Universidad del Valle de Guatemala
Sistemas Operativos Laboratorio 5

Investigue y resuma:

- Funcionamiento y sintaxis de uso de structs.

Un struct es otro tipo de dato definido por el usuario disponible en C que permite combinar elementos de datos de diferentes tipos. Los strutcs se utilizan para representar un registro. Para definir un struct, la sintaxis es la siguiente:

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

Al final de la definición de la estructura se puede especificar una o más variables de estructura, pero es opcional. Para acceder a cualquier miembro de una estructura se utiliza “.”. Se codifica como un punto (.) entre el nombre de la variable de estructura y el miembro de la estructura al que queremos acceder.

- Propósito y directivas del preprocesador.

El preprocesador de C (cpp) es el primer programa invocado por el compilador y procesa directivas. Estas directivas no son específicas de C y pueden ser usadas con cualquier tipo de archivo. El preprocesador utiliza 4 etapas denominadas Fases de traducción: tokenizado léxico, empalmado de líneas y manejo de directivas.

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

La difernecia es que “*” es utilizado para referirse a un puntero hacia el espacio en memoria, mientras que “&” es utilizado para referirse a la dirección en memoria (es la referencia como tal).

- Propósito y modo de uso de APT y dpkg.

Advanced Packaging Tool (APT) es un gestor de paquetes diseñado originalmente para Debian como front-end para la utilidad dpkg. Se utiliza para instalar o actualizar todas las aplicaciones dependientes necesarias para que se puedan instalar paquetes .deb. Incluye a apt-get para realizar instalaciones y apt-cache para consultar información de la base de datos de paquetes Debian.

dpkg es una herramienta de sistema de bajo nivel para extraer, analizar, descomprimir e instalar o eliminar archivos .deb. Sin embargo, lo que sucede realmente (a la hora de instalar paquetes) es que apt-get obtiene los archivos necesarios, que luego los pasa a dpkg para extraerlos, analizarlos e instalarlos en las ubicaciones correctas y configurarlos de acuerdo con los scripts dentro de ellos.

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

<sched.h> define la estructura sched_param, que contiene los parámetros de programación requeridos para la implementación de cada política de programación soportada. Este struct contiene la variable sched_priority que se utiliza para saber la prioridad de calendarización. Hay 3 políticas de calendarización ya definidas, y esas son FIFO (SCHED_FIFO), Round robin (SCHED_RR) y otras (SCHED_OTHER). Los modificados tienen como fin alterar la calendarización de eventos.

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

SCHED_BATCH se puede usar con una prioridad estática o dinámica. Esta política es similar a SCHED_OTHER pero la diferencia es que esta política hará que se asuma que el subproceso es intensivo en CPU.

- ¿Qué es una *task* en Linux?

El término task se utiliza en el kernel de Linux para referirse a una unidad de ejecución, que puede compartir varios recursos del sistema con otras tareas en el sistema.

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

Este struct se utiliza para almacenar toda la información con respecto a una tarea (task) que el kernel puede necesitar consultar. Actua como un descriptor de proceso.

El análogo en Windows es thread_info.

- ¿Qué información contiene sched_param?

sched_param contiene información sobre la prioridad del proceso (sched_priority), la prioridad del procesos que se está corriendo actualmente (sched_curpriority) y otras variables como sched_ss_low_priority, sched_ss_max_repl, sched_ss_repl_period y sched_ss_init_budget.

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

rt_policy() es una función usada por el sched_set_scheduler() y nice() y se utiliza para decidir si una política de programación dada pertenece a la clase en tiempo real (SCHED_RR y SCHED_FIFO) o no.

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

La política EDF (Earliest deadline first) es un algoritmo de calendarización de prioridades dinámico utilizado en tiempo real para colocar procesos en una cola de prioridad. Cada vez que se produce un evento con algun task se busca en la cola el proceso más cercano a su fecha límite y este son el tipo de tareas que calanderiza.

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

EDF → RT → CFS → IDLE.

- Explique el contenido de la estructura casio_task.

Contiene el nodo del task como tal, un deadline absoluto, el nodo en la cabeza de la lista de tasks y un puntero a el task_struct del task.

- Explique el propósito y contenido de la estructura casio_rq.

Permite que el Sistema pueda referirse a las tareas calanderizadas con la política nueva, contiene la tarea raiz, la cabeza de la lista de casio tasks y un identificador atomic_t sobre su estado.

- ¿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).

Se utiliza el tipo atomic_t como un Contador entero no volatil. De forma breve, RMW (read-modify-write) es una clase de operaciones atómicas que leen una ubicación de memoria y escriben un nuevo valor en ella simultáneamente. Puede ser con un valor nuevo o alguna función del valor anterior. MMIO es un método para hacer intercambio de información del CPU con otras unidades del equipo. MMIO utiliza el mismo espacio de direcciones para direccionar tanto la memoria como los dispositivos de input/output.

- ¿Qué indica el campo .next de esta estructura?

El puntero ".next" direcciona a una singly-linked list donde se tienen los identificadoes para las siguiente(s) tasks que debe hacer el Real-time scheduler.

- ¿Por qué se guardan las casio_tasks en un *red-black tree* y en una lista encadenada?

Se utiliza un red-black tree ya que es una estructura de datos en forma de abrol que permite el auto-balanceo de nodos, al realizarle modificaciones este se vuelve a balancear y pintar y no requiere de mucho espacio de memoria ya que solo utiliza 1 bit para guardar la coloración del arbol. Ahora bien, utilizar una linked list también es buena idea ya que es una estructura de datos dinámica, no realiza desperdicio de memoria y es muy eficiente a la hora de recorrerla.

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

Primero se encarga de que sea el identificador de una SCHED_CASIO, luego se busca que haya un task con con deadline cercano y se verifica que exista una lista de tasks. De no ser esto posible, es decir, no lograr obtener un *t* y *curr*, se preemtea.

- Adjunte ambos archivos de resultados de casio_system a su entrega, comentando sobre sus diferencias.

- 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.

Es utilizado para correr programas de forma segura, aislados del sistema real, lo cual permite hacer pruebas sin afectar al sistema. El kernel de un sistema operativo suele tener “celdas” y estas pueden llegar a tener sus propios servicios y permisos en cuanto el sistema. Se pueden utilizar como servidores virtualizados dentro del núcleo. El SCHED_DEADLINE, es una implementación de EDF dentro de la clase sched_dl, que con la adición de un mecanismo de CBS hace posible el aislamiento temporal de procesos.