que brinda la combinación de lenguaje C y ASM según los requerimientos y complejidad del programa.

Entender el uso de los compiladores de C y ASM, de los linkeadores y sus diferencias. Paso de argumentos por la pila.

Para ello escribir dos programas:

a) Cuerpo principal en C que solicite dos operandos por consola, y una operación (s: suma, r:resta) y devuelva por pantalla el resultado.

Las operaciones serán realizadas por funciones auxiliares en ASM (func.asm)

b) Cuerpo principal en ASM. Espera presión de una tecla. Según la tecla presionada (a-z) cambia el color del texto y sale.

El "switch" que evalúa el scan code se realizará en un programa externo en C (evalscancode.c)

T.P. Nº 4. IA-32 - Modo Protegido.

Ejercicio 4.1. Entrada a Modo Protegido

Escriba un programa ***autobooteable*** que ponga al procesador en modo protegido, seguidamente ponga la pantalla en modo de video inverso, y termine su ejecución mediante HLT.

Asuma un controlador de video color.

Ejercicio 4.2. Manejo de Interrupciones en Modo Protegido

Agregue al programa desarrollado en el Ejercicio 4.1 el manejo de la interrupción de teclado de la PC (INT 9h), de modo tal que una vez puesta la pantalla en modo de video inverso espere la presión de una tecla cualquiera. Por cada tecla presionada, incrementará una variable cuyo contenido presentará en pantalla .

Ejercicio 4.3. Ordenamiento de las Interrupciones

Tome el programa del Ejercicio 4.2. Antes de pasar a modo protegido, reprograme los PICs 1 y 2 de modo que utilicen el rango de tipos de Interrupción INT 20h a INT 2Fh.

Ejercicio 4.4. Manejo de Excepciones en Modo Protegido *Entrega Obligatoria*

Tome el código del Ejercicio 4.3 e inserte un handler para cada una de las excepciones del procesador. Se busca proveer un mínimo manejo de excepciones de modo de evitar que el sistema se estrelle ante fallas de protección. Para ello cada excepción deberá tener un handler que detenga allí al procesador permitiendo examinar los registros con el debugger del bochs.

A los fines de la prueba de funcionamiento del sistema de escape de excepciones, se pide que cuando el programa espera la presión de una tecla para salir, en el caso en que se pulse la tecla 'J' (mayúscula o minúscula), se ejecute cualquier operación no válida que genere una excepción.

Ejercicio 4.5. Cuenta de memoria RAM en el sistema

Tome el código del Ejercicio 4.4.

1. Quite la activación de excepciones al pulsar "J".

2.Agregue el código necesario para determinar la cantidad de memoria RAM presente en el sistema. La cantidad total en KBytes deberá imprimirse en el borde superior izquierdo de la pantalla.

Ejercicio 4.6.Programa de arranque

Utilizando un bootloader, arrancar un sistema almacenado en un archivo denominado kernel.bin, que estará en la primer entrada del Directorio Raíz del disquete. El disquete tiene el formato FAT 12 standard de DOS.

El archivo kernel.bin será el programa del Ejercicio 4.5, con las adaptaciones que correspondan a la condición de ser booteado.

Además deberá:

- ❑Armar una GDT con al menos 4 descriptores de segmento que conformen un modelo flat con dos segmentos de código (uno en DPL=00 y el otro en DPL=11), y dos segmentos de datos (uno en DPL=00 y el otro en DPL=11). Los 4 segmentos deben tener 4 GBytes de tamaño. Como tenemos la gate A20 deshabilitada, la GDT debe colocarse en alguna dirección del primer mega (antes de la dirección 100000h).

- ❑Imprimir en pantalla un mensaje de bienvenida, y quedarse esperando una tecla (mediante interrupciones).

Ejercicio 4.7. Paginación.

Tome el Ejercicio 4.6 y agregue las tablas de paginación correspondientes para trabajar mediante identity mapping. Cubra solamente los primeros 4 Mbytes de RAM.

Ejercicio 4.8.Manejo de tareas simple – scheduler como tarea

Lea el artículo publicado en:

http://embedded.com/columns/technicalinsights/55301875?_requestid=244198

Luego tome el código del Ejercicio 4.7 y agregue el código necesario para administrar dos tareas encargadas de la presentación de información en forma simultánea en las mitades superior e inferior de la pantalla respectivamente, utilizando al timer tick de la PC como base de tiempos.

La información a presentar en cada mitad de la pantalla, consiste en el tiempo acumulado (expresado en décimas de segundos) que lleva escribiendo en cada mitad.

Para simplificar los cálculos en su código, reprograme el Timer Tick para generar una interrupción por mseg. Tenga en cuenta que el Timer 0 de la PC tiene una señal de clock externa establecida por un cristal de 1.19 MHz.

En su condición de arranque (default), el programa dedicará el 50% de los ciclos de timer para cada tarea (de modo que en esta condición el número presentado en cada mitad será el mismo).

Para alterar la prioridad de cada tarea (y por ende desbalancear los valores presentados en cada una) se desea utilizar la tecla F2, y la tecla F3 para aumentar en pasos del 10% la prioridad de la mitad superior e inferior respectivamente. Cada mitad aumenta su prioridad en desmedro de la otra. Cuando se llega al extremo de tener una tarea al 0% no se la debe invocar hasta que su prioridad aumente al menos al 10%

Se vuelve al modo real limpiando la pantalla cuando se pulsa la tecla F10.

Utilizar una puerta de tarea para IRQ0, y una puerta de interrupción para IRQ1.

Ejercicio 4.9. Manejo de tareas simple – scheduler como rutina de interrupción

Repita el ejercicio 4.8 utilizando una puerta de interrupción en IRQ0.

Ejercicio 4.10. Uso del debugger de Bochs para analizar el comportamiento *Entrega Obligatoria*

Tome los kernels de los Ejercicios 4.8 y 4.9, y mediante la inclusión de breakpoints analice los siguientes aspectos del comportamiento:

1. Punto del código en el que el scheduler reasume su ejecución como respuesta al timer tick en el problema del Ejercicio 4.8. Represente en base al comportamiento observado un diagrama de transiciones entre las diferentes tareas involucradas.

2. Idem para el 4.9

3. Como se comporta el bit Busy y el Bit NT para el programa del 4.8

4. Idem para el 4.9

5. Escriba un informe del comportamiento del procesador para los casos planteados en los diferentes ítems.

Entregable: Diagrama de estados y transiciones para los ítems 1 a 4, en Visio, Powerpoit, o Word. Documento en formato .DOC para el punto 5.

Ejercicio 4.11.Ejecución en dos niveles de privilegio.

Tome el código del 4.8 o del 4.9 (elija el que prefiera), y modifíquelo para que ambas tareas ejecuten, en un segmento de RPL = 11.

Las tareas dejarán de presentar su tiempo de ejecución y mediante el acceso a dos servicios (Fecha y Hora) implementados en el segmento de código de RPL = 00, accediendo directamente al Real Time Clock de la PC. Las tareas presentarán:

- Tarea 1: Presenta la Hora del Sistema en la posición de pantalla Fila 8 Columna 35, en el formato **hh:mm:ss**.
- Tarea 2: Presenta Fecha y Hora del Sistema en la posición de pantalla Fila 16 Columna 35 en formato **dd:mm:aaaaa** y en Fila 17 Columna 35 en formato **hh:mm:ss**. Ídem Servicio Hora del Sistema.

Las tareas, descriptas se seguirán ejecutando de acuerdo al esquema de manejo de prioridades establecido en el 4.8 o en el 4.9 según haya sido su elección**.**

El sistema finaliza (HLT) cuando se pulsa la tecla F10, o cuando alguna de las dos tareas llega a los 3 minutos de operación efectiva.

Ejercicio 4.12.Scheduler con manejo de lista de tareas de longitud dinámicamente variable.

Tome el código del 4.11, y realice las siguientes modificaciones / mejoras:

- El scheduler maneja las tareas que se encuentran en una lista de elementos que se definen a través de la siguiente estructura

```
struc task_sel
     selector        resw 1     ;Selector del TSS de la tarea
     prioridad_ini   resb 1     ;Número entero entre 1 y 10 que
                                ;indica cuantos ciclos de timer tick
                                ;se asignaron a la ejecución de la
                                ;tarea al momento de su carga.
     prioridad       resb 1     ;Número entero entre 1 y 10 que
                                ;indica cuantos ciclos de timer tick
                                ;debe estar en ejecución la tarea
                                ;actualmente. Al inicio es igual
                                a ;prioridad_ini.
endstruc
```

- La lista no es enlazada, sino una lista de estructuras task_sel consecutivas en memoria terminadas en NULL (00h)
- El scheduler debe alojar a las tareas de la lista en un frame de 100 mseg. El tiempo remanente a esta duración máxima lo completará con una tarea denominada Idle que pone al procesador en estado HALT, de modo de minimizar el consumo del sistema.

- La cantidad de elementos de la lista es administrado por un módulo externo, que se encarga de insertar y remover elementos de la misma de manera transparente a su scheduler (lo desarrollará en el Ejercicio 4.14, no aquí). Este mismo módulo se encarga además de modificar las prioridades de todas las tareas de la lista de modo que siempre quepan en 100 mseg., descontando ticks a cada campo prioridad cuando se recarga de tareas el sistema, y restituyéndolos a su valor inicial a medida que se descarga la demanda de ejecución de tareas

Ejercicio 4.13. Kernel DSP *Entrega Obligatoria*

Tome el código del ejercicio 4.12 y reemplace a las tareas por las siguientes:

- Tarea 1: Recibe por COM1 un mapa de bits y lo presenta en pantalla. A partir de entonces de acuerdo a la tecla que se pulse efectuará un procesamiento determinado:

Tecla 'E' ejecuta una erosión de la imagen presentada

Tecla 'B' Muestra los bordes de la imagen presentada

Tecla 'D' ejecuta la Dilatación de la imagen presentada

- Tarea 2: Presenta al lado de la tarea 1 el histograma de brillo de las imágenes

- Tarea Idle. Pone al sistema en HLT

Ejercicio 4.14.Manejo completo de tareas.

Tome el programa del 4.13 e incorpore un sub-sistema que permita agregar y quitar tareas de la lista de ejecución dinámicamente según el procedimiento descrito en el último ítem del mencionado Ejercicio.

Ampliando la especificación el sub-sistema a incorporar debe activarse al pulsar F8. A continuación debe pulsarse la tecla "I" si se desea Insertar una tarea, seguida del número de la tarea (01 a 20), y de la prioridad (01 a 10). No requiere los datos, sino que el operador los ingresa directamente. Por ejemplo, si ingresa la string "I1708", significará Insertar la tarea 17 con 8 como prioridad. Los números ingresados se asumen en decimal. Si desea remover una tarea, luego de F8, ingresará "R", seguido del número de tarea (01 a 20).

La tarea Idle no puede removerse. Es la tarea 0 y ocupa la primer posición de la lista de tareas.

Las funciones que realizan las dos operaciones requeridas funcionarán según el siguiente detalle de especificación:

- Función ***_insert_task***:
  - Recibe como argumentos el selector de la tarea a insertar y su prioridad.
  - Inserta la tarea al final de la lista de ejecución, siempre que la tarea no esté ya incluida en la lista de ejecución. En tal caso no hace nada.
- Función ***_delete_task***:
  - Recibe como argumento el selector de la tarea.
  - Remueve la tarea de la lista de ejecución. Si la tarea no está en la lista, entonces no hace nada.

Para mayor simplicidad, las 20 tareas están todas descargadas en la memoria del sistema y la GDT tiene precargados todos los descriptores de TSS asociados a las mismas.

T.P. Nº 5.Arquitectura de Procesadores *Entrega Obligatoria*

Ejercicio 5.1.Pipeline

Un determinado microprocesador dispone de un "instruction pipeline" lineal, en el cual se implementan, mediante la técnica de "look ahead", las siguientes etapas:

a- Búsqueda de instrucciones (Instruction fetch)

b- Decodificación de instrucciones (Instruction decode)

c- Cálculo de la suma de operandos (Operand add calculation)

d- Búsqueda de operandos (Operand fetch)

Se desea ejecutar 20 instrucciones, si cada etapa demora un ciclo de reloj en procesar la instrucción , determine:

a- La cantidad de ciclos de reloj que demora el "pipeline" en procesar las 20 instrucciones.

b- La cantidad de ciclos de reloj que demora el procesador en ejecutar las 20 instrucciones si no dispusiera de "pipeline". Calcule la optimización lograda por el uso de "pipeline".

c- Sabiendo que el 50% de las etapas demoran 30ns y el restante 50% demora 45ns en procesar cada instrucción y teniendo en cuenta que el tiempo de cada

etapa de "latch" es de 5ns. Indique cual es la frecuencia máxima a la cual puede trabajar el "pipeline".

Ejercicio 5.2. Arquitectura Superescalar

Recordando que el procesador P-I de la familia IA-32 de Intel, dispone de 2 "instruction pipeline" (U & V) y sabiendo que opera a 133 MHz. Indique cuánto tiempo demora en procesar el siguiente código:

```
pushf
xor   ax, ax
mov   ax, 60h
popf
```

Ejercicio 5.3. Ejecución fuera de orden

Si a un procesador Pentium IV con motor de ejecución fuera de orden ingresa el siguiente código.

```
      mov   edi,14456
      mov   ecx,10000
      mov   esi,[0x1F0E9708]
      cld
otro:
      lodsd
      and   [edi+eax],0xFF7FE806
      jz    true
      mov   edx,12345
      add   edi,100
true:
      loop  otro
```

Considerando que el tamaño de una línea de cache L1 en este procesador es de 64 bytes, y que las direcciones de memoria referidas en el programa no son precacheables, se pide:

a. Analizar el efecto del "cache miss" en cada referencia a memoria respecto de su impacto en la ventana de ejecución.

b. De acuerdo a la cantidad y tipos de ports de ejecución del procesador, los "cache miss" que pueden producirse, y considerando que el procesador puede retirar a lo sumo 3 resultados de instrucciones por ciclo de clock, determinar cuántos ciclos de clock insume la ejecución del bloque de código anterior, y escriba la secuencia en que se ejecutarán las instrucciones.

Ejercicio 5.4. Memoria Cache

Un microprocesador de 32 bits, tiene incorporado un subsistema de memoria cache Level 1 de 16 Kbytes de capacidad total. El subsistema trabaja en modo asociativo de 4 vías con líneas de 16 Bytes de tamaño, agrupadas en sets de 8 líneas.

El microprocesador tiene 36 líneas de address hacia la memoria externa.

a. Dibuje el diagrama de organización del subsistema completo: Controlador cache – memoria cache – memoria del sistema, indicando claramente en cuantas páginas se divide ésta última.

b. Indique para una dirección física como se asignan dentro del cache los diferentes campos de bits para determinar si el acceso es un hit o un miss.

c. Donde se asigna dentro del cache la dirección física 0x7FF608EAC.

Ejercicio 5.5. Memoria Cache

Dado el siguiente código

```
while ( semáforo == TRUE)
{
      buffer [i] = buffer [i] * j;
      i++;
}
```

a. Señalar un caso de vecindad espacial

b. Señalar un caso de vecindad temporal

T.P. Nº 6. Sistemas Operativos Multitasking

Ejercicio 6.1. Señales

Escribir un programa que cree un proceso hijo que imprima un mensaje a fin de identificarse escribiendo su número de proceso. El proceso padre al recibir queda esperando por stdin. Cuando recibe "S" por dicha entrada, le envía una señal SIGUSR1 al hijo, el hijo acusa recibo de la misma y termina la ejecución.

No deben quedar procesos en estado ZOMBIE (defunct).

Ejercicio 6.2. Señales

Escriba un proceso que maneje las siguientes señales por medio de handlers propios:

SIGUSR1: Cada vez que la recibe crea una instancia child. La instancia child presenta por stdout el siguiente mensaje "Soy el proceso N°: ", y luego de 60

segundos finaliza su ejecución. No se deben crear más de ***n*** childs, en donde ***n*** es el primer argumento recibido por línea de comandos.

SIGINT: El proceso no debe aceptar ser interrumpido desde el teclado mediante CTRL-C

SIGCHLD: Evitar la generación de programas zombies.

Ejercicio 6.3.Procesos

Modifique el programa del 6.2, para que cada proceso child presente la siguiente información en pantalla cada 10 segundos.

"Soy el proceso ***xxxx***. Mi padre es el proceso ***yyyy***. Mi GroupId es ***zzzz***. Mi copia de la variable ***Nchilds*** vale ***nn***". (NChilds es la variable en la que se cuentan los procesos hijos creados mediante SIGUSR1.

Todos los procesos finalizan únicamente con SIGKILL. No incluyen el tiempo de expiración solicitado en el 6.2.

Ejercicio 6.4.Redirecciones

Para mejor visualización de los resultados modifique el programa del 6.3 para que la salida que dirige con printf() en lugar de salir por stdout vaya al archivo /home/tdiii/pidxxxx, en donde pid es el process ID del child.

Ejercicio 6.5.Redirecciones

Hacer un programa que utilice como mecanismo de comunicación un Named PIPE y que corra en dos instancias separadas en diferentes consolas. Recibe un argumento por línea de comandos que le indica si lee o escribe. El argumento es -r o –w. De este modo el mismo programa lee stdin y escribe en el Named PIPE, o lee el Named PIPE y escribe en stdout respectivamente.

Ejercicio 6.6.System V IPC's

Desarrollar un par de programas que se ejecutará múltiples instancias. La primera que se ejecuta crea una shared memory, y una cola de mensajes. Anota en la shared memory su PID.

Las demás instancias se conectarán a estos recursos y harán lo propio con su PID en la shared memory.

Cada proceso que finaliza retira su PID de la shared memory.

Cada proceso interroga al usuario para escribir datos en la cola de mensajes. Cuando el usuario termina de tipear el mensaje y pulsa la tecla ENTER, el

proceso que lo atiende mira en la shared memory los posibles destinatarios, presenta la lista de PIDS en la pantalla uno al lado del otro separados por un espacio, y a continuación solicita al usuario el ingreso del destinatario.

Si el usuario ingresa ´*´, entonces, el mensaje es para el primero que lo lea

Si el usuario ingresa un PID, el proceso debe validarlo contra la lista leída de la shared memory, si no coincide solicitar reingreso, y si coincide enviarlo de modo que solo lo pueda retirar dicho destinatario.

Ejercicio 6.7.System V IPC's

Tome el programa del 6.6, y solucione la condición de borde que puede darse si un proceso le envía información a un proceso que está finalizando.

Ejercicio 6.8.Manejo de bloqueo de procesos

Escriba un proceso que espere al mismo tiempo strings, por una Named PIPE /tmp/tdiii_pipe, e ingresos por el teclado. Por cada ingreso debe escribir en consola el mensaje precedido de "[PIPE]: ", o "[KEYB]: ", según su procedencia.

Ejercicio 6.9.Acceso al dispositivo de audio

Escriba un programa que acceda al dispositivo de audio /dev/dsp, y distribuya la información obtenida en tres instancias child. El dispositivo de audio se debe programar para entregar dos canales, cada uno con muestras de 16 bits y a una velocidad de 40000 SPS.

Cada instancia child llama a una de tres rutinas externas para procesamiento de la señal: ***echo***, ***fir_low_pass***, y ***delay***. A los fines de este ejercicio no es necesario implementar las tres funciones. Los resultados de la llamada se guardan en sendos archivos: ***/home/tdiii/echo***, ***/home/tdiii/fir_low_pass***, y ***/home/tdiii/delay***, respectivamente.

Utilice un shared memory como recurso para almacenar el audio y un par de semáforos para sincronizar el acceso por parte de los childs. Tenga en cuenta que el proceso es en tiempo real. Un child no debe escribir un bloque de datos de audio nuevamente en un archivo una vez que ya lo ha hecho, y el proceso padre debe asegurar que todos los childs han leído la información antes de refrescar el buffer de la shared memory con un nuevo bloque de información.

Para no perder información el proceso padre debe monitorear la actividad de los tres childs en forma periódica y si no está funcionando adecuadamente debe terminar esa instancia y relanzar una nueva. Si un child termina inesperadamente su ejecución debe relanzar una nueva instancia de ese proceso.

Ejercicio 6.10.System V IPC y Señales

Tome el programa del 6.9. Se pide:

Reemplazar la sincronización de procesos utilizando señales al Group ID

Trabajar mediante archivo de configuración para establecer el comportamiento del proceso. El archivo de configuración llamado streamer.conf tiene el siguiente formato:

```
Nchilds=nn            // Número máximo de instancias child
SampligRate=sssss     // Velocidad de Muestreo del DSP
Nchanels=c            // Número de canales de audio a adquirir
SampleSize=ss         // Tamaño de la muestra en bits
BufferSize=k          // Tamaño del buffer en kbytes
```

El archivo es texto puro en ASCII. Si no se especifica alguno de los campos en el archivo de configuración, deberá tomar los siguientes valores default: Nchilds = 20, SampligRate = 8000, Nchanels = 1, SampleSize = 8, BufferSize = 8.

Este archivo de configuración puede ser modificado por el usuario durante la operación del proceso.

Cada vez que recibe SIGUSR2, el padre debe releer el archivo de configuración y modificar en tiempo real su comportamiento, de acuerdo a los nuevos valores establecidos, por el usuario.

Ejercicio 6.11.DSP sobre Linux

Tome el programa del 6.10 y agregue las tres rutinas externas programadas en assembler utilizando las instrucciones SIMD de los procesadores IA-32.

Ejercicio 6.12.DSP sobre Linux *Entrega Obligatoria*

Modifique el programa del 6.11 para que determine que versión del Modelo de ejecución SIMD contiene el procesador de la PC en que ejecuta y en función de esto llame a las rutinas mas adecuadas para su ejecución. Debe tener diferentes rutinas para cada función de acuerdo con las instrucciones y formatos de datos definidos en cada versión SIMD de la familia IA-32.

Sugerencia: Utilice la instrucción CPUID al inicio.

Ejercicio 6.13.System V IPC's

Modifique el esquema de distribución de audio del programa del 6.11 utilizando una message queue.

Ejercicio 6.14.Threads *Entrega Obligatoria*

Modifique el programa del 6.13 empleando Linux threads en lugar de crear procesos con fork().

Ejercicio 6.15.In Line assemby. *Entrega Obligatoria*

Tome los programas del 6.11 y mediante *in line assembly* acceda al registro TSC del procesador inmediatamente antes e inmediatamente después de la ejecución de cada función.

Obtenga la cantidad de ciclos de clock que demanda la ejecución de las diferentes versiones de cada rutina y establezca una tabla comparativa entre las diferentes tecnologías involucradas: MMX, SSE, SSE2, o SSE3.

Ejecutado en diferentes máquinas, el programa deberá presentar en pantalla los valores correspondientes al procesador.

Una vez completado el protocolo anterior, vuelva a repetirlo, pero utilizando para la ejecución del programa el comando adecuado para lanzarlo con prioridad Real Time.

Ejercicio 6.16.Device Drivers *Entrega Obligatoria*

Hacer un driver que utilice memoria del sistema para hacer un clipboard entre procesos de usuario usando el /dev/portapapeles. El driver debe permitir a las aplicaciones de usuario escribir datos en un bloque de memoria del Kernel, y leer dicho bloque utilizando los métodos tradicionales de un driver.

¿Que ventajas y desventajas puede tener implementar un mecanismo IPC de esta manera?.

¿Que debemos tener en cuenta cuando reservamos memoria en modo kernel?

Ayuda: en modo kernel existe la función kmalloc() que equivale a malloc() en modo user

T.P. Nº 7.Procesamiento Digital de Señales

Ejercicio 7.1.Convolución 1D

Escriba utilizando instrucciones SSE, una función invocable desde un programa C que reciba dos punteros a sendos arreglos, y que asumiendo al primero como un vector de muestras de señal y al segundo como la respuesta de un sistema lineal invariante en el tiempo, calcule la convolución circular de ambos, retornando el puntero al vector resultado (cadena terminada en NULL).

Verificar el resultado modelizando previamente la función en Matlab.

Ejercicio 7.2. Convolución 2D

Utilizando instrucciones SSE escriba una función invocable desde un programa C que reciba un puntero a una matriz de ***n***x***n***, un puntero a una matriz de ***m***x***m*** *y* dos enteros con los valores de ***n*** y ***m***.

La función debe asumir a la primer matriz como una matriz de píxeles de una imagen en escala de grises de 8 bits por píxel y a la segunda como la respuesta de un sistema lineal invariante en el tiempo, y calcular la convolución circular de ambas, retornando el puntero al vector resultado (cadena terminada en NULL).

Verificar el resultado modelizando previamente la función en Matlab.

Ejercicio 7.3. Suma de Imágenes

Escriba utilizando instrucciones SSE, una función invocable desde un programa C que reciba dos punteros a sendas matrices de ***n***x***n***, un entero con el valor de ***n***, y un entero con el coeficiente de mezcla ***f***. Asumiendo a ambas matrices como imágenes en escala de gris de 8 bits por píxel, y a ***f*** < 256, la función deberá devolver un puntero a una matriz que contenga la suma de ambas según la siguiente fórmula:

$$\mathrm{Pr}_{(x,y)} = \left[\frac{f \cdot \mathrm{Pa}_{(x,y)}}{255}\right] + \left[\frac{(255-f) \cdot \mathrm{Pb}_{(x,y)}}{255}\right]$$

Donde $\mathrm{Pa}_{(x,y)}$ y $\mathrm{Pb}_{(x,y)}$, son respectivamente los píxeles correspondientes a las coordenadas (x,y) de ambas imágenes, $\mathrm{Pr}_{(x,y)}$ es el valor del píxel correspondiente a las coordenadas (x,y) de la matriz resultado.

Verificar el resultado modelizando previamente la función en Matlab.

Ejercicio 7.4. Procesamiento de Imágenes: Detección de Bordes

Escriba utilizando instrucciones SSE, una función invocable desde un programa C que reciba un puntero a una matriz de ***n***x***n*** *y* un entero con el valor de ***n***. Asumiendo a la matriz como una imagen en escala de gris de 8 bits por píxel, implementar la función de detección de borde, retornando la matriz resultado con la imagen binarizada mediante un puntero.

Verificar el resultado modelizando previamente la función en Matlab.

Ejercicio 7.5. Procesamiento de Imágenes: Plano de Bits

Escriba utilizando instrucciones SSE, una función invocable desde un programa C que reciba un puntero a una matriz de ***n***x***n*** y un entero con el valor de ***n***.

Asumiendo a la matriz como una imagen en escala de gris de 8 bits por píxel, implementar 8 imágenes una por cada plano de bits de la imagen original.

Verificar el resultado modelizando previamente la función en Matlab.

Ejercicio 7.6.Procesador de imágenes. *Entrega Obligatoria*

Escriba un programa en C que abra los archivos de imagen necesarios para cada caso, todos en escala de grises y 8 bits por píxel, y que aplique las funciones desarrolladas en 7.2, 7.3, 7.4, y 7.5.

T.P. Nº 8.Redes de datos.

Ejercicio 8.1.Cliente servidor TCP/IP No concurrente.

Escriba un par cliente - servidor que cumpla los siguientes requisitos.

Servidor:

Ejecución no concurrente. Espera conexiones por el port TCP 8145. Por cada pedido de conexión devuelve al cliente remoto la string "Conexión aceptada". A continuación ejecuta una demora de 60 segundos, y vuelve a esperar conexión. Termina cuando el usuario ejecuta CTRL-C.

Cliente:

Conecta con el servidor en el port indicado. Al recibir la string de conexión aceptada la presenta en pantalla y finaliza su ejecución.

Ejercicio 8.2.Concepto de conexión aceptada

Ejecute en consolas separadas los programas del 8.1. Abra una consola adicional y ejecute el comando netstat con las opciones adecuadas. Documente el comportamiento del servidor y de los clientes en función de ejecutar múltiples llamadas desde diferentes clientes.

Ejercicio 8.3.Servidores concurrentes

Modifique el servidor del 8.1 para que ejecute en forma concurrente. Una vez hecho esto, repita la experiencia del 8.2 y documente las diferencias en el comportamiento.

Ejercicio 8.4.Combinando TCP con UDP

Escriba un servidor concurrente que por cada pedido de conexión que le ingresa por el port TCP 3456, cree un proceso child. Cada instancia child buscará un port local UDP libre a partir del port 10000. Cuando lo encuentra lo informa al cliente por el port TCP heredado, y queda esperando información por el port UDP,

Condición de finalización: Recibir por el port TCP heredado la string "FIN" por parte del cliente remoto. Los datos recibidos por el port UDP se envían al padre mediante el IPC que Ud. prefiera.

Ejercicio 8.5. Combinando TCP con UDP e IPC's

Escriba el código de la siguiente pareja Cliente - Servidor

Servidor:

Ejecución en forma concurrente. Limita inicialmente a 10 instancias child.

Espera conexiones por el port TCP 2233. Por cada conexión crea un child que lee el port UDP 6252 por el cual se reciben logs remotos. Cada child graba los logs recibidos en el archivo /home/tdiii/log. El archivo no puede ser accedido en forma concurrente por cada child de modo que se requiere sincronizar de alguna manera el acceso. Utilice un semáforo para tal fin.

Condición de finalización de los procesos child: Cada child debe recibir cada 30 segundos por el port UDP 5263 un mensaje de control que consiste en la string "sigo vivo". Transcurridos 2 minutos sin recibir este mensaje, cierra el socket UDP y termina su ejecución.

Condición de finalización del proceso principal: 1 minuto sin childs activos y sin pedidos de conexión entrantes (las dos condiciones).

Cliente :

Se ejecuta mediante el siguiente comando: sender [argumento], en donde ***sender*** es el nombre del programa ejecutable, y ***argumento*** es un comando cualquiera de LINUX (por ejemplo ls –las, ps -ef, etc)

El programa debe conectar con el servidor anteriormente definido al port TCP 2233, ejecutar mediante la llamada al sistema adecuada el comando, previo generar la redirección requerida de modo tal que la salida que normalmente se produce por stdout salga por un socket para enviarla al port UDP 6252 del server definido anteriormente.

Finalizada la transmisión esperará dos eventos: Un nuevo comando por teclado de parte del usuario para repetir la operatoria, y un timer de 30 segundos para enviar "Sigo vivo" al port UDP 6253 del servidor.

El cliente termina si el usuario pulsa CTRL-C en la línea de comandos.

Ejercicio 8.6. Sistema sencillo de DSP por red *Entrega Obligatoria*

Tome el programa del 6.11 modifíquelo para que se comporte como un servidor concurrente, que escuche conexiones por los puertos 8193, 8194 y 8195. De este modo cada función del DSP corresponde a un servicio accesible por el port correspondiente.

Por cada conexión debe crear un proceso child que abra un port UDP del mismo contra el port UDP remoto 8193, 8194 u 8195 de acuerdo con el servicio requerido. En este caso, en lugar de guardarla en un archivo, el child debe transmitir, por dicho port UDP la información procesada por las rutinas echo, fir_low_pass y delay.

Condición de finalización de cada child y del proceso principal: La misma establecida para el programa del 8.5.

Ejercicio 8.7.Sistema de chat

Desarrollar un sistema de Chat que cumpla con las siguientes condiciones.

Debe existir proceso "servidor" que reciba peticiones de conexión por el puerto TCP 9001 por cada cliente que quiera unirse al chat. Al aceptar la conexión deberá generar un proceso hijo por cada cliente que se conecta, estableciéndose de esta manera la sesión de chat del cliente.

Una vez establecidas la sesión de chat, el cliente podrá comenzar a enviar cadenas de texto (mensajes) que los procesos hijos recibirán; y mediante un adecuado mecanismo IPC se debe lograr que cada mensaje sea reenviado a todos los clientes que mantienen sesión excepto al cliente que envió el mensaje. (Se recomienda que el proceso padre lleve una tabla en memoria de los clientes conectados).

En los procesos cliente: cada vez que se recibe un mensaje de alguien del chat, se debe imprimir el nombre del host que inició la sesión (para que se sepa de quien se trata) y luego el mensaje, de la siguiente manera:

[HostName]:[mensaje]

Ejercicio 8.8.Análisis de secuencia de comunicaciones.

Se tiene la red del grafico con los siguientes datos :

PC :

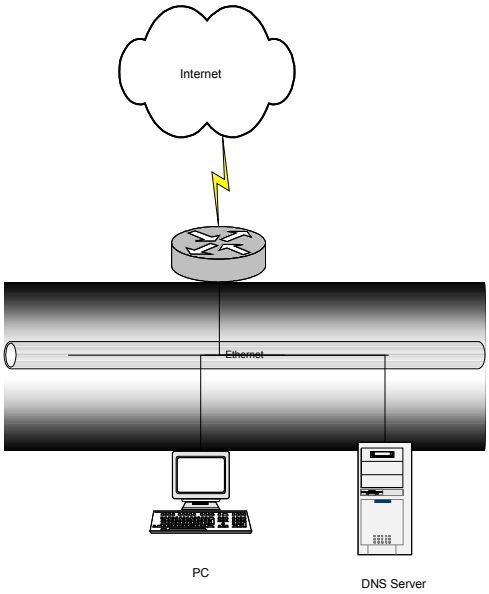
Dirección IP = 192.168.0.1
Dirección MAC = 00-00-DA-1E-B0-CA
Subnet Mask = 255.255.255.0
Default Gateway = 192.168.0.2
DNS = 192.168.0.3

DNS Server :

Dirección IP = 192.168.0.3
Dirección MAC = 00-00-B1-BA-B0-CA
Subnet Mask = 255.255.255.0
Default Gateway = 192.168.0.2

Router :

Dirección IP LAN = 192.168.0.2
Dirección MAC = 00-00-B0-CA-19-05
Subnet Mask = 255.255.255.0
Dirección IP WAN = 200.69.237.23



Grafique todos los paquetes enviados o recibidos por PC cuando se levanta un web browser y se ingresa como dirección: http://www.frba.utn.edu.ar suponiendo que la PC no tuvo ninguna comunicación previa

1.¿Que se necesita en el router para que la comunicación se pueda establecer?

2.¿En que difiere el intercambio de paquetes si se utiliza http 1.0 o http 1.1?

3.¿Que alternativa tiene para permitir el acceso a navegación web si no se configurara nada en el router?

4.A continuación, desde el mismo browser se accede a www.nic.ar. ¿En que difieren los paquetes recibidos y enviados por PC?

Ejercicio 8.9.Trabajo a nivel de paquetes. Uso de Libpcab

Escriba un programa para Linux que funcione como un Sistema de detección de intrusiones básico (IDS: Intrusion Detection System). Un primer programa deberá capturar paquetes sobre la tarjeta ethernet, almacenarlos en un archivo para su posterior lectura. Un segundo programa deberá correr cada 1 minuto buscando las siguientes anomalías en los paquetes capturados:

1-Paquetes fragmentados

2-Paquetes TCP syn con datos enviados

3-Conjunto incorrecto de flags TCP (ningun flag, syn + fin, syn + rst, syn + fin + rst, paquete de datos sin three way hadshake previo)

4-Direcciones IP invalidas (RFC 1918, Clases D y E y loopback)

http://www.ietf.org/rfc

Cuando se detecten esas anomalías deberán enviarse por pantalla

Ejercicio 8.10.Trabajo a nivel de paquetes. Uso de Libpcab

Escriba un programa para Linux que funcione como network scanner. Su función será la de detectar servicios ofrecidos por un host sobre protocolo TCP, (puertos bien conocidos). El programa recibirá como primer parámetro direcciones IP en formato:

1-W.X.Y.Z, solo una dirección IP

El programa deberá listar todos los servicios disponibles por nombre cuando sea posible en cada host escaneado.

T.P. Nº 9.Casos de Proyecto

Ejercicio 9.1.

Se requiere un pequeño sistema multitasking, que cargue en la memoria de la PC aplicaciones que le son transmitidas por el port serie COM1, y lance su ejecución a partir de la dirección 200000h de memoria RAM.

Para ello el dispositivo serie debe bajar a la memoria RAM las aplicaciones que le llegan desde el extremo remoto, crear los descriptores necesarios en las tablas que corresponda, y anexar la aplicación a la lista de procesos en ejecución por el scheduler del sistema. El formato de esta lista queda a su criterio.

Las aplicaciones recibidas contienen en primer lugar, el código completo de la aplicación, seguido de las variables de memoria necesarias. El esquema de direccionamiento empleado en el modelo de programación de estas aplicaciones es relativo a la base del bloque de datos que contiene código y variables, de modo tal que se requiere definir por cada una un segmento de código y otro de datos de igual tamaño y dirección base. El primer byte de la aplicación corresponde a la primera instrucción a ejecutar.

En función de esto, escriba el código de un programa en lenguaje ensamblador que cumpla los siguientes requerimientos:

- Configurar al sistema completo para trabajar en Multitasking.
- Controlar por Interrupción de Hardware tanto la recepción como la transmisión por el dispositivo serie COM1.

2.a.Por el dispositivo serie se reciben datos con formato. Los dos primeros bytes representan en hexadecimal el tamaño del bloque de datos, que se recibirá a continuación, y que contiene una aplicación a descargar en memoria RAM. **En este punto** se requiere chequear si existe espacio suficiente en la memoria RAM instalada en el sistema para descargar la aplicación y si existen descriptores disponibles en la GDT para crear la tarea. Si la comprobación es positiva se envía al extremo remoto el código "Ready" para que comience con la transmisión de la aplicación, y en caso contrario, el mensaje de error correspondiente (ver ítem subsiguiente para detalles).

2.b.Por el dispositivo serie se transmiten al extremo remoto los códigos de error, o "Ready" de acuerdo con las comprobaciones que se definieron en el ítem anterior:

Valor	***Tipo de dato***	***Significado***
'0'	Byte ASCII	Listo para Recepción
'1'	Byte ASCII	El Sistema no tiene espacio en RAM para cargar la aplicación
'2'	Byte ASCII	El Sistema ha llegado a su límite de tareas máximo

Revisar ***periódicamente*** si se completó la recepción de una aplicación por parte de la puerta serie, y en tal caso generar en la GDT todos los descriptores necesarios para poder ejecutarla en el entorno multitasking, ***inicializados con los valores correspondientes*** e incorporar la nueva tarea a la lista de tareas activas del proceso scheduler.

Ejercicio 9.2.

Para un equipo configurado con dos placas de red, y si cada placa tiene, como es de esperar, una IP de redes distintas (correspondientes a las interfaces eth0=192.168.0.1/24 y eth1=192.168.1.1/24) y utilizando la posibilidad de configurar dos IP en la misma NIC, se requiere desarrollar un router que permita distribuir los paquetes que le lleguen de cada red a cualquiera de las otras tres. Utilizar la libpcap y raw sockets.

Ejercicio 9.3.

Tome el programa desarrollado en el 8.5 y modifíquelo para que procese digitalmente la señal de audio leída en /dev/dsp. Ahora se pretende procesar muestras de 16 canales stereo.

Modificación a la Operatoria:

El cliente enviará por el port UDP 9090 un requerimiento con el siguiente formato:

Tamaño (bytes)	3	1	2	2	2	.....	2
Campo	**'FIL'**	n	a_0	a_1	a_2	.....	a_n
Descripción	*Filtrado digital*	*Orden*	└--------- *coeficientes* ---------┘				

Func. transferencia

$$y(n)=\sum_{k=0}^{9} h(k)*x(n$$

Los valores de los ***coeficientes*** están normalizados a formato punto fijo 0.16, y su cantidad es la especificada en el campo ***Orden***.

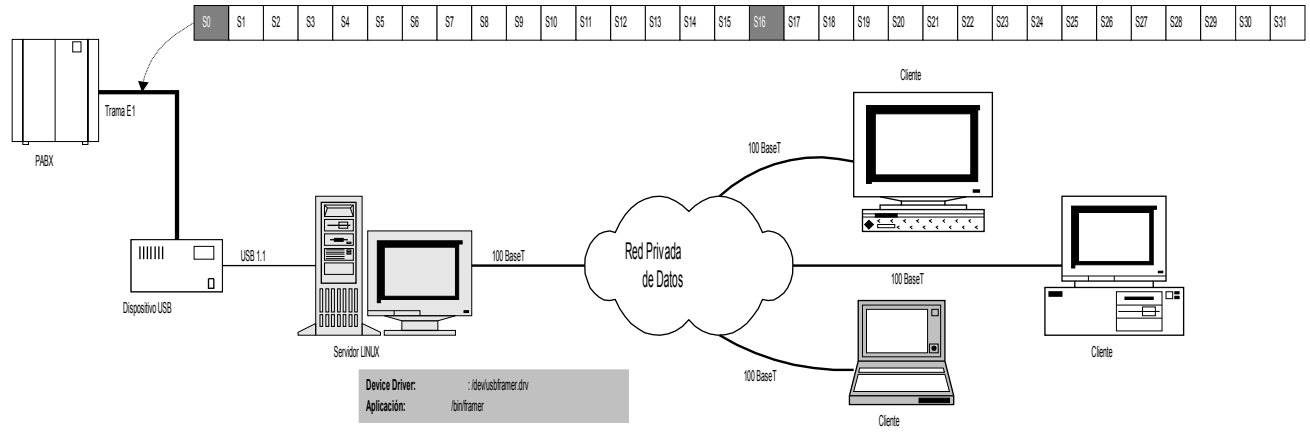
Cada instancia child será reemplazada por un proceso ***/$HOME/dsp*** que se encargará de esperar un comando con el formato indicado, y una vez recibido procesar el requerimiento del cliente.

dsp leerá el comando, dimensionará un buffer circular con los coeficientes y otro en el que tendrá las últimas ***n*** muestras, siendo ***n*** el valor recibido en el campo ***Orden*** del paquete TCP. Posteriormente comenzará a leer las muestras tal como lo hizo en el programa del 8.5. Por cada lectura recibe un par de números de 16 bits (canal derecho y canal izquierdo), correspondientes a las muestras adquiridas, los normaliza dividiéndolos por el valor de fondo de escala para llevarlos al formato 0.16, y les aplica la función transferencia correspondiente. Los valores de la salida se envían por el port UDP 9090 por el que se recibió el pedido del cliente remoto.

Se pide, además desarrollar el proceso cliente.

Ejercicio 9.4.

Se tiene el siguiente sistema de conversión de una trama E1 saliente de la PABX a IP plano:



El dispositivo USB entrega los 30 bytes útiles de cada trama (uno por cada canal (S1 a S15 y S17 a S31). Cada 16 tramas entrega la información de control de los canales en otro paquete de 30 bytes, uno por cada canal de información. Los aspectos de la señalización de la trama E1 son resueltos por el dispositivo USB y no forman parte del problema a resolver. El dispositivo regresa un byte para cada Time Slot con un 00h si el canal está libre o 55h si está ocupado. En función de esta información deberá tratarse el contenido de cada canal de la trama recibido previamente.

La información que provee el dispositivo USB se obtiene de un driver vendor specific instalado en el sistema que se accede mediante el nodo **/dev/usbframer**, por medio de las funciones de acceso a los device drivers standard de Linux.

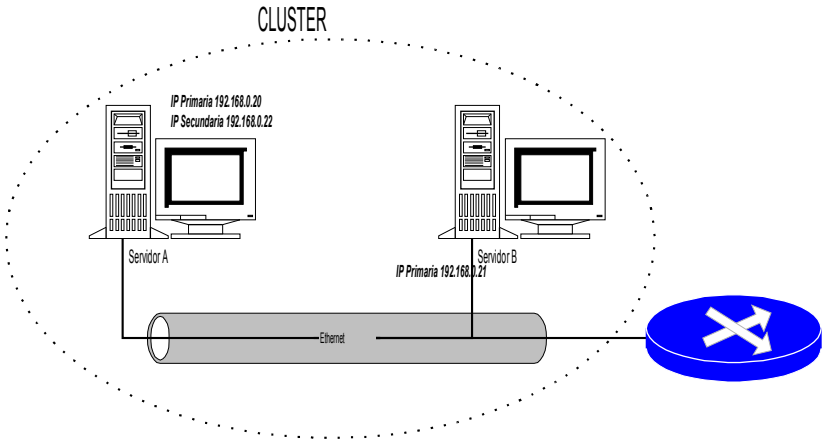
El servidor se encuentra conectado a una red de datos. La aplicación **/$HOME/framer** provee la transmisión de los canales de información correspondientes a los time slots S1 – S15 y S17 – S31. Para tal fin escucha requerimientos por los ports TCP 15001 a 15015 y 15017 a 15031 respectivamente. De este modo se responde por cada port con la información del Time Slot correspondiente. Los requerimientos se responden por el mismo port que se reciben.

Se pide que desarrolle el programa completo de la aplicación **/$HOME/framer**, que realice las siguientes tareas:

- ❑Escuchar los pedidos de servicio por los ports TCP especificados.
- ❑Resolver cada pedido por medio de una instancia child específica, que mantendrá la conexión con el cliente.
- ❑Cada instancia child activa termina cuando el cliente se desconecta.
- ❑Acotar a no más de 200 procesos hijos las sesiones posibles. Deberá denegar la conexión llegado al límite.
- ❑Tomar la información que provee el dispositivo USB, y distribuir el Time Slot a las instancias child activas que lo están enviando a los diversos clientes, intercomunicando los procesos padre y child como estime mas adecuado.

Ejercicio 9.5.

Se tiene un Cluster de 2 servidores Linux, dispuestos de acuerdo con el siguiente diagrama:



Ambos tienen idéntica configuración y conforman un Tandem Fault Tolerance. El servidor A es quien arranca activo (default) y el Servidor B está como Backup. Cuando el Servidor A deja de trabajar por el motivo que fuese, B toma el control. Continúan así hasta que B falle.

El principio de funcionamiento se basa en la posibilidad de definir sobre la interfaz LAN eth0 de Linux múltiples direcciones IP. La IP primaria es la que utilizan los servidores como propia para ser accedidos en la LAN en forma específica. La IP

secundaria trabaja como la dirección IP del cluster para el resto de la red independientemente de cual de los dos servidores esté controlando el sistema. Es decir se accede al servicio (independiente del equipo) por medio de la IP secundaria del Servidor activo. ***Solo uno de los servers debe tener configurada la IP secundaria en un mismo momento***.

El servidor backup deberá configurar la IP del cluster como su IP secundaria en el momento en el que toma el control del sistema.

Ver en el Apéndice como se configura la IP secundaria.

- En cada servidor se ejecuta un servicio llamado ***guardian***, que se comunica con su par del otro equipo utilizando el port UDP 5555. Solo reciben mensajes emitidos en origen por ese mismo port. A continuación se detalla la operatoria de ***guardian***:

- Cuando recibe START por el port 5555, levanta la aplicación /bin/aplicac que ejecutará reemplazando a una instancia child de ***guardian***.

- La instancia de /bin/aplicac envía a ***guardian*** un signo de actividad por medio de la señal SIGUSR1 cada 50 mseg. Si expira este plazo sin haber recibido dicha señal, ***guardian*** termina la ejecución de /bin/aplicac. Acto seguido elimina su IP secundaria y envía por el port 5555 el mensaje ***START*** al otro servidor. Si recibe normalmente la señal ***SIGUSR1***, se comunica con su peer en el otro servidor y envía el string OK.

- Si transcurren 50 mseg sin recibir desde el lado activo el OK, envía tres veces consecutivas el comando ***SYNC*** a intervalos de 10 mseg. Si no recibe respuesta (otro comando ***SYNC***), se activa sólo, y loguea por el port TCP 12345 "CLUSTER ERROR".

- Cuando recibe ***SWITCH*** por el port 5555 configura su IP secundaria como 192.168.0.22

- Si recibe ***STOP*** por el port 5555, detiene la aplicación. Si recibe KILL la termina.

Se pide desarrollar el programa ***guardian***. Considere que las demoras se ejecutan al mismo tiempo que la espera de mensajes por el port UDP. Ambos procesos bloquean al programa. Resuelva esta situación como considere más conveniente.

Apéndice

La Interfaz eth0 está configurada en /etc/sysconfig/network-scripts/ifcfg-eth0. El formato de ese archivo es el siguiente:

```
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.0.21
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
```

```
TYPE=Ethernet
USERCTL=no
NETWORK=192.168.0.0
BROADCAST=192.168.0.255
PEERDNS=no
```

Para definir una IP secundaria se necesita simplemente copiar este archivo con el nombre /etc/sysconfig/network-scripts/ifcfg-eth0:2 y en éste reemplazar la dirección IP del campo **IPADDR** por la IP definida como secundaria.