**Laboratorio 5**

* Funcionamiento y sintaxis de uso de structs.

Es un tipo de dato definido por el usuario que permite combinar data de diferentes tipos. Los structs o estructuras son utilizadas frecuente mente para representar un registro, funcionan de manera muy similar a una class de java.

Su sintaxis es la siguiente:

struct [structure tag] {

member definition;

member definition;

...

member definition;

} [one or more structure variables];

* Propósito y directivas del preprocesador.

Las directivas del preprocesador se usan frecuentemente para que los programas de origen sean más fáciles de cambiar y compilar en diferentes entornos de ejecución. Las directivas del archivo de código fuente le indican al preprocesador que realice acciones específicas.

Dentro de las directivas que reconoce el preprocesador tenemos:

* + #define
  + #elif
  + #else
  + #endif
  + #error
  + #if
  + #ifdef
  + #ifndef
  + #import
  + #include
  + #line
  + #pragma
  + #undef
  + #using
* Diferencia entre \* y & en el manejo de referencias a memoria (punteros).

Se difieren ya que el operador & devuelve la dirección de memoria de una variable, incluso la de un puntero. Por otro lado, un puntero simplemente es una variable que guarda la dirección de memoria de otra variable, su objetivo principal es leer y escribir en la memoria.

* Propósito y modo de uso de APT y dpkg.

El comando dpkg instalara un paquete en específico, y notficara de cualquier dependencia adicional que se tenga que instalar, pero no los instalara.

APT es un sistema de gestión de paquetes, esta maneja las instalaciones de paquetes en distribuciones Linux basadas en Debian. Un sistema de gestión de paquetes instala, elimina y cambia paquetes fácilmente.

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

El propósito principal de estos archivos es definir estructuras, constantes y otros elementos que nos ayudaran a implementar nuestra política de calendarización.

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

Su propósito es definir constantes para identificar la política de calendarización a ser implementada por el sistema.

* ¿Qué es una task en Linux?

Task es otra forma de llamar a un proceso en Linux.

* ¿Cuál es el propósito de task\_struct y cuál es su análogo en Windows?

Task\_struct es una estructura de datos del kernel de Linux, es cargada en la RAM y contiene información del proceso. Es llamada la implementación de un PCB (Process Control Block) de Linux, por lo que su análogo en Windows sería el KPROCESS.

* ¿Qué información contiene sched\_param?

Este almacena los parámetros que permitirán la implementación de las políticas de calendarización. Para esta implementación se guarda un identificador, un cuantificador de tiempo que indica en cuantos nanosegundos concluirá la task y la prioridad de calendarización.

* ¿Para qué sirve la función rt\_policy y para qué sirve la llamada unlikely en ella?

La función rt\_policy nos ayuda a determinar si un cambio en la calendarización se hará hacia la clase de tiempo real. La llamada unlikely como su nombre lo indica nos ayuda a saber si algo no es probable, así mismo esta función favorece la velocidad de ejecución para aquellas instrucciones que si favorecen el incumplimiento de la condición.

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

Para aquellas tareas que presentan un deadline, esto lo sabemos por el nombre de la póliza y los cambios realizados, dichas características representan a 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.

El orden de prioridad será, EDF-RT-CFS, donde EDF es el de prioridad más alta y CFS el de más baja.

* Explique el contenido de la estructura casio\_task:

La estructura casio\_task define una tarea calendarizable por la SCHED\_CASIO\_POLICY. Esta tarea es representada por un nodo en el red-black tree, este es usado para calendarizar su la ejecución de casio\_task. Tambien es representada en la lista doblemente encadenada, esta registra todas las casio\_tasks del sistema. Por último, presenta una referencia a un proceso y un campo numérico, que representa el deadline absoluto que debe cumplir la ejecución del task.

* Explique el propósito y contenido de la estructura casio\_rq.

La estructura casio\_rq mantiene la cola de casio\_tasks listas para ejecución, esto se hace por procesador. Esta estructura es importante ya que como es bien sabido un procesador mantiene una cola de procesos compitiendo por ejecución, casio\_rq es una manera de asegurarse que casio\_tasks esten en la competición de ejecución.

* ¿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).

Atomic\_t es una simple variable de tipo entera que cuenta con un conjunto de operación que están garantizadas para ser atómicas sin la necesidad de un bloqueo explicito.

RMW es una calse de operadores atómicos que leen locación de memoria y pueden escribir nuevos valores a ella simultáneamente. Estas operaciones evitan race-conditions en aplicaciones de multi-threading.

MMIO es una técnica de direccionamiento de memoria, esta técnica perite construir el sistema de tal forma que la CPU considera los registros de ese chip como si fueran posiciones de memoria RAM.

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

Este campo 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 comienza asignando la clase de calendarización mediante sched\_setscheduler, al ejecutarse esto se agrega la task a la lista encadenada mediante la función add\_casio\_task\_2\_list, esto representa la creación de una nueva casio\_task en el sistema.

Al un momento en que una casio\_task cambia su estado a ready se agrega al red-black-tree mediante la función enqueue\_task\_casio, durante el proceso se configura su deadline absoluta. Cabe destacar que durante este proceso se utilizan las funciones, find\_casio\_task\_list y insert\_casio\_task\_rb\_tree.

Al una casio\_task salir del estado ready se debe sacar del RBT, esto se hace con la función equeue\_task\_casio, esta llama a find\_casio\_task\_list y a remove\_casio\_task\_rb\_tree. Una casio\_task solo puede salir del estado ready si termina su ejecución o se hace una llamada bloqueadora.

En caso de que una casio\_task termine su ejecución también debe ser sacada de la lista encadenada, esto se hace con la función rem\_casio\_task\_list.

Se utiliza tanto una lista encadenada como un RBT ya que, la lista encadenada es la encargada de almacenar las casio\_tasks existentes en el sistema, mientras que en el RBT solo están aquellas que están listas para ejecución.

* ¿Cuándo preemptea una casio\_task a la task actualmente en ejecución?

Debido a que las casio\_task son las que tienen máxima prioridad, esto solo puede pasar cuando la tarea en ejecución no es una casio\_task. También puede suceder cuando el deadline absoluto del casio\_task que esta en el extremo izquierdo del RBT es menor al de la tarea en ejecución.

* Ejecute nuevamente el archivo casio\_system tal como se hizo al inicio del laboratorio, pero guardando los resultados en un archivo diferente. Adjunte ambos archivos de resultados de casio\_system a su entrega, comentando sobre sus diferencias.

Archivos entregados en el git del laboratorio.

Diferencias:

Lo primero que podemos observar es que el pre\_casio.txt contiene solamente un task, por ser un único task tuvo la prioridad mas alta. En comparación podemos ver que post\_casio.txt tuvo mas tasks, en donde todos ellos tuvieron una prioridad alta al ser del tipo casio\_task. Otro dato a tener en cuenta es que los deadlines de post\_casio.txt se mantienen en un ranfo de 14000000000 a 16000000000 nanosegundos. También podemos ver por los tiempos de ejecución que en post\_casio.txt que en conjunto tomaron mas tiempo para la ejecución, sin embargo, esto se debe a que esta calendarización hizo más tasks que en comparación con pre\_casio.txt

Dentro de las similitudes encontradas en los .txt podemos encontrar que cuando se terminan los tasks ambos mandan una señal a SIGUSR1 para indicar que se inician todos los tasks, seguidamente de esto se manda una señal a SIGUSR2 para finalizar todos los tasks. De ultimo muestran un mensaje indicanco que todos los tasks fueron completados correctamente.

* Ubique el archivo de log de eventos registrados por la calendarización implementada. Adjunte este archivo con su entrega.

Archivo entregado en el git del laboratorio.

* Agregue comentarios explicativos a los archivos casio\_task.c y casio\_system.c que permitan entender el propósito y funcionamiento de este código. Asegúrese de aclarar el uso de instrucciones y estructuras que no conozca (como, por ejemplo, los timers y la estructura itimerval). ¿Qué información contiene el archivo system que se especifica como argumento en la ejecución de casio\_system?

Archivos con comentarios entregados en el git del laboratorio.

Respuesta a la pregunta:

El archivo system contiene información de configuración destinada para las tareas con las que se probara el nuevo calendarizador. Entre los argumentos que se incluyen están desde el numero de proceso asignado por la simulación, retrasos mínimos y máximos, parámetros en segundos para la deadline de cada tarea, tiempos mínimos y máximos de ejecución de cada tarea, intervalos de tiempo para disparar tareas subsecuentes, y muchos más.

* 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.

Asilamiento temporal es la capacidad ofrecida por un SO para garantizar que un proceso es independiente de otros procesos en relación a su tiempo de ejecución, esto quiere decir que el comportamiento temporal de un proceso no se ve afectado por otros.

SCHED\_DEADLINE funciona de tal manera que cada proceso tiene un deadline, un periodo de realización y un requisito. Este requisito es una cantidad de tiempo que el proceso necesitara al terminar cada periodo. El SO hace pruebas para garantizar que el task tenga todo lo que necesita y cando lo necesita. También se busca que el tiempo en que una tarea se suspende hasta que se reactiva sea relativamente corto, si este tiempo es corto se mantiene la misma deadline y si no se amplía la deadline.

Screen demostrando que la recopilación del kernel fue exitosa:

