Universidad del Valle de Guatemala
Sistemas Operativos
Docentes: Ing. Juan Luis Garcia Zarceño
Semestre I, 2022

Fredy Velásquez 201011

Laboratorio #5

Fecha de Entrega: 30 de marzo, 2022.

Descripción: en este laboratorio se programará y agregará una política de calendarización de CPU a un sistema Linux, con lo que se visitarán los componentes de su *kernel* involucrados en la calendarización de procesos, profundizando el ejemplo de clase sobre calendarización en Linux. Deberá entregar un archivo con respuestas a las preguntas planteadas en este documento, así como los archivos requeridos en ciertos incisos. Este laboratorio se basa en un tutorial en tres partes por Paulo Baltarejo Sousa y Luis Lino Ferreira (ver fuente al final).

Materiales: necesitará Ubuntu 8.04 *a.k.a. Hardy Heron* (http://old-releases.ubuntu.com/releases/hardy/ubuntu-8.04.4-desktop-i386.iso); y el *kernel* 2.6.24 de Linux (se descargará durante el laboratorio). Descargue también el material del tutorial de Sousa y Ferreira, en https://sourceforge.net/projects/linuxedfschedul/files/?source=navbar.

Contenido:

Aunque ya ha visto bastante código en C y probablemente tenga experiencia previa con el lenguaje, es importante conocer algunas de sus características para que el código que se provea en este laboratorio no sea copiado ciegamente, sino entendido en el proceso. Por ello, investigue y resuma:

- Funcionamiento y sintaxis de uso de `structs`.

  Las matrices permiten definir tipos de variables que pueden contener varios elementos de datos del mismo tipo. La estructura similar es otro tipo de datos definido por el usuario disponible en C que permite combinar elementos de datos de diferentes tipos. Las estructuras se utilizan para representar un registro. Para definir una estructura, debe utilizar la instrucción struct. La instrucción struct define un nuevo tipo de datos, con más de un miembro.

- Propósito y directivas del preprocesador.

  El preprocesador C, a menudo conocido como cpp, es un procesador de macros que el compilador C utiliza automáticamente para transformar el programa antes de la compilación. Se llama procesador de macros porque le permite definir macros, que son abreviaturas breves para construcciones más largas. El preprocesador C está diseñado para usarse solo con código fuente de C, C++ y Objective-C

- Diferencia entre `*` y `&` en el manejo de referencias a memoria (punteros).

  Un puntero en C++ es una variable que contiene la dirección de memoria de otra variable. Una referencia es un alias para una variable ya existente. Una vez que una referencia se

inicializa en una variable, no se puede cambiar para hacer referencia a otra variable. Por lo tanto, una referencia es similar a un puntero const.

- Puntero

```
int a = 5;
// some code
int *p = &a;
```

- Referencia

```
int a = 5;
int &ref = a;
```

- Propósito y modo de uso de APT y `dpkg`.

Dpkg es el administrador de paquetes Linux Debian. Cuando se utilizan apt o apt-get invocan el programa dpkg para instalar o quitar aplicaciones, mientras que incluye funciones adicionales dpkg no le gusta la resolución de dependencias. El programa dpkg se puede utilizar para instalar o eliminar programas, enumerarlos o información específica sobre ellos. El apt del comando es ventajoso sobre dpkg porque resuelve las dependencias y descarga el software actualizado automáticamente. Para descargar software este comando apunta a una serie de repositorios de software ubicados en el archivo /etc/apt/sources.list. Inicialmente después de instalar Debian necesitamos editar este archivo comentando la línea que apunta a la ruta de instalación de DVD/USB de Debian y añadiendo los repositorios adecuados. El comando apt utiliza el programa dpkg para administrar paquetes.

- ¿Cuál es el propósito de los archivos `sched.h` modificados?

  Definir estructuras, constantes y prototipos de funciones para implementarlas en distintas políticas de calendarización.

- ¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?

  Estas macros definen constantes para identificar las políticas de calendarización implementadas por el sistema.

- ¿Qué es una *task* en Linux?

  Es un proceso.

- ¿Cuál es el propósito de `task_struct` y cuál es su análogo en Windows?

  Task_struct es la implementación de un PCB en Linux, por lo que su análogo en Windows sería un KPROCESS en conjunto con el PEB y EPROCESS

- ¿Qué información contiene `sched_param`?

  Su propósito es almacenar los parámetros que permitirán la implementación de las políticas de calendarización. En este caso posee únicamente la prioridad de calendarización y, luego de los cambios, un identificador y una cantidad de nanosegundos al cabo de los cuales debe concluir la task, luego de iniciar su ejecución.

- ¿Para qué sirve la función `rt_policy` y para qué sirve la llamada `unlikely` en ella?

  Funciona para determinar si un cambio en la calendarización se hará hacia la clase de tiempo real. Unlikely favorece la velocidad de ejecución de las instrucciones que corresponden al incumplimiento de la condición especificada. En este caso, ayuda a la velocidad de respuesta cuando el cambio de calendarización no involucra las pólizas SCHED_FIFO, SCHED_RR y SCHED_CASIO_POLICY, lo que sugiere que se espera que los cambios se hagan a clases de prioridad normal.

- ¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?

  El nombre de la póliza y los cambios realizados indican que dichas tareas presentan un deadline, característico de tareas que se ejecutan en tiempo real

- Describa la precedencia de prioridades para las políticas EDF, RT y CFS, de acuerdo con los cambios realizados hasta ahora.

  En orden de prioridad descendente, EDF, luego RT y luego CFS.

- Explique el contenido de la estructura `casio_task`.

  Esta estructura define una tarea calendarizable por la SCHED_CASIO_POLICY existente en el sistema. Se representa con un nodo en el árbol red-black para calendarizar la ejecución de casio_tasks.

- Explique el propósito y contenido de la estructura `casio_rq`.

  Por procesador se mantiene una cola de procesos compitiendo por ejecución. La estructura casio_rq mantiene la cola de casio_tasks listas para ejecución, por procesador.

- ¿Qué es y para qué sirve el tipo `atomic_t`? Describa brevemente los conceptos de operaciones RMW (*read-modify-write*) y *mappeo* de dispositivos en memoria (MMIO).

- Es una variable en la que las operaciones siempre son atómicas.
- Rmw → operadores atómicos que evitan las race condition.
- Mmio → redireccionamiento de memoria que simula registros de la memoria RAM en CPU.

¿Qué indica el campo .next de esta estructura?
Indica la siguiente clase de calendarización en la jerarquía de prioridades.

• Tomando en cuenta las funciones para manejo de lista y red-black tree de casio_tasks, explique el ciclo de vida de una casio_task desde el momento en el que se le asigna esta clase de calendarización mediante sched_setscheduler. El objetivo es que indique el orden y los escenarios en los que se ejecutan estas funciones, así como las estructuras de datos por las que pasa. ¿Por qué se guardan las casio_tasks en un red-black tree y en una lista encadenada?

Se guardan de esta manera, debido a que una lista encadenada ayuda a almacenar las casio_tasks que existen en el sistema, pero el red black tree organiza únicamente aquellas que están listas para ejecución.

• ¿Cuándo preemptea una casio_task a la task actualmente en ejecución?

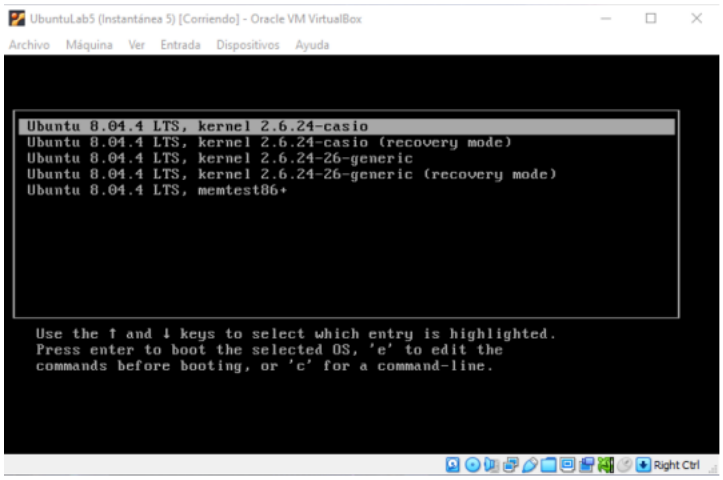
Cuando la tarea en ejecución no es una casio_task o cuando la deadline absoluta de la casio_task en el extremo izquierdo del árbol red-black es menor a la de la task en ejecución

• ¿Qué información contiene el archivo system que se especifica como argumento en la ejecución de casio_system?

Este archivo contiene información de configuración para las tareas con las que se prueba el nuevo calendarizador. Estos argumentos incluyen el número de proceso asignado por la simulación y parámetros en segundos para la deadline de cada tarea

• Investigue el concepto de aislamiento temporal en relación a procesos. Explique cómo el calendarizador SCHED_DEADLINE, introducido en la versión 3.14 del kernel de Linux, añade al algoritmo EDF para lograr aislamiento temporal.

El aislamiento temporal se refiere a la capacidad ofrecida por un sistema operativo para garantizar que el comportamiento temporal es independiente de otros procesos concurrentes.



Fuente:

- http://www.embedded.com/design/operating-systems/4204929/Real-Time-Linux-Scheduling-Part-1
- https://www.embedded.com/design/operating-systems/4204971/Real-Time-Linux-Scheduling-Part-2
- https://www.embedded.com/design/operating-systems/4204980/Real-Time-Linux-Scheduling-Part-3