Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Departamento de Ingeniería Electrónica

Técnicas Digitales III

Guía de Trabajos Prácticos

Ciclo lectivo 2014 **Plan 95A**

– Esta página fue dejada intencionalmente en blanco –

CONTENIDO

GENERALIDADES

1. 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 comprensión de los diferentes conceptos teóricos y su aplicación práctica efectiva, desarrollando en el alumno un enfoque metodológico y sistémico que le permita resolver problemas de Ingeniería utilizando como herramientas los diferentes componentes digitales, subsistemas y sistemas abordados a lo largo del ciclo lectivo.

El grado de complejidad se irá incrementando a través de los diferentes ejercicios planteados para cada Unidad Temática.
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, 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, considerándose en consecuencia, **no aprobado** dicho práctico.

En el caso de esta guía de Trabajos Prácticos la aprobación del 100% de los ejercicios, es una de las condiciones necesarias para la aprobación por promoción o firma de Trabajos Prácticos de la materia.

Más allá de las fechas establecidas como entrega, se recomienda hacer actualizaciones periódicas sobre el SVN del avance que vayan realizando sobre los Trabajos Prácticos a fin de demostrar la evolución en los mismos así como permitir un método de "back up" para los mismos.

2. Formato de presentación

Los archivos fuente deben tener en todos los casos los comentarios necesarios para clarificar su lectura, estar bien tabulados y mantener una rigurosa prolijidad para facilitar su lectura y seguimiento (revisar el archivo ***template.asm***).

Como encabezado del programa, debe haber un comentario que explique claramente la tarea que realiza, y las instrucciones detalladas (comandos) para su compilación y enlace o "linkeo".

Cada subrutina/función, debe también contar con un encabezado describiendo la operación que realiza, los parámetros que espera como entrada, y los resultados que debe presentar, indicando formato y método de entrega.

Se debe proveer como entregable: Los archivos fuente ***(.c, .asm***), el archivo "***Makefile***" o "***shell script***" con la secuencia de comandos necesarios para construir el programa mediante la simple orden ***make*** o ***./[shellscript]***.

3. Procedimiento de entrega de los Trabajos Prácticos

La entrega de los Trabajos Prácticos se realizará en primer lugar en un sistema de control de versiones del tipo SVN administrado por la cátedra.
El alumno utilizará para acceder a dicho repositorio el mismo usuario y clave que utiliza para ingresar en el Sistema de Gestión Electrónica (SGE) del Departamento.

El alumno subirá a este repositorio los ejercicios prácticos intermedios (se actualizarán los avances sobre **la misma** carpeta del ejercicio) que lo conducirán al entregable, con la frecuencia en que vaya dedicándose a las tareas de ejercitación previstas en la asignatura.
Por cada ejercicio de la guía deberá tener una carpeta separada dentro de su directorio con un nombre del tipo **ej1-4**, según corresponda.

Para mayores detalles sobre la configuración y uso de la herramienta se recomienda leer:

http://wiki.electron.frba.utn.edu.ar/doku.php?id=td3:uso_del_subversion_para_la_entrega_de_tp

La versión final del trabajo 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. Además, cada archivo fuente deberá estar correctamente documentado.

Importante:
Solo subir archivos fuente y Makefile. Evitar ejecutables, objetos y demás archivos innecesarios lo cual hace colapsar al software de versiones cuando se quiere acceder al repositorio SVN desde un cliente.

Aclaración:
No es necesario cambiar el nombre del TP cada vez que se sube una actualización.
Por ejemplo, para el ejercicio 1.4, subirá a la carpeta ej1-4 "pisando" el contenido anterior cada agregado que haga. El SVN automáticamente registrará los cambios entre versiones guardando TODAS ellas

La no presencia de la versión final completa del Trabajo Práctico en la fecha estipulada en el repositorio indicado se considerará ausente.

Solo bajo prueba fundada de indisponibilidad del servicio de internet de la Facultad se aceptarán entregas vía e-mail, siempre que los docentes lo hayan autorizado expresamente por comunicación a través de la lista de correo de los diferentes cursos.

TRABAJOS PRÁCTICOS – PRIMERA PARTE

1. Modo Protegido

El presente trabajo práctico está pensado de manera incremental, de modo que cada ejercicio tomará como base el precedente, añadiéndole el código necesario para implementar las funcionalidades y modificaciones requeridas.

Notas:
- Utilice el número de ejercicio como parte del nombre del archivo fuente, por ejemplo para el ejercicio **3.1** use ***ej3-1.asm***
- Todos los ejemplos deben arrancar mediante el mecanismo de carga (***booteo***) establecido por la Cátedra.

1.1. Primer contacto con *modo protegido*

Escriba un programa que cumpla los siguientes requerimientos:

a) Establecer el procesador en modo protegido.

b) Leer la dirección de E/S 0x60 (buffer de teclado) a la espera del número 0x01 (código de escaneo de la tecla ESC).
¿Cómo considera el método utilizado desde el punto de vista del consumo de la CPU?

c) Finalizar su ejecución quedando en estado ***halted*** en forma permanente.
¿Qué sucede si no finalizó el programa?
¿Cómo debería resolverse si estuviera trabajando bajo un Sistema Operativo?
¿Podría finalizarlo con un código como el que sigue? ¿Por qué si o no? ¿Qué ventajas y/o desventajas tendría?

```
fin:
jmp     fin
```

d) ¿Cómo debe estar el flag de interrupciones del procesador?. ¿Por qué?

¿Puede un programa en modo protegido funcionar sin GDT e IDT? ¿Por qué? ¿En qué condiciones?

Notas:
- El programa no debe tener definida una GDT.
- Utilice el debugger para interpretar el funcionamiento.
- Indente y comente adecuadamente el código.
- No deje registros de segmento sin inicializar.
- Tenga en cuenta el modo de direccionamiento en modo real.

1.2. Inclusión de una GDT y manejo básico de segmentos

Agregar al programa del ejercicio anterior los siguientes ítems:

a) Agregar una GDT que contenga dos segmentos, uno para datos y otro para código, con las siguientes características:
DPL=00
Base=0x0
Tamaño: 4 Gbytes
¿Cuál es la diferencia entre tamaño y límite? ¿Qué se especifica en el descriptor?

b) Definir una pila dentro del segmento de datos e inicializar el par de registros **SS:xSP** adecuadamente. Definirla en forma dinámica de modo que pueda modificarse su tamaño y ubicación de manera simple.

c) Poner la pantalla de video modo texto en video inverso. Escriba para ello una rutina que será llamada desde el flujo principal del programa.

d) Realizar una función para imprimir en pantalla que respete el prototipo:

void print (char string_ptr, char columna, char fila, char color);

¿Puede utilizar la misma para los modos 16, 32 y 64 bits? (Tenga presente la ABI, registros, y modos de direccionamiento)

e) Finalizar su ejecución quedando en estado ***halted*** en forma permanente.

Notas:
- Ubicar la GDT en un archivo separado donde irá agregando estructuras de datos equivalentes (*ej.* ***structs.asm***).
- El video inverso se realizará con una llamada a una función ubicada en un archivo de funciones auxiliares (*ej.* ***utils.asm***).
- Ídem para la función de impresión en pantalla.
- El acceso a la pantalla se efectúa escribiendo en un área de memoria de 4000 bytes (**no 4 kB**) denominada ***Video RAM Buffer*** que se describe en la Figura 1.
- Los prototipos de funciones se definen en forma genérica usando notación en lenguaje C. El manejo de parámetros en ASM deberá ser acorde a la **ABI** del modelo en uso.

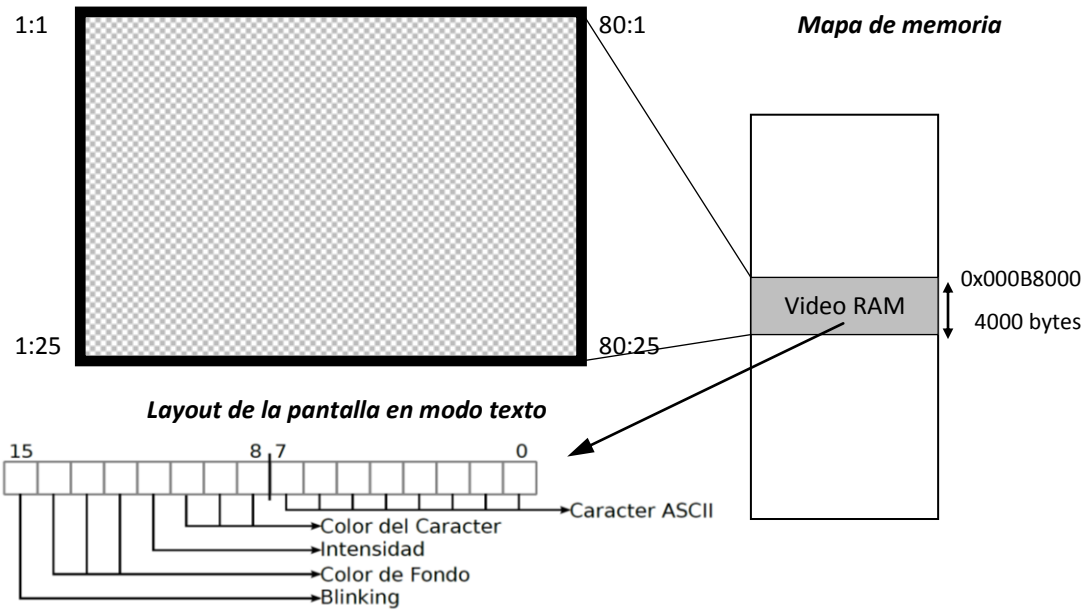


Fig. 1: Mapeo de la pantalla en la RAM de video y formato de cada Word

1.3. Habilitando paginación con prudencia: Identity Mapping

Tome el programa desarrollado en el ejercicio anterior, y active la unidad de paginación, armando previamente las tablas de traducción adecuadamente para que cada dirección lineal se traduzca en la misma dirección física (***identity mapping***).
Por el momento trabaje solamente con el primer megabyte de RAM.

Sugerencias:

- Para reducir la longitud del archivo binario inicialice las estructuras en forma dinámica.
- Su ubicación, tamaño y demás parámetros se especificaran mediante directivas ***"define"*** fácilmente modificables***.***

1.4. Habilitando PAE. Siempre en *identity mapping*

Modifique el programa del ejercicio anterior, habilitando PAE y manteniendo la traducción de la dirección lineal a la física en ***identity mapping***, para el primer megabyte de RAM.

Ubique las tablas de paginación en una zona segura, es decir en páginas independientes a las asociadas a los datos del programa, se recomienda a partir de la dirección ***0x7E00*** de forma de no utilizar direcciones reservadas para la BIOS.
Inicialice correctamente las entradas no utilizadas, verificando mediante Bochs, que no se visualizan tablas espurias.

Sugerencias:
- Defina la ubicación de las tablas de página mediante directivas "**define**" de modo que sean fácilmente desplazables.
- Compruebe siempre que los modos están disponibles antes de habilitarlos (vea **MSW** y la instrucción **CPUID**)

1.5. Inicialización en 64 bits. Modo Compatibilidad

Tome el programa del ejercicio anterior, y modifique lo necesario para satisfacer los siguientes requerimientos:

a) Habilitar **Gate A20** para acceder a la memoria por encima del primer Mbyte de RAM.
Pruebe escribir y leer en alguna dirección por encima del megabyte antes y después de habilitarla. ¿Qué nota? ¿Por qué?

b) Establecer al procesador en modo **IA-32e** (64 bits, sub-modo compatibilidad).

c) Organice el código fuente de manera que en el binario final, los datos, la pila y el código se encuentren alineados en direcciones múltiplos de 4kB diferentes, de manera que cada uno de ellos corresponda a una página diferente.

Notas:
- Debe realizar todas las comprobaciones necesarias antes de inicializar algún modo
- Revise la directiva "***align***"
- La función de habilitación de **Gate A20** será provista por la cátedra y deberá ubicarse junto al resto de funciones auxiliares.

1.6. Manejo sencillo de excepciones en 64 bits

En base al ejercicio anterior, agregue una tabla IDT capaz de manejar todas las interrupciones y excepciones del procesador, más las que estén reservadas, hasta el tipo 0x2F.

Además deberá:

a) Configurar el PIC maestro y esclavo de manera que utilicen el rango de tipos de interrupción **0x20-0x27** y **0x28-0x2F** respectivamente.

b) Inicializar el registro de máscaras, de modo que estén deshabilitadas todas las interrupciones de hardware en ambos PIC's.

c) Implementar cada una las rutinas de atención a excepción, de manera que impriman en pantalla el número de excepción generada y finalicen deteniendo la ejecución de instrucciones mediante la instrucción "**hlt**".

Notas:
- Ubicar la IDT en el mismo archivo que la GDT
- La configuración del PIC se hará mediante una llamada a función provista por la cátedra y ubicada en el archivo auxiliar de funciones previamente armado.
- Las rutinas de atención de excepciones se ubicarán en un archivo separado (**isr.asm**) y utilizarán la función imprimir ya realizada.

1.7. Interrupciones de hardware

Tome el código del ejercicio anterior y agregue los siguientes componentes:

a) Extender el mecanismo de traducción de páginas para que abarque, sin generar excepción de fallo de página (***#PF***) los primeros 2 Mbytes de RAM.

b) Efectuar una prueba de acceso con una dirección superior al primer Mbyte. Notar que si ***Gate A20*** no está correctamente habilitada el dato se escribe en la posición de memoria cuyos 20 bits menos significativos coinciden con la dirección utilizada para el acceso.

c) Reemplazar la secuencia de lectura de la dirección de E/S 0x60, por un controlador de **IRQ1** (teclado) que detecte que se ha presionado la tecla ESC, tras lo cual se debe establecer el procesador en estado **halted** en forma permanente. Recuerde que el procesador no debe quedar en ningún momento consumiendo el 100% de recursos.

d) Mover las tablas de paginación de manera que comiencen encima del primer megabyte.

Notas:
- Al leer el teclado tener en cuenta que se producen dos interrupciones, una por el ***make code*** y otra por el ***break code***.
- Toda sección de código que ya no utilice eliminarla para mayor claridad del entregable.
- Recuerde indentar y comentar **PERFECTAMENTE** todo el código.

1.8. Manejo avanzado de interrupciones

A partir del código del ejercicio anterior, ampliar la rutina de atención de **IRQ1** a fin de implementar las siguientes funcionalidades:

a) Al presionar la teclas **F5** a **F8** se le debe permitir al usuario habilitar o deshabilitar ***breakpoints*** prestablecidos en el código mendiante "**defines**". Se indicará en pantalla el estado de los mismos.

   ¿Qué permisos se requieren para habilitar un breakpoint? ¿Puede hacerlo? La dirección definida para el breakpoint es lineal ¿Qué sucede si la unidad de paginación no está activada? ¿Se puede hacer?

b) Detectar las combinaciones de teclas que se indican en siguiente tabla, las cuales deben efectuar operaciones para generar la excepción asociada a cada una:

CTRL+1	**Interrupción Tipo 7**	**#DF (Double Fault)**
CTRL+2	**Interrupción Tipo 13**	**#GP (General Protection)**
CTRL+3	**Interrupción Tipo 14**	**#PF (Page Fault)**

Importante:
- No es válido activar las rutinas de atención de las excepciones mediante la instrucción **INT n**, siendo **n** el tipo de la interrupción solicitada. La generación de la excepción debe ser producto de la ejecución de una o más instrucciones que generen las condiciones establecidas para la excepción.

1.9. Manejo de excepciones avanzado

A partir del ejercicio anterior, se pide adicionalmente los siguientes requerimientos:

a) En el flujo principal del programa se debe llamar a la función de generación de números pseudo-aleatorios provista por la cátedra, y mediante un factor de escala y ajuste, dar formato al número para convertirlo en una dirección lineal.
   Éste debe ser utilizado para intentar leer/escribir un dato de tamaño adecuado en memoria.
   Se pretende acceder a direcciones lineales generadas al azar **independientemente** de que éstas hayan sido o no asignadas al programa.

b) Implementar un controlador de excepción de fallo de página (**#PF**) que cumpla con la siguientes especificaciones:

   Ubicar las páginas correspondientes a la dirección virtual requerida por el sistema, en páginas físicas consecutivas a partir del primer megabyte de memoria física (luego del espacio reservado para las tablas de paginación), **independientemente del valor** de la dirección lineal generada por la unidad de segmentación (**NO** es identity mapping).

   Crear una nueva página de **4kB** cada vez que la aplicación intente acceder a una dirección no paginada.

c) El sistema se debe establecer en modo **halted** en forma permanente, cuando se agote la memoria física existente e informarlo por pantalla.

Para los fines del punto b) ¿Es lo mismo leer o escribir? ¿Trae alguna ventaja? Justifique
¿Puede establecer el procesador en modo **halted** desde una tarea? ¿Por qué?

Importante:
- Tener en cuenta que la dirección debe ser canónica.
- Toda función o código provisto por la cátedra **debe** ser analizado y entendido por el alumno.
- La memoria física existente se especificará mediante un **define** en 128 MBytes para los fines prácticos de este ejercicio, pero puede ser modificado y el programa debe funcionar perfectamente.
- No perder de vista que el código encargado de "leer/escribir" cumple la función de una tarea de **usuario**, mientras que el mecanismo de asignación de páginas corresponde al **sistema** (#PF), es decir la asignación de memoria y chequeo de disponibilidad **ES AJENO** al usuario y competencia **INTEGRA** del sistema operativo (en este caso el kernel).

1.10. Niveles de privilegio

Tomar el código del ejercicio anterior y modificar/agregar los siguientes ítems:

a) Implementar una GDT que contenga solamente descriptores de 64 bits, debiendo éstos disponer de PL=00 (**kernel**) y PL=11 (**usuario**), tanto para código como para datos.

b) Implementar en el segmento de datos una pila para las aplicaciones de usuario. Establezca ubicación y tamaño de forma coherente.
¿Es necesario tener dos pilas? ¿Por qué? ¿Con que características?

c) Modificar el código utilizado para acceder a memoria, transformándolo en una rutina de usuario y adecuar el esquema de paginación a tal fin.

d) La función que genera la semilla pseudo-aleatoria, debe ser accedida como un servicio y ubicarse en el segmento de código y página acorde con su privilegio.

e) Diseñe un mecanismo apropiado de acceso a la función aleatoria (API, wrapper, permisos, etc).

Notas:
- Explique la diferencia entre segmentos de datos de 32 bits y 64 bits ¿Tienen sentido?
- Utilice el vector **80h** para los servicios o bien un **FAR CALL**.
- Arme un ***wrapper*** para dar funcionalidad a los servicios.

1.11. Conmutación de tareas

Al código del ejercicio anterior, es necesario incorporarle una capacidad mínima de administrar tareas. Para ello se requiere, agregar las siguientes prestaciones:

a) Modificar el valor del temporizador 0 del PIT, para que genere una interrupción cada 1 mseg aproximadamente.

b) Implemente dos tareas de **usuario**. Una de ellas imprimirá la fecha de sistema y la otra la hora.
Ambas funciones serán provistas por la cátedra y deberán ser accedidas como servicios de kernel.
Elimine la llamada a la función de generación aleatoria de direcciones solicitada anteriormente.

c) Implementar un controlador para la interrupción 32 (**IRQ0: timer tick**). Este handler contendrá el servicio de administración de tareas (**scheduler**).

d) Diseñe el mecanismo de conmutación de tareas. Para esta instancia simplemente despachará una u otra tarea en forma alternada.

e) Considere al código que establece al procesador en estado ***halted*** como una tarea de ***kernel***.

Sugerencias:
- En esta instancia del desarrollo puede utilizar un **CR3** único para ambas tareas, si esto le facilita el desarrollo.

Notas:
- Establezca los niveles de privilegio correspondientes para cada tarea y estructura según corresponda.
- Utilizar la función de impresión en pantalla ya realizada.
- Los tiempos en bochs pueden no coincidir con los esperados. Recordar que es un emulador por ende no espere 1mseg real.

1.12. Paginación por cada tarea

En base al código del ejercicio anterior implementar una estructura de paginación independiente para cada tarea, a fin de que ambas instancias posean un adecuado marco de protección de sus áreas de memoria.

Tenga presente que determinadas áreas deben compartirse por todas las tareas. ¿Cuáles? ¿Por qué?

1.13. Mejoras en el scheduler. Lista de Tareas

Tomar el código del ejercicio anterior, y agregar los siguientes ítems:

a) Manejar las tareas mediante una lista de modo de independizar el código del ***scheduler*** de la cantidad de tareas a ejecutar y sus prioridades respectivas.

b) Relice una función **jiffies()** que devolverá la cantidad de milisegundos acumulados que tiene cada tarea en ejecución. Ubíquela en el archivo de funciones auxiliares.

c) Incorpore a las dos tareas de usuario que ya posee la función **jiffies()**.

d) Agregue dos tareas más que incluyan la llamada a **jiffies()**. **Todas** las tareas mostrarán en pantalla su identificador, prioridad y tiempo en ejecución. Las dos primeras mostrarán además hora y fecha como estaba especificado anteriormente.

e) Asignar a cada tarea de usuario un valor de prioridad, el cual equivale a la cantidad de ***ticks*** durante los cuales el ***scheduler*** no conmuta la tarea en curso. El rango de valores posible es de 1 a 20.

f) Mediante las teclas **F1** a **F4** y **CTRL+F1** a **CTRL+F4** se agregan o quitarán estas tareas de la lista de ejecución.

g) Mediante las teclas **q** y **w**, permitir cambiar la prioridad de la tarea 1 (entre 1 y 20) para comprobar su funcionamiento.

Sugerencias:

- En esta instancia del desarrollo puede utilizar varios CR3, uno por cada tareas para poder identificarlas.
- ***Idle*** será la tarea **0** y tendrá prioridad **0**. Al quitar todas las tareas ésta será la única en ejecución.
- Evite complicar el código desarrollando demasiado la presentación en pantalla. Complicara su fase de depuración y nuestra corrección.

Sugerencias:

- Utilice una estructura para el scheduler del tipo lista circular:

ID_Tarea	Prioridad	Prioridad_inicial
...	...	...
...	...	...

Con ***ID_Tarea*** en **0** puede indicar el fin de la tabla, de este modo es fácil incorporar nuevas tareas.
Prioridad_inicial será el valor a recargar en ***Prioridad*** (contador decreciente) y será el valor a modificar si se solicita modificar la prioridad de la tarea.

Notas:

- Utilizar la función de impresión en pantalla ya realizada.
- El sistema debe prever una cantidad variable de tareas (aunque solo use cuatro).
- Cada tarea debe tener su espacio de memoria independiente y con privilegios acordes.

1.14. Llamadas al sistema

Tomar el código del ejercicio anterior e incluir el código necesario para implementar la llamada a sistema:

void msleep(int milisegundos);

La misma duerme el proceso (la saca de la lista de ejecución) **inmediatamente**, la cantidad de milisegundos especificada en su argumento.

Incorporarla a las tareas con distintos tiempos y verificar su funcionamiento.

Notas:

- Utilizar demoras de entre 1 y 5 segundos para una correcta visualización.

1.15. Tareas con uso de instrucciones SIMD

Al sistema del ejercicio anterior, agregar dos tareas de usuario que utilicen registros **SIMD**, realizando el soporte necesario mediante la excepción 7 (***#NM***) y las modificaciones correspondientes en la función de cambio de contexto para resguardar el contenido de dichos registros.

Las tareas pueden realizar un simple movimiento de datos para verificar su integridad a través de los cambios de tarea.

Notas:

- Puede agregar el contexto **SIMD** a la estructura de datos de cada tarea
- Analizar si el procesador soporta el set de instrucciones **SIMD** y su versión.
- Revisar el uso de las instrucciones ***FXSAVE*** y ***FXRSTOR***
- Leer el capítulo 13.3 y subsiguientes del manual de Intel 3A
- Utilice el mecanismo provisto por Intel, no innove!

TRABAJOS PRÁCTICOS – SEGUNDA PARTE

2. Sistemas operativos multitarea

2.1. Procesos

Escribir un programa que cree procesos hijos. La cantidad máxima de estos procesos se recibe como primer argumento por línea de comandos en el momento de su ejecución (ejemplo: ***./ej2-1 15*** ejecutará **ej2-1** y le permitirá crear hasta **15** procesos hijos.

Cada proceso hijo generará una demora aleatoria entre 1 y 20 segundos luego de la cual finalizará su ejecución. Su inicialización y fin serán informadas por pantalla junto a su **PID**.

El proceso padre cada vez que termina un proceso creará uno nuevo de modo de mantener siempre ***n*** instancias ***child*** siendo ***n*** el primer argumento recibido por línea de comandos.
Los hijos que finalizan no deben quedar en estado ***EXIT_ZOMBIE*** (***defunct***)

¿Los PIDs generados, son consecutivos o distantes? ¿Por qué?

Notas:
- Chequear mediante los comandos **ps** o **top** su estado y evolución.

2.2. Señales

Escriba un proceso que reemplace los handlers default que le provee el kernel durante su creación para:

a) Ignorar los envíos de ***CTRL-C*** desde el teclado. De este modo no debe finalizar su ejecución mediante esta combinación de teclas.

b) Ignore la orden ***kill-15***, para terminar su ejecución.

c) Reemplace ***SIGUSR1*** por un handler que le permita solicitar por ***stdin*** la cantidad de procesos child a crear y los cree. Verifíquelo mediante una orden de consola.

d) Reemplace ***SIGUSR2*** por un handler que presente los pids de los childs que tiene en ejecución y permita ingresar por ***stdin*** el pid del proceso child a terminar y lo termine.

Al ejecutar el proceso original indique la forma en que debe terminarse el proceso ya que la mayoría de los medios de finalización estándar se habrán deshabilitado.

2.3. Pipes

Realice un programa que genere dos instancias child.

El proceso 1:
Recibirá por ***stdin*** strings
Los enviará a un archivo /var/log/chldmsg.log
Los enviará por un pipe al proceso 2

El proceso 2:
Imprimime en pantalla (***stdout***) el string recibido
Cuando recibe el string "FIN" termina.

Cada proceso deberá imprimir su PID y PPID.

Notas:
- Evite procesos zombies
- Evite procesos con consumo incontrolado de energía.

2.4. Threads

Realice un programa que genere un vector con números aleatorios, dispare dos threads y quede a la espera de su finalización.

Cada thread buscará un número especificado (uno desde el comienzo y el otro desde el final a fin de acelerar la búsqueda)
Éstos informarán la posición de la primera ocurrencia o cero si no lo encontraron.

¿Qué ventajas nota al usar threads en vez de procesos hijos? ¿Y desventajas?

Notas:
- El proceso principal esperará por los otros dos (***pthread_join***)
- E proceso que primero termine finalizará al otro (***pthread_cancel***)

2.5. Bloqueo de procesos

Escriba un proceso que espere al mismo tiempo strings, por una named PIPE **./tdiii_pipe**, e ingresos por el teclado. Por cada ingreso debe escribir en consola el mensaje precedido de "**[PIPE]:** ", o "**[KEYB]:** ", según su procedencia.
No debe perder datos de ninguna de las entradas.

Notas:
- Revise la función ***select()*** o ***poll()***

3. Procesamiento digital de señales

3.1. Acceso al dispositivo de audio

Escriba un programa que cree una instancia child que acceda al dispositivo de audio ***/dev/dsp***, y escriba las muestras en un buffer circular ubicado en una **shared memory**.

Otra instancia child leerá del buffer y lo enviará al parlante.

El dispositivo de audio se debe programar para entregar dos canales, cada uno con muestras de 16 bits y a una velocidad de 40 ksps. Estos parámetros deben encontrarse en "**defines**" para su fácil modificación.

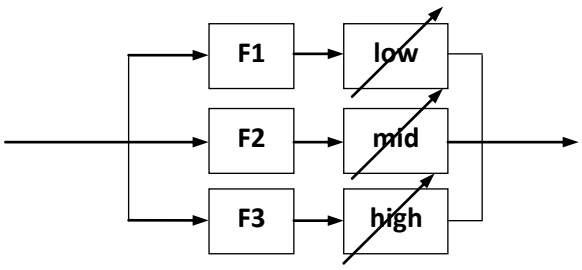
Notas:
- Utilice semáforos para controlar el acceso concurrente a datos.
- Tenga en cuenta que es un proceso en tiempo real. No deben sobre escribirse datos ni enviar la misma muestra más de una vez.

3.2. DSP: Procesamiento multimedia SIMD

Tome el ejercicio anterior y agregue la siguiente función:
void equalizer(char low, char mid, char high);

La función **equalizer()** estará compuesta por 3 filtros FIR de 32 taps cada uno que procesarán las muestras de entradas.
Los parámetros pasados corresponden al nivel para cada banda ajustable entre 0 y 255.

Las características de los filtros serán, respectivamente:
F1: fo= 2kHz BW= 3kHz
F2: fo= 4kHz BW= 3kHz
F3: fo= 6kHz BW= 3kHz



a) Realice el filtro íntegramente en C
b) Realice el filtro íntegramente con instrucciones SIMD
c) Compare los resultados de tiempos

Notas:
- Tenga en cuenta la ABI para evaluar la forma en que pasará los argumentos.
- Los coeficientes del filtro pueden calcularse mediante el programa **WinFilter8** provisto por la cátedra
- Utilice coeficientes de 8 bits

4. Redes de datos

4.1. 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 10 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.

Notas:
- Recuerde cerrar o liberar todo recurso no utilizado.

4.2. Servidor concurrente

Modifique el programa del ejercicio anterior para que se ejecute de forma concurrente.
Cada instancia de conexión deberá manejarse mediante un proceso separado, y deberá tener presente el uso de todos los recursos IPC vistos en clase que considere necesarios.

Notas:
- Puede comprobar el funcionamiento mediante la red entre varios compañeros

4.3. Streamer de audio simple

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.
Ambos procesos crearán una conexión UDP por medio da la cual el servidor enviará las muestras de audio obtenidas en el ejercicio 3.2.
En caso que el proceso cliente termine, el servidor deberá cerrar la conexión y recursos asociados.