Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Departamento de Electrónica

Cátedra: Técnicas Digitales III - Plan 95A

GUIA DE TRABAJOS PRACTICOS

Ciclo Lectivo 2007

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 o grupo 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. La elaboración del Ejercicio será individual o grupal según se indique.

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.

La entrega puede realizarse en medio magnético personalmente a los docentes, o vía e-mail. **Acompañando a la entrega electrónica de la información se requiere una copia en papel.**

Para los prácticos de hardware, los circuitos deben estar realizados en ORCAD, o cualquier herramienta similar, y su presentación se requiere en medio magnético, e impreso en papel.

T.P. Nº 1.Decodificación de Hardware en el Bus ISA.

Ejercicio 1.1.Decodificación parcial en espacio de prototipo

Dibuje el circuito de una placa para el bus ISA de una PC, que contenga un port paralelo de entrada y otro de salida, ambos de 16 bits, accesibles mediante las 16 líneas de datos de la interfaz ISA (D_{00} a D_{15}), a partir de las direcciones 300h y 310h respectivamente sin dejar imágenes.

Construya la lógica de decodificación utilizando un solo elemento lógico.

Construya los dos ports a partir de componentes de lógica TTL.

Ejercicio 1.2.Interfaz de Adquisición de locación para PC por Bus ISA *Entrega Obligatoria grupal*

Para un sistema compuesto por un GPS de la familia (GM-83) conectado a un microcontrolador 8051 y este último conectado a una PC en un slot ISA de 8 bits, utilizando una línea de interrupción disponible. Se pide:

1. Definir el modelo de la familia de GPS más conveniente para el sistema.

2. Dibujar el circuito completo de la placa de interfaz con el GPS elegido, el microcontrolador y la lógica de decodificación necesaria para mapear sin dejar imágenes, utilizando el espacio de entrada salida reservado para prototipos. Asigne las direcciones que considere convenientes.

3. Explique la función de las señales que emplea en la lógica de decodificación en el bus ISA.

Entregables: Plano del circuito en ORCAD

HOLUX GM-83 Engine board: http://www.holux.com

Interfaz de Adquisición de señales Analógicas por Bus ISA

Para el diseño de un sistema de adquisición de audio estereofónico de alta calidad se desea conectar en un slot ISA de 8 bits de una PC, dos conversores A/D de 16 bits cada uno con su correspondiente circuito de sample and hold en su entrada de señal, y un port paralelo para status y comandos. Queda a su criterio definir la estructura de bits de este registro.

Se pide:

1. Seleccionar el conversor A/D que considere apropiado para esta aplicación pensando que el ancho de banda requerido a dispositivos de audio de alta calidad es de 20Hz. a 20KHz. Justificar la respuesta.
2. Dibujar el circuito completo de la placa de interfaz con los conversores, los circuitos de sample and hold, el port paralelo, y la lógica de decodificación necesaria para mapear todos los dispositivos sin dejar imágenes, utilizando el espacio de entrada salida reservado para prototipos. Asigne las direcciones que considere convenientes.
3. Manejar la señal SOC de los conversores A/D, de modo de manejar la frecuencia de adquisición de datos desde la aplicación, así como el envío al procesador de la PC de una señal de interrupción para la obtención de los datos. La placa interrumpirá una vez producida la señal EOC en ambos conversores.
4. Explique la función de las señales que emplea en la lógica de decodificación.

Entregables: Plano del circuito en ORCAD para el punto 2. y 3. Documento en formato .DOC con las respuestas a 1. y 4.

T.P. Nº 2.Universal Serial Bus.

Ejercicio 2.1.Sistema de Adquisición de señales por USB *Entrega Obligatoria grupal*

Se desea implementar un sistema de adquisición de datos de 8 canales de audio de alta fidelidad (20Hz. a 20KHz.), muestreando a 16 bits, en un dispositivo periférico portable. Por tal motivo, se establece como condición de diseño que se conecte a la PC a través del USB.

Se piensa en dos alternativas:

I. Utilizar un controlador USBN9603 stand alone con un microcontrolador a su elección para el control del periférico.

II. Utilizar un Cypress AN2131 compuesto por un core 8051 y un USB node controller embebido.

Se pide:

1. Establezca el ancho de banda que consumirá su dispositivo del USB para cumplir los requisitos solicitados para los 8 canales de audio.
2. Seleccione y dimensione apropiadamente la cantidad y tipo de Endpoints que va a utilizar para transmitir los 8 canales estereofónicos a la PC, y en función de esta respuesta dimensione el tamaño del paquete de datos a transmitir. Justifique detalladamente.
3. En función de los resultados de los análisis de ancho de banda en el USB y volumen de información a transmitir por segundo, estudie y justifique la factibilidad de aplicación de las dos alternativas planteadas al principio.

Recomiende la que considere mas apta. Justifique la razón de su recomendación.

3.a. En el caso de elegir la Alternativa I, seleccione el Microcontrolador o Microprocesador que considere más apropiado para la aplicación requerida. Justifique su respuesta, aportando datos concretos de la CPU adoptada.

3.b. En el caso particular de recomendar la Alternativa II, justifique la aptitud de la CPU para cumplir los requisitos establecidos en el sistema.

4. Dibuje el diagrama de conexionado completo del dispositivo USB: Microprocesador, Controlador USB y Lógica Adicional, para la Alternativa I.

Entregables: Plano del circuito en ORCAD para el punto 4. Documento en formato .DOC con las respuestas a 1, 2 y 3.

Ejercicio 2.2.Descriptores y aplicaciones *Entrega Obligatoria grupal*

Para la alternativa seleccionada en el 2.1 :

1. Escriba la rutina de inicialización del procesador y del controlador USB. Incluya los descriptores de Dispositivo, Interfaz, y Endpoint en el código. Incluya para cada campo relevante del descriptor un comentario justificando la adopción de los valores asignados. El lenguaje de programación queda a su criterio

2. Desarrolle una aplicación, que utilizando el driver standard del sistema operativo, correspondiente a la clase de dispositivo definido en el descriptor de dispositivo, recupere los 8 canales y guarde su información respectiva en un archivo individual para cada canal.

Entregables: Programas fuente en formato electrónico y listado en papel.

T.P. Nº 3.IA-32 - Modo Protegido.

Ejercicio 3.1.Entrada a Modo Protegido

Escriba un programa que ponga al procesador en modo protegido, seguidamente ponga la pantalla en modo de video inverso, y retorne al modo real para devolver el control al Sistema Operativo base, dejando el sistema en perfectas condiciones de funcionamiento.

Asuma un controlador de video color.

Ejercicio 3.2.Manejo de Interrupciones en Modo Protegido

Agregue al programa desarrollado en el 3.1 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 para retornar al modo real.

Ejercicio 3.3.Ordenamiento de las Interrupciones

Tome el programa del 3.2. 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. Debe restituirlos al regresar a modo real antes de devolver el control al sistema operativo.

Ejercicio 3.4.Manejo de Excepciones en Modo Protegido

Tome el programa del 3.3 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 llame a la porción de código que retorna al modo real devolviendo el control al Sistema Operativo base.

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 3.5.Cuenta de memoria RAM en el sistema

Agregue al programa del 3.4 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 3.6.Programa de arranque

Se requiere desarrollar un programa que sea apto para residir en el boot sector de un disquete. Este programa debe poder 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 programa de arranque pedido debe realizar las siguientes funciones:

- Poner el procesador en Modo Protegido
- Habilitar el gate A20
- Armar una GDT con 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. La GDT debe colocarse a partir de la dirección 100000h.

- Pasar a big real mode.
- Leer el Directorio Raíz del disquete y ubicar la entrada del archivo kernel.bin
- Leer los sectores correspondientes al archivo kernel.bin utilizando las rutinas disponibles en el BIOS a partir del primer cluster de la entrada del Directorio Raíz y los subsiguientes clusters obtenidos a partir de la FAT.
- Copiar el contenido de kernel.bin a la memoria RAM del sistema a continuación de la GDT.
- Pasar el procesador a modo protegido.
- Saltar al inicio de kernel.bin

En caso de entrega: Una imagen de disquete para Bochs. Listado del programa fuente en papel. Programa fuente en formato electrónico.

Ejercicio 3.7.Manejo de tareas simple – scheduler como tarea

Tome el programa del 3.4, y agregue el código necesario para administrar la presentación de información en forma simultánea en las mitades superior e inferior de la pantalla, utilizando al timer tick de la PC como base de tiempos.

La presentación en cada mitad de la pantalla estará administrada por una tarea diferente.

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, re programe 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.

Entregables: Programas fuente en formato electrónico y sus listados en papel

Ejercicio 3.8.Manejo de tareas simple – scheduler como rutina de interrupción

Repita el 3.7 utilizando una puerta de interrupción en IRQ0.

Entregables: Programas fuente en formato electrónico y sus listados en papel

Ejercicio 3.9.Uso del debugger de Bochs para analizar comportamiento *Entrega Obligatoria individual*

Tome los programas de 3.7 y del 3.8, 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 3.7. Represente en base al comportamiento observado un diagrama de transiciones entre las diferentes tareas involucradas.

2. Idem para el 3.8

3. Como se comporta el bit Busy y el Bit NT para el programa del 3.7

4. Idem para el 3.8

5. Escriba un análisis 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 3.10.Ejecución en dos niveles de privilegio. *Entrega Obligatoria grupal*

Tome el programa del 3.7 o del 3.8 (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:aa y en Fila 17 Columna 35 en formato hh:mm:ss. Ídem Servicio Hora del Sistema.
- Retorno al modo Real.

Las tareas, descriptas se seguirán ejecutando de acuerdo al esquema de manejo de prioridades establecido en el 3.7 o en el 3.8 según haya sido su elección**.**

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

Entregables: Programas fuente en formato electrónico y sus listados en papel

Ejercicio 3.11.Carga y ejecución de un mini Kernel

Tome el programa del 3.10 y adáptelo (escribiendo todo lo posible en lenguaje C) de modo de compilarlo en formato binario puro en modelo de 32 bits, para que copiado en un disquete en el archivo kernel.bin pueda ser cargado en memoria como sistema de arranque con el programa loader obtenido en el 3.6.

Entregable: Una imagen de disquete para Bochs. Listado del programa fuente en papel. Programa fuente en formato electrónico.

T.P. Nº 4.Sistemas Operativos Multitasking

Ejercicio 4.1.Señales

Escribir un programa que cree un proceso hijo, que imprima un mensaje a fin de identificarse escribiendo su numero de proceso. El proceso padre al recibir ENTER por stdin le envía una señal SIGUSR1, el hijo acusa recibo y termina la ejecución. No deben quedar procesos en estado ZOMBIE (defunct).

Ejercicio 4.2.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 finaliza luego de 60 segundos.

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

SIGCHLD : Evitar la generación de programas zombies.

El proceso no debe crear más de n childs (mediante SIGUSR1), en donde n es el primer argumento recibido por línea de comandos. La cantidad de hijos se cuenta en una variable denominada ***Nchilds***.

Ejercicio 4.3.Procesos

Modifique el programa del 4.2, 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***"

Todos los procesos finalizan únicamente con SIGKILL. No incluyen el tiempo de expiración solicitado en el 4.2.

Ejercicio 4.4.Redirecciones

Para mejor visualización de los resultados modifique el programa del 4.3 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 4.5.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 4.6.System V IPC's

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 4.7.System V IPC y Señales *Entrega Obligatoria individual*

Tome el programa del 4.6. 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 4.8.DSP sobre Linux

Tome el programa del 4.7 y agregue las tres rutinas externas programadas en assembler utilizando las instrucciones SIMD de los procesadores IA-32.

Ejercicio 4.9.DSP sobre Linux *Entrega Obligatoria grupal*

Modifique el programa del 4.8 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 4.10.System V IPC's

Modifique el esquema de distribución de audio del programa del 4.9 utilizando una message queue.

Ejercicio 4.11.Threads *Entrega Obligatoria grupal*

Modifique el programa del 4.10 empleando Linux threads en lugar de crear procesos con fork().

Ejercicio 4.12.In Line assemby. *Entrega Obligatoria grupal*

Tome los programas del 4.9 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 4.13.Device Drivers *Entrega Obligatoria grupal*

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 5.Procesamiento Digital de Señales

Ejercicio 5.1.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 5.2.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 5.3.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:

$$Pr_{(x,y)} = \left[\frac{f \bullet Pa_{(x,y)}}{255} \right] + \left[\frac{(255 - f) \bullet Pb_{(x,y)}}{255} \right]$$

Donde $Pa_{(x,y)}$ y $Pb_{(x,y)}$, son respectivamente los píxeles correspondientes a las coordenadas (x,y) de ambas imágenes, $Pr_{(x,y)}$ 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 5.4.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 5.5.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 5.6.Procesador de imágenes. *Entrega Obligatoria individual*

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 5.2, 5.3, 5.4, y 5.5.

T.P. Nº 6.Redes de datos.

Ejercicio 6.1.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 acaptada la presenta en pantalla y finaliza su ejecución.

Ejercicio 6.2.Concepto de conexión aceptada

Ejecute en consolas separadas los programas del 6.1. 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 6.3.Servidores concurrentes

Modifique el servidor del 6.1 para que ejecute en forma concurrente. Una vez hecho esto, repita la experiencia del 6.2 y documente las diferencias en el comportamiento.

Ejercicio 6.4.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 6.5.Combinando TCP con UDP e IPC's *Entrega Obligatoria grupal*

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 6.6.Sistema sencillo de DSP por red *Entrega Obligatoria grupal*

Tome el programa del 4.9 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 6.5.

Ejercicio 6.7.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 te 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 6.8.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

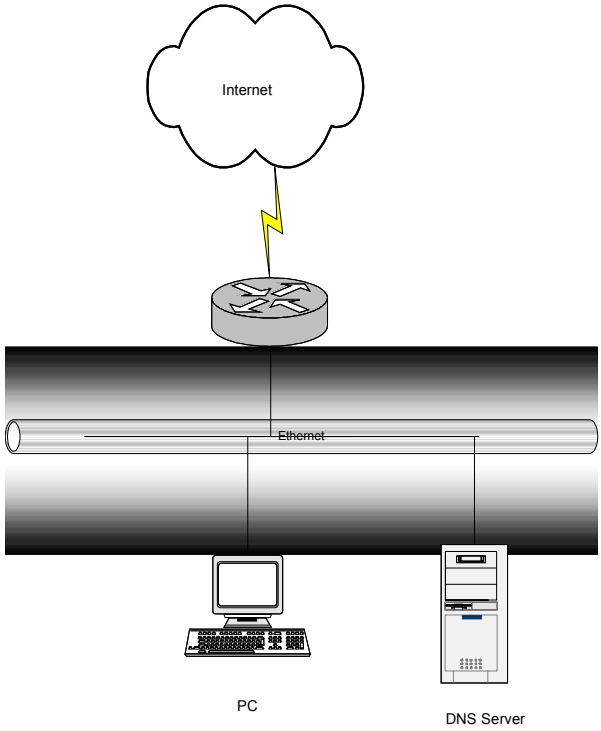
Router :

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



1. 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
2. ¿Que se necesita en el router para que la comunicación se pueda establecer?
3. ¿En que difiere el intercambio de paquetes si se utiliza http 1.0 o http 1.1?
4. ¿Que alternativa tiene para permitir el acceso a navegación web si no se configurara nada en el router?
5. A continuación, desde el mismo browser se accede a www.nic.ar. ¿En que difieren los paquetes recibidos y enviados por PC?

Ejercicio 6.9.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)

Cuando se detecten esas anomalías deberán enviarse por pantalla

Ejercicio 6.10.Trabajo a nivel de paquetes. Uso de Libpcab

Entrega Obligatoria grupal

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 7.Casos de Proyecto

Ejercicio 7.1.

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:

1 Configurar al sistema completo para trabajar en Multitasking.

2 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

3 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 7.2.

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 7.3.

Tome el programa desarrollado en el 6.5 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 $y(n) = \sum_{k=0}^{9} h(k) * x(n-k)$

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

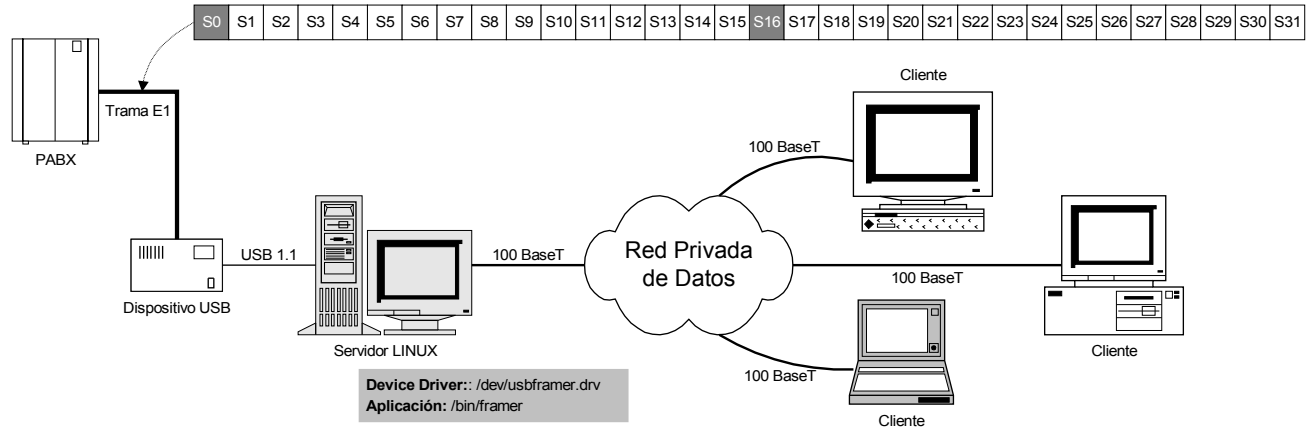
Cada instancia child será reemplazada por un proceso ***/$HOME/dsp*** que se encargará de esperar un comando con el formato indicado, y una vez recibido procesar el requerimiento del cliente.

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 6.5. 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 7.4.

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.

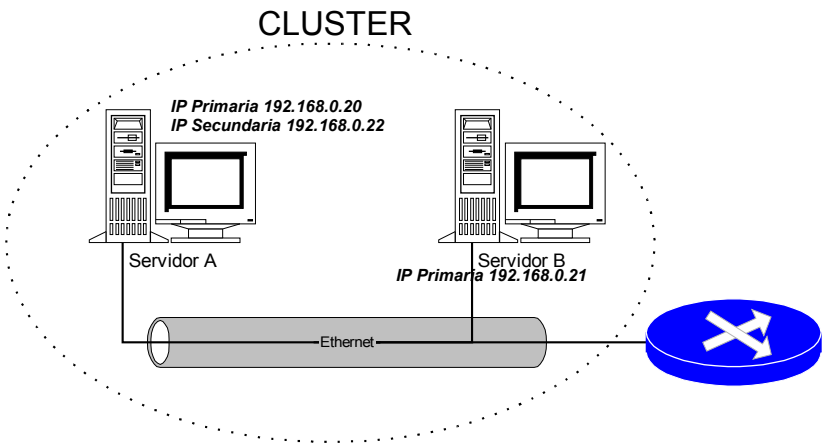
El servidor se encuentra conectado a una red de datos. La aplicación **/$HOME/framer** provee la transmisión de los canales de información correspondientes a los time slots S1 – S15 y S17 – S31. Para tal fin escucha requerimientos por los ports TCP 15001 a 15015 y 15017 a 15031 respectivamente. De este 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 7.5.

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.