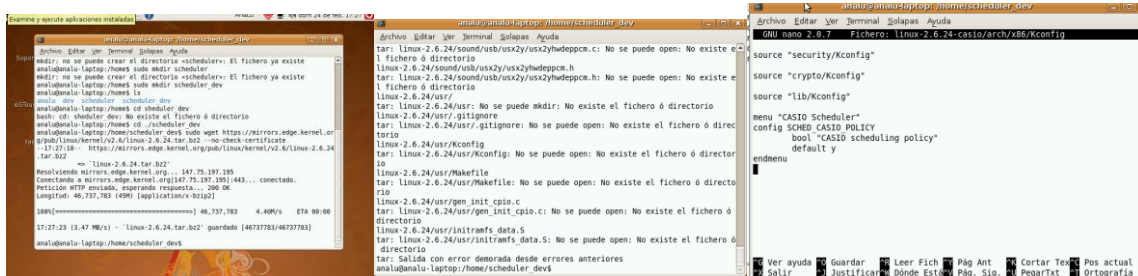


Laboratorio 4



- **Funcionamiento y sintaxis de uso de structs.**

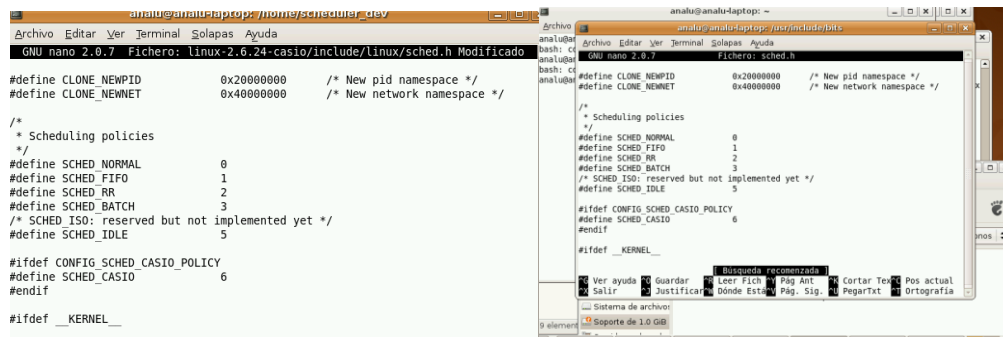
es una estructura en el lenguaje de programación C. Es una declaración de tipo de datos compuesta que define una lista agrupada físicamente de variables para colocarlas bajo un nombre en un bloque de memoria, lo que permite acceder a las diferentes variables a través de un solo puntero, o el nombre declarado de la estructura que devuelve la misma dirección. El tipo de datos de estructura puede contener muchos otros tipos de datos complejos y simples en una asociación, por lo que es un tipo natural para organizar registros de tipo de datos mixtos, como listas de entradas de directorio del disco duro.

```
struct [ <nombre tipo de estructura > ] {  
    [<tipo> <nombre-variable[, nombre-variable,...]>]; [<tipo> <nombre-variable[, nombre-variable,...]>]; ...  
}  
[ <variables de estructura> ] ;
```
- **Propósito y directivas del preprocesador.**

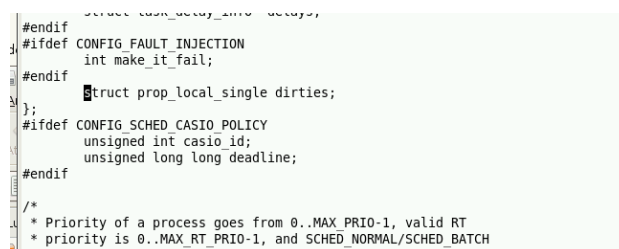
Las directivas de pre procesamiento se encargan de mirar la información del documento antes de compilarlo completamente verificando que no hayan errores, estas directivas es lo primero que se dispone en el archivo a compilar ya que de ahí es donde se llenaran las diferentes bibliotecas a utilizar. Entre estas se encuentra el if, else , elif, endif...El preprocesador es el preprocesador para el lenguaje de programación C. Es el primer programa invocado por el compilador y procesa las directivas antes mencionadas.
- **Diferencia entre * y & en el manejo de referencias a memoria (punteros).**

Para definir a los tipos punteros sino que según sea el tipo de dirección que va a almacenar ese será el tipo de operador *. Uno de los casos para utilizar es un puntero que almacene una dirección de memoria en un entero se utiliza int* px;. Por otro lado, utilizamos & como para imprimir el valor de la dirección de memoria. Sin embargo, como posición de memoria que ocupa también su dirección de memoria que se puede obtener con el operador & es lo que se llama un puntero de puntero.
- **Propósito y modo de uso de APT y dpkg.**

Dpkg instala un paquete entonces cuando hace `dpkg -i packageName.deb` instala solo del paquete y notificara de las dependencias mientras este se instale. Pero no lo instalara porque las dependencias no se encuentran allí actualmente. Por otro lado, apt-get son herramientas que se preparan para instalar, remover y cambiar de paquetes con facilidad.



- **¿Cuál es el propósito de los archivos sched.h modificados?**
Este inicializa el casio_rq fields y nos servirá para construir un nuevo Scheduling policy module.
- **¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?**
Por lo tanto, SHED_CASIO es la plataforma que se utiliza para implementar el algoritmo de programación EDF mediante la modificación de la versión del kernel. . Cada procesador contiene una cola de ejecución de todos los procesos ejecutables asignados a él. El POLICY utiliza esta cola de ejecución para seleccionar el mejor proceso a ejecutar. La información para estos procesos se almacena en una estructura de datos por procesador llamada struct rq.



- **¿Qué es una task en Linux?**
Durante la administración de un sistema, poder programar un task para una ejecución posterior es una capacidad crucial : ya que permite realizar una copia de seguridad de una base de datos o ejecutar un script de mantenimiento.
- **¿Cuál es el propósito de task_struct y cuál es su análogo en Windows?**
Es un elemento en la lista de tareas. Contiene toda la información sobre un proceso específico. El task_struct es una estructura de datos relativamente grande , el descriptor del proceso contiene los datos que describen el programa en ejecución: archivos abiertos, espacio de direcciones del proceso. Windows puede utilizar un task scheduler schema.

```

{
    sg->_cpu_power += val;
    sg->reciprocal_cpu_power = reciprocal_value(sg->_cpu_power);
}
#endif

static inline int rt_policy(int policy)
{
    if (unlikely(policy == SCHED_FIFO) || unlikely(policy == SCHED_RR))
    #ifdef CONFIG_SCHED_CASIO_POLICY
        || unlikely(policy == SCHED_CASIO)
    #endif
    {
        return 1;
    }
    return 0;
}

```

- **¿Para qué sirve la función `rt_policy` y para qué sirve la llamada `unlikely` en ella?**
`Rt_policy` se utiliza para decidir si una política de programación dada pertenece a la clase en el tiempo real. El `rt_policy` tiene un `unlikely()` que la política que esta verificando es de prioridad RT (`SCHED_FIFO` o `SCHED_RR`).
- **¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?**
 Trabaja con un set de tareas periódicas independientes.

```

#include "sched_stats.h"
#include "sched_idletask.c"
#include "sched_fair.c"
#include "sched_rt.c"
#ifdef CONFIG_SCHED_DEBUG
#include "sched_debug.c"
#endif
#ifdef CONFIG_SCHED_CASIO_POLICY
#include "sched_casio.c"
#endif

#ifdef CONFIG_SCHED_CASIO_POLICY
#define sched_class_highest (&casio_sched_class)
#else
#define sched_class_highest (&rt_sched_class)
#endif
/*
 * Update delta_exec, delta_fair fields for rq.
 */

```

- **Describe la precedencia de prioridades para las políticas EDF, RT y CFS, de acuerdo con los cambios realizados hasta ahora.**
 El RT (tiempo real) y el CFS (programación completamente justa) están codificados en los codigos `kernel_source_code/kernel/sched_rt.c...` En `casio_sched_class` es el modulo del planificador de mayor prioridad, este campo apunta al siguiente modulo del planificador de baja prioridad que es `struct rt_sched_class` que implementa el modulo RT.
 En el EDF, cada vez que llega una nueva tarea, ordena la cola lista que la tarea más cercana al final de su periodo asignado tiene una prioridad más alta. Anula la tarea en ejecución si no se coloca en el primero de la cola en la última de la clasificación.

```

p->sched_class = &rt_sched_class;
break;
}

p->rt_priority = prio;
p->normal_prio = normal_prio(p);
/* we are holding p->pi_lock already */
p->prio = rt_mutex_getprio(p);
set_load_weight(p);

#ifdef CONFIG_SCHED_CASIO_POLICY
case SCHED_CASIO:
    p->sched_class = &casio_sched_class;
    break;
#endif

**
* sched_setscheduler - change the scheduling policy and/or RT priority of a task
* @p: the task in question.

```

```

/* Like positive nice levels, dont allow tasks to
 * move out of SCHED_IDLE either:
 */
if (p->policy == SCHED_IDLE && policy != SCHED_IDLE)
    return -EPERM;

/* can't change other user's priorities */
if ((current->euid != p->euid) &&
    (current->euid != p->uid))
    return -EPERM;

#ifdef CONFIG_SCHED_CASIO_POLICY
if (policy == SCHED_CASIO){
    p->deadline = param->deadline;
    p->casio_id = param->casio_id;
}
#endif

```

- **Explique el contenido de la estructura `casio_task`.**
 Lleva al punto a la tarea `struct casio` almacenada en la lista vinculada de la estructura `struct casio_rq` que apunta a la tarea `p`.

```

struct load_weight load;
unsigned long nr_load_updates;
u64 nr_switches;

struct cfs_rq cfs;

#ifdef CONFIG_FAIR_GROUP_SCHED
/* list of leaf cfs_rq on this cpu: */
struct list_head leaf_cfs_rq_list;
#endif

struct rt_rq rt;

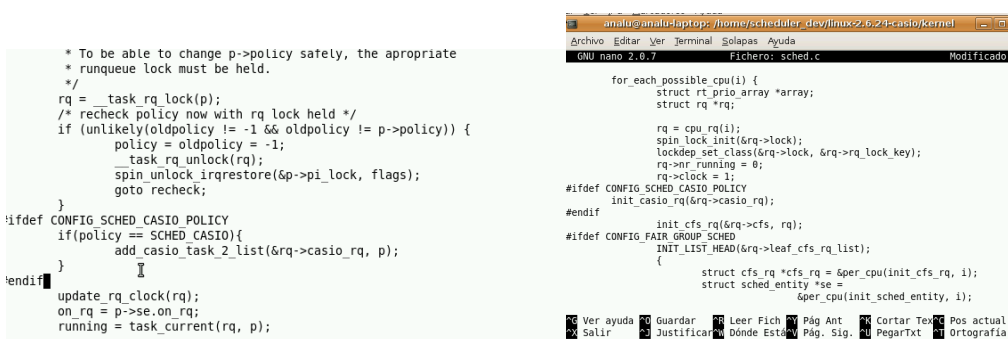
#ifdef CONFIG_SCHED_CASIO_POLICY
struct casio_rq casio_rq;
#endif

/*
 * This is part of a global counter where only the total sum
 * over all CPUs matters. A task can increase this counter on
 * one CPU and if it got migrated afterwards it may decrease
 * it on another CPU. Always updated under the runqueue lock:

```

Ver ayuda Guardar Leer Fich Pág Ant Cortar Text Pos actual

- **Explique el propósito de la estructura casio_rq.**
 Los otros campos son funciones que actúan como devoluciones de llamadas a eventos específicos. Las tareas de CASIO que ya se encuentran almacenadas en la lista enlazada, tiene como encabezado es el campo de lista de struct casio_rq.



```

* To be able to change p->policy safely, the appropriate
* runqueue lock must be held.
*/
rq = task_rq_lock(p);
/* recheck policy now with rq lock held */
if (unlikely(oldpolicy != -1 && oldpolicy != p->policy)) {
    policy = oldpolicy = -1;
    task_rq_unlock(rq);
    spin_unlock_irqrestore(&p->pi_lock, flags);
    goto recheck;
}

#ifdef CONFIG_SCHED_CASIO_POLICY
if (policy == SCHED_CASIO) {
    add_casio_task_2_list(&rq->casio_rq, p);
}
#endif

update_rq_clock(rq);
on_rq = p->se.on_rq;
running = task_current(rq, p);

```

```

for_each_possible_cpu(i) {
    struct rt_prio_array *array;
    struct rq *rq;

    rq = cpu_rq(i);
    spin_lock_init(&rq->lock);
    lockdep_set_class(&rq->lock, &rq->rq_lock_key);
    rq->nr_running = 0;
    rq->clock = 1;

#ifdef CONFIG_SCHED_CASIO_POLICY
    init_casio_rq(&rq->casio_rq);
#endif

    init_cfs_rq(&rq->cfs, rq);

#ifdef CONFIG_FAIR_GROUP_SCHED
    INIT_LIST_HEAD(&rq->leaf_cfs_rq_list);
    {
        struct cfs_rq *cfs_rq = &per_cpu(init_cfs_rq, i);
        struct sched_entity *se =
            &per_cpu(init_sched_entity, i);
    }
}

```

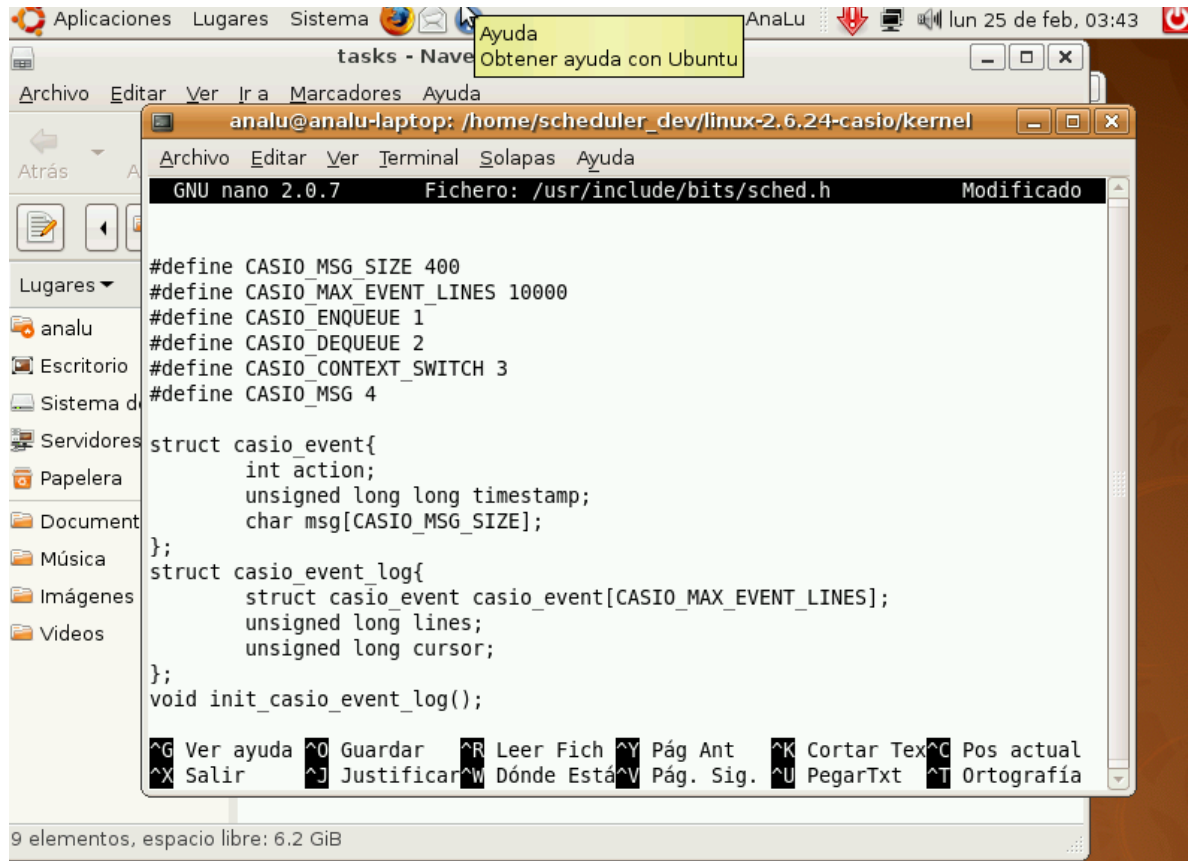
Ver ayuda Guardar Leer Fich Pág Ant Cortar Text Pos actual Salir Justificar Dónde Está Pág. Sig. Pegar Text Ortografía

- **¿Qué indica el campo .next de esta estructura?**
 Es un puntero a shed_class que se usa para organizar los módulos del planificador por prioridad en una lista vinculada y núcleo de planificador, comenzando por el módulo del planificador de mayor prioridad, busca una tarea ejecutable de cada módulo en un orden decreciente de prioridad. En este caso, dado que casio_sched_class es el módulo del planificador de mayor prioridad, este campo apunta al siguiente módulo del planificador de baja prioridad que es struct rt_sched_class que a su vez implementa un módulo RT.
- **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?**
 Al invocar la función de lista de tareas find_casio, lleva el puntero a la tarea struct casio almacenada en la lista vinculada de la estructura struct casio_rq que apunta a la tarea p. Luego, actualiza la fecha límite absoluta e inserta casio_task en el árbol rojo-negro. Cuando una tarea CASIO ya no se puede ejecutar, se llama la función dequeue_task_casio que deshace el trabajo de la función enqueue_task_casio. Si la tarea no se está ejecutando, entonces se elimina de la lista vinculada. La tarea CASIO con la primera fecha límite absoluta se elige para ser ejecutada en el procesador, esta función devuelve NULL y de esta manera el núcleo del planificador intenta encontrar una tarea en la siguiente clase de planificación de baja prioridad.

Sched_Setscheduler establece la política de programación del proceso especificado por el puntero p y los parámetros especificados por el parámetro de puntero , respectivamente.

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

Esta función, al invocar la función de lista de tareas find_casio, lleva el puntero a la tarea struct casio almacenada en la lista vinculada de la estructura struct casio_rq que apunta a la tarea p. Luego, actualiza la fecha límite absoluta e inserta casio_task en el árbol rojo-negro (insert_casio_task_rb_tree).



```
analalu@analalu-laptop: /home/scheduler_dev/linux-2.6.24-casio/kernel
GNU nano 2.0.7 Fichero: /usr/include/bits/sched.h Modificado

#define CASIO_MSG_SIZE 400
#define CASIO_MAX_EVENT_LINES 10000
#define CASIO_ENQUEUE 1
#define CASIO_DEQUEUE 2
#define CASIO_CONTEXT_SWITCH 3
#define CASIO_MSG 4

struct casio_event{
    int action;
    unsigned long long timestamp;
    char msg[CASIO_MSG_SIZE];
};

struct casio_event_log{
    struct casio_event casio_event[CASIO_MAX_EVENT_LINES];
    unsigned long lines;
    unsigned long cursor;
};

void init_casio_event_log();

^G Ver ayuda ^O Guardar ^R Leer Fich ^Y Pág Ant ^K Cortar Tex ^C Pos actual
^X Salir ^J Justificar ^W Dónde Está ^V Pág. Sig. ^U PegarTxt ^T Ortografía
```

- **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 un mecanismo para ejecutar programas con seguridad y de manera separada. A menudo se utiliza para ejecutar código nuevo o software de dudosa confiabilidad proveniente de terceros. Ese entorno aislado permite controlar de cerca los recursos proporcionados a los programas cliente a ejecutarse. admiten reservas de recursos: cada tarea programada en El presupuesto de Q se asocia con el presupuesto de Q (también conocido como tiempo de ejecución) y el período Q, que corresponde a una declaración al núcleo de que la unidad requiere Q unidades de tiempo.

Las tareas son entonces programado utilizando EDF [1] en estos plazos de programación (la tarea con el el primer plazo de programación se selecciona para su ejecución). Tenga en cuenta que el

la tarea realmente recibe las unidades de tiempo de "tiempo de ejecución" dentro de la "fecha límite" si un se utiliza la estrategia de "control de admisión" (ver Sección "4. Gestión de ancho de banda"). (claramente, si el sistema está sobrecargado, esta garantía no se puede respetar).