

Universidad del Valle de Guatemala  
Sistemas Operativos  
Douglas de León Molina  
Carné 18037

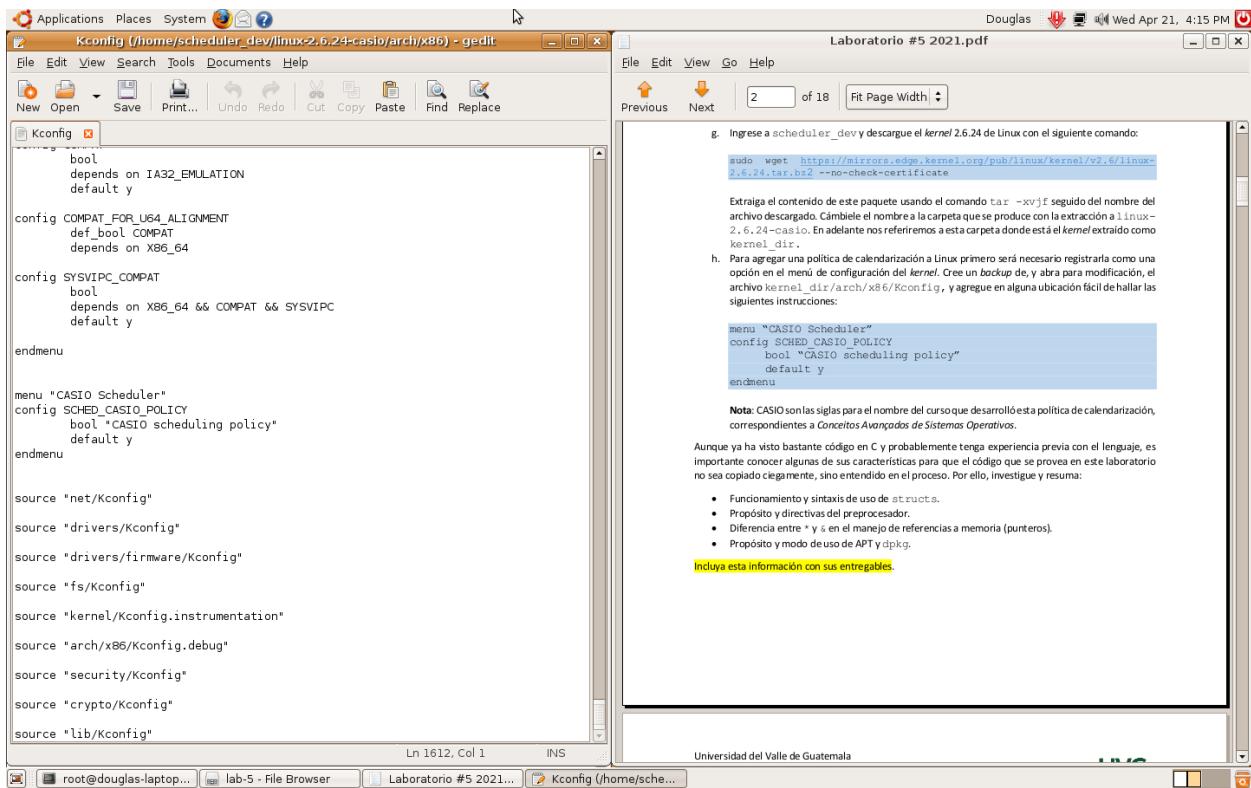
## Laboratorio #5

```
root@douglas-laptop:/home/douglas/Documents/lab-5/tasks# sudo ./casio_system system
> pre_casio.txt.
ERROR: Invalid argument
ERROR: Invalid argument
ERROR: Invalid argument
ERROR: Invalid argument
root@douglas-laptop:/home/douglas/Documents/lab-5/tasks# cat pre_casio.txt.

pid[4]
deadline[150000000000]
Before sched_setscheduler:priority 0
After sched_setscheduler:priority 0

Task(4) has just started
Job(4,1) starts
Job(4,1) ends (1.920000)
Job(4,2) starts
Job(4,2) ends (2.240000)
Job(4,3) starts
Job(4,3) ends (1.410000)

Task(4) has finished
I will send a SIGUSR1 signal to start all tasks
I will send a SIGUSR2 signal to finish all tasks
All tasks have finished properly!!!
root@douglas-laptop:/home/douglas/Documents/lab-5/tasks# █
```



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

Un struct o data structure, es un grupo de elementos de data juntos bajo un nombre. A esos elementos se les conoce como miembros y pueden ser diferentes tipos y tener diferentes largos. La sintaxis para un struct es la siguiente:

```
struct type_name {
    member_type1 member_name1;
    member_type2 member_name2;
    member_type3 member_name3;
    .
    .
}
```

- **Propósito y directivas del preprocesador**

El preprocesador realiza operaciones preliminares a los archivos antes de que pasen al compilador. El preprocesador se puede utilizar para compilar código condicionalmente, insertar archivos, especificar errores en tiempo de compilación y aplicar reglas específicas a secciones de código.

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

Los punteros (\*) son representaciones simbólicas a registros de memoria. Estos apuntan a los valores guardados en un registro. El símbolo & nos ayuda a obtener el registro de memoria asignado a una variable.

- Propósito y modo de uso de APT y dpkg.  
dpkg nos ayuda a instalar paquetes y APT es un manejador de paquetes. Ambos nos ayudan a instalar paquetes, pero APT es más completo y nos puede ayudar a instalar dependencias también.

The screenshot shows a Linux desktop environment with two windows open:

- Terminal Window (gedit):** Displays the content of the file `sched.h`. The code defines various clone flags and scheduling policies. It includes comments explaining the purpose of each define, such as clearing the TID in the child or setting the TID in the child. It also defines constants for different scheduling policies like SCHED\_NORMAL, SCHED\_FIFO, SCHED\_RR, SCHED\_BATCH, and SCHED\_IDLE.
- PDF Viewer:** Displays a document titled "Laboratorio #5 2021.pdf". The document is from the "Universidad del Valle de Guatemala" (UVG) and is dated Wednesday April 21, 4:21 PM. It contains several questions related to the `sched.h` file. One question asks to create a backup of the file and modify it. Another asks to modify the `/usr/include/bits/sched.h` file. The document also includes a note about the color coding of the code in the terminal window.

```

sched.h
-----
#define CLONE_CHILD_CLEARTID 0x00020000 /* clear the TID in the child */
#define CLONE_DETACHED 0x00040000 /* Unused, ignored */
#define CLONE_UNTRACED 0x00080000 /* set if the tracing process
can't force CLONE_PTRACE on this clone */
#define CLONE_CHILD_SETTID 0x01000000 /* set the TID in the child */
#define CLONE_STOPPED 0x02000000 /* Start in stopped state */
#define CLONE_NEWUTS 0x04000000 /* New utname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipc */
#define CLONE_NEWUSER 0x10000000 /* New user namespace */
#define CLONE_NEWPID 0x20000000 /* New pid namespace */
#define CLONE_NEWNET 0x40000000 /* New network namespace */

/*
 * Scheduling policies
 */
#define SCHED_NORMAL 0
#define SCHED_FIFO 1
#define SCHED_RR 2
#define SCHED_BATCH 3
/* SCHED_ISO: reserved but not implemented yet */
#define SCHED_IDLE 5

#ifndef CONFIG_SCHED_CASIO_POLICY
#define SCHED_CASIO 6
#endif
|
#ifndef __KERNEL__
struct sched_param {
    int sched_priority;
};

#include <asm/param.h> /* for HZ */

#include <linux/capability.h>
#include <linux/thread.h>
#include <linux/kernel.h>
#include <linux/types.h>
#include <linux/timex.h>
#include <linux/jiffies.h>

```

Ln 44, Col 1      INS

Note que lo que este extracto de código le indica con los colores es que agregue la parte de  
*#ifndef luego de #define SCHED\_IDLE 5.*

- i. En el archivo `/usr/include/bits/sched.h` realice la siguiente modificación:  

```

...
#define SCHED_BATCH 3
#endif

#define SCHED_CASIO 6

```
- ¿Cuál es el propósito de los archivos `sched.h` modificados?
- ¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?
- k. En `kernel_dir/include/linux/sched.h` busque la definición de la estructura `task_struct` (debería estar en la línea 921). Se agregarán a ella los parámetros con los que se relacionará una task general con una task calendarizada por nuestra nueva política. Para ello su modificación al archivo debe ser la siguiente:

The screenshot shows a Linux desktop environment with two windows. The left window is a terminal window titled 'sched.h (/usr/include/bits) - gedit' containing the source code for the 'sched.h' header file. The right window is a PDF viewer window titled 'Laboratorio #5 2021.pdf' showing the title page of a lab report. The title page includes the University of Valle de Guatemala logo, the course name 'Sistemas Operativos', and the instructors 'Docentes: Erick Pineda; Tomás Gálvez P.'. It also specifies 'Semestre I, 2021'. The terminal window displays the 'sched.h' code, which includes definitions for scheduling algorithms like SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR, and SCHED\_BATCH, along with various flags and constants. The PDF viewer window shows the beginning of the lab report, including instructions for modifying the kernel's scheduling policy.

```

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with the GNU C Library; if not, write to the Free
Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA
02111-1307 USA. */

#ifndef __need_schedparam
#define __need_schedparam
#endif

/* Scheduling algorithms. */
#define SCHED_OTHER    0
#define SCHED_FIFO     1
#define SCHED_RR      2
#ifndef __USE_GNU
#define SCHED_BATCH   3
#endif

#define SCHED_CASIO   5

#ifndef __USE_MISC
/* Cloning flags. */
#define CSIGNAL 0x000000ff /* Signal mask to be sent at exit. */
#define CLONE_VM 0x00000100 /* Set if VM shared between processes. */
#define CLONE_FS 0x00000200 /* Set if fs info shared between processes. */
#define CLONE_FILES 0x00000400 /* Set if open files shared between processes. */
#define CLONE_SIGHAND 0x00000800 /* Set if signal handlers shared. */
#define CLONE_PTRACE 0x00002000 /* Set if tracing continues on the child. */
#define CLONE_VFORK 0x00004000 /* Set if the parent wants the child to
wake it up on mm_release. */
#define CLONE_PARENT 0x00008000 /* Set if we want to have the same
parent as the cloner. */
#define CLONE_THREAD 0x00010000 /* Set to add to same thread group. */
#define CLONE_NEWNS 0x00020000 /* Set to create new namespace. */
#define CLONE_SYSVSEM 0x00040000 /* Set to shared SVID SEM_NDO semantics. */
#define CLONE_SETTLS 0x00080000 /* Set TLS info. */

```

Note que lo que este extracto de código le indica con los colores es que agregue la parte de `#ifndef` luego de `#define SCHED_IDLE` .

j. En el archivo `/usr/include/bits/sched.h` realice la siguiente modificación:

```

...
#define SCHED_BATCH 3
#endif

#define SCHED_CASIO 6

```

k. ¿Cuáles es el propósito de los archivos `sched.h` modificados?

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

m. En `kernel_dir/include/linux/sched.h` busque la definición de la estructura `task_struct` (debería estar en la linea 921). Se agregarán a ella los parámetros con los que se relacionará una task general con una task calendarizada por nuestra nueva política. Para ello su modificación al archivo debe ser la siguiente:

```

struct task_struct {
...
#endif

```

- ¿Cuál es el propósito de los archivos sched.h modificados?  
Los archivos sched.h modificados son nuestros calendarizadores y donde nosotros especificamos la nueva política de calendarización para el kernel.
- ¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?  
Las definición incluida es nuestra política de calendarización SCHED\_CASIO, mientras que las existentes son las que trae el kernel por defecto. Entre las incluidas están Round Robin y FIFO que vimos en clase.

The screenshot shows a Linux desktop environment with two windows open. On the left, a terminal window titled 'Applications Places System' displays the contents of the file 'sched.h' using the gedit text editor. The code in the file includes various #ifdef directives for different kernel configuration options like CONFIG\_COMPAT, CONFIG\_TASK\_DELAY\_ACCT, and CONFIG\_FAULT\_INJECTION. It defines structures such as compat\_robust\_list\_head, list\_head, and task\_struct, which contains fields like pi\_state\_list, rCU\_head, and fs\_excl. On the right, a PDF viewer window titled 'Laboratorio #5 2021.pdf' is open, showing a question from a lab assignment. The question asks about the purpose of the 'task\_struct' definition in 'sched.h' and compares it to its Windows counterpart. It also asks about the 'sched\_param' structure and its relation to 'rt\_policy'. The PDF has a watermark for 'UNIVERSIDAD DEL VALLE DE GUATEMALA' (UVG) and is dated Wednesday April 21, 8:43 PM.

- ¿Qué es una task en Linux?  
Un task en Linux es término utilizado para referirse a una unidad de ejecución.
- ¿Cuál es el propósito de task\_struct y cuál es su análogo en Windows?  
task\_struct es una estructura de datos que contiene toda la información acerca de un proceso, se podría decir que es el PCB. El análogo en Windows es el EPROCESS structure.

The terminal window shows the content of the file `sched.h` from the directory `/usr/include/bits`. The code defines various functions like `clone`, `unshare`, and `sched_getcpu`, and a structure `_sched_param` which includes fields for priority, deadline, and a CPU mask.

```

/* Clone current process. */
extern int clone (int (*__fn) (void *__arg), void *__child_stack,
                 int __flags, void *__arg, ...) __THROW;

/* Unshare the specified resources. */
extern int unshare (int __flags) __THROW;

/* Get index of currently used CPU. */
extern int sched_getcpu (void) __THROW;
#endif

__END_DECLS

#endif /* need schedparam */

#if !defined __defined_schedparam \
    && (defined __need_schedparam || defined __SCHED_H)
#define __defined_schedparam 1
/* Data structure to describe a process' schedulability. */
struct __sched_param
{
    int __sched_priority;
    unsigned int casio_id;
    unsigned long long deadline;
};

#undef __need_schedparam
#endif

#if defined __SCHED_H && !defined __cpu_set_t_defined
#define __cpu_set_t_defined
/* Size definition for CPU sets. */
#define __CPU_SETSIZE 1024
#define __NCPUBITS (8 * sizeof (__cpu_mask))

/* Type for array elements in 'cpu_set_t'. */
typedef unsigned long int __cpu_mask;

/* Basic access functions. */
#define __CPUELT(cpu) ((cpu) / __NCPUBITS)

```

The PDF viewer window displays a laboratory assignment for week 5. It contains several questions and code snippets related to the kernel's scheduling mechanism.

**Questions:**

- ¿Cuál es el propósito de los archivos `sched.h` modificados?
- ¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?
- k. En kernel dir/include/linux/sched.h busca la definición de la estructura `task_struct` (debería estar en la línea 921). Se agregarán a ella los parámetros con los que se relacionará una `task` general con una `task` calendarizada por nuestra nueva política. Para ello su modificación al archivo debe ser la siguiente:

```

struct task_struct {
    ...
    struct prop_local_single dirties;
#ifndef CONFIG_SCHED_CASIO_POLICY
    unsigned int casio_id;
    unsigned long long deadline;
#endif
};

```

**Answers:**

- ¿Qué es una task en Linux?
- ¿Cuál es el propósito de `task_struct` y cuál es su análogo en Windows?
- l. En este mismo archivo busca también la estructura `ached_param` (línea 47) y agréguele los mismos parámetros al final (siempre dentro de un bloque `#ifdef`). En /usr/include/bits/sched.h hay dos definiciones de `ached_param` [en realidad, una es para `_sched_param`; incluya estos cambios en ellas también, pero sin encerrarlos en un bloque `#ifdef`. Grabe y cierre `sched.h`.
- m. ¿Qué información contiene `sched_param`?
- n. Diríjase al archivo kernel\_dir/kernel/sched.c. La política de calendarización que emplearemos es la de `earliest deadline first` (EDF), por lo que debemos indicar al sistema operativo que nuestra política pertenece a esta clase. Busque la función `rt_policy` y modifiquela de la siguiente manera (sin olvidar crear una copia de `backup` del archivo):

```

static inline int rt_policy(int policy)
{
    if (unlikely(policy == SCHED_FIFO) || unlikely(policy == SCHED_RR))
#endif

```

- ¿Qué información contiene `sched_param`? `sched_param` contiene los parámetros de calendarización para un `task`, como la prioridad.

The screenshot shows a Linux desktop environment with two windows open. The left window is a terminal window titled "gedit" showing the file "sched.c" from the kernel source code. The right window is a PDF viewer titled "Wed Apr 21, 10:56 PM" showing a document titled "Laboratorio #5 2021.pdf".

```

*static inline u32 sg_div_cpu_power(const struct sched_group *sg, u32 load)
{
    return reciprocal_divide(load, sg->reciprocal_cpu_power);
}

/*
 * Each time a sched group cpu_power is changed,
 * we must compute its reciprocal value
 */
static inline void sg_inc_cpu_power(struct sched_group *sg, u32 val)
{
    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)
#ifndef CONFIG_SCHED_CASIO_POLICY
    || unlikely(policy == SCHED_CASIO))
#endif
    {
        return 1;
    }
    return 0;
}

static inline int task_has_rt_policy(struct task_struct *p)
{
    return rt_policy(p->policy);
}

/*
 * This is the priority-queue data structure of the RT scheduling class:
 */
struct rt_prio_array {
    DECLARE_BITMAP(bitmap, MAX_RT_PRIO+1); /* include 1 bit for delimiter */
    struct list_head queue[MAX_RT_PRIO];
};

```

The PDF document contains the following text:

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

UVG  
UNIVERSIDAD  
DEL VALLE  
DE GUATEMALA

L. En este mismo archivo busque también la estructura `sched_param` (línea 47) y agregue los mismos parámetros al final (siempre dentro de un bloque `#ifdef`). En `/usr/include/bits/sched.h` hay dos definiciones de `sched_param` (en realidad, una es para `_sched_param`). Incluya estos cambios en ellas también, pero sin encerrarlos en un bloque `#ifdef`. Grabe y cierre `sched.h`.

- ¿Qué información contiene `sched_param`?

m. Diríjase al archivo `kernel_dir/kernel/sched.c`. La política de calendarización que emplearemos es la de *earliest deadline first* (EDF), por lo que debemos indicar al sistema operativo que nuestra política pertenece a esta clase. Busque la función `rt_policy` y modifiquela de la siguiente manera (sin olvidar crear una copia de `backup` del archivo):

```

static inline int rt_policy(int policy)
{
    if (unlikely(policy == SCHED_FIFO) || unlikely(policy == SCHED_RR)
#ifndef 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?
- ¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?

n. Nuestra política será implementada en un archivo llamado `sched_casio.c`. Modifique `sched.c` de la siguiente manera:

```

...
#ifndef CONFIG_SCHED_CASIO_POLICY
#include "sched_debug.c"
#endif

#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
................................................................

```

- ¿Para qué sirve la función `rt_policy` y para qué sirve la llamada `unlikely` en ella?  
La función `rt_policy` define nuestra política de calendarización para real time. La llamada `unlikely` hace que el compilador optimice para evitar overhead en estas comparaciones.
- ¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?  
Calendariza las tareas que se clasifican como real time.

The screenshot shows a Linux desktop interface with two windows. The left window is a terminal window titled 'Applications Places System sched.c (/home/scheduler\_dev/linux-5.6.24-casio/kernel) - gedit'. It displays the source code for the 'sched.c' file, specifically focusing on the SCHED\_CASIO\_POLICY section. The right window is a PDF viewer titled 'Wed Apr 21, 11:04 PM Douglas Laboratorio #5 2021.pdf'. The PDF contains a question and instructions related to the scheduler policy.

```

scheduler.c (./home/scheduler_dev/linux-5.6.24-casio/kernel) - gedit
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
sched.c
iter_move_one_task(struct rq *this_rq, int this_cpu, struct rq *busiest,
                   struct sched_domain *sd, enum cpu_idle_type idle,
                   struct rq_iterator *iterator);
#endif

#ifndef CONFIG_CGROUP_CPUACCT
static void cpacct_charge(struct task_struct *tsk, u64 cputime);
#else
static inline void cpacct_charge(struct task_struct *tsk, u64 cputime) {}
#endif

#include "sched_stats.h"
#include "sched_idletask.c"
#include "sched_fair.c"
#include "sched_rt.c"
#ifndef CONFIG_SCHED_DEBUG
#include "sched_debug.c"
#endif

#ifndef CONFIG_SCHED_CASIO_POLICY
#include "sched_casio.c"
#endif
#ifndef 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.
 *
 * delta_fair clock advances at a rate inversely proportional to
 * total_load (rq->load.weight) on the runqueue, while
 * delta_exec advances at the same rate as wall-clock (provided
 * cpu is not idle).
 *
 * delta_exec / delta_fair is a measure of the (smoothed) load on this
 * runqueue over any given interval. This (smoothed) load is used
 * during load balance.
 */

```

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.

UVG  
UNIVERSIDAD  
DEL VALLE

que nuestra política pertenece a esta clase. Busque la función `rt_policy` y modifiquela de la siguiente manera (sin olvidar crear una copia de *backup* del archivo):

```

static inline int rt_policy(int policy)
{
    if (unlikely(policy == SCHED_FIFO) || unlikely(policy == SCHED_RR))
        /* 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?
• ¿Qué tipo de tareas calendula la política EDF, en vista del método modificado?
n. Nuestra política será implementada en un archivo llamado sched_casio.c. Modifique sched.c de la siguiente manera:
...
#include "sched_debug.c"
#endif

#ifndef CONFIG_SCHED_CASIO_POLICY
#include "sched_casio.c"
#endif

#ifndef CONFIG_SCHED_CASIO_POLICY
#define sched_class_highest (&casio_sched_class)
#else
#define sched_class_highest (&rt_sched_class)
#endif

/*
 * Update delta_exec, delta_fail fields for rq.
...

```

- Describa la precedencia de prioridades para las políticas EDF, RT y CFS, de acuerdo con los cambios realizados hasta ahora.  
Con nuestros cambios la política de mayor prioridad sería la de EDF, luego RT y por último CFS.

Laboratorio #5 2021.pdf

Wed Apr 21, 11:12 PM Douglas

**ched.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - gedit**

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

```
1 return cpu_rq(cpu)->idle;
```

```
2
```

```
3 /**
4 * find_process_by_pid - find a process with a matching PID value.
5 * @pid: the pid in question.
6 */
7 static struct task_struct *find_process_by_pid(pid_t pid)
8 {
9     return pid ? find_task_by_vpid(pid) : current;
10}
11
12 /* Actually do priority change: must hold rq lock. */
13 static void
14 _setscheduler(struct rq *rq, struct task_struct *p, int policy, int prio)
15 {
16     BUG_ON(p->se.on_rq);
17
18     p->policy = policy;
19     switch (p->policy) {
20     case SCHED_NORMAL:
21     case SCHED_BATCH:
22     case SCHED_IDLE:
23         p->sched_class = &fair_sched_class;
24         break;
25     case SCHED_FIFO:
26     case SCHED_RR:
27         p->sched_class = &rt_sched_class;
28         break;
29 #ifdef CONFIG_SCHED_CASIO_POLICY
30     case SCHED_CASIO:
31         p->sched_class = &casio_sched_class;
32         break;
33 #endif
34     }
35
36     p->rt_priority = prio;
37     p->normal_prio = normal_prio(p);
38     /* we are holding navi lock already */
39 }
```

Ln 4253, Col 31 INS

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... sched.c (/home/sche...

Semestre 1, 2021 DEL VALLE GUATEMALA

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

o Para que los procesos puedan calendarizarse con nuestra política deben cambiar su calendarizador con llamadas a sistema durante su ejecución. En la función \_\_setscheduler realice la siguiente modificación:

```
1     p->policy = policy;
2     switch(p->policy){
3
4 #ifdef CONFIG_SCHED_CASIO_POLICY
5     case SCHED_CASIO:
6         p->sched_class = &casio_sched_class;
7         break;
8 #endif
9 }
```

Y en la función sched\_setscheduler realice las siguientes modificaciones:

```
1 ...
2 if (policy < 0)
3     policy = oldpolicy = p->policy;
4 else if ((policy != SCHED_FIFO && policy != SCHED_RR &&
5          policy != SCHED_NORMAL && policy != SCHED_BATCH &&
6          policy != SCHED_IDLE)
7     /* */
8 #ifdef CONFIG_SCHED_CASIO_POLICY
9     && policy != SCHED_CASIO
10 #endif
11     )
12     return -EINVAL;
13 ...
14     /* can't change other user's priorities */
15     if ((current->euid != p->euid) &&
16         (current->euid != p->uid))
17         return -EPERM;
18     }
19
20 #ifdef CONFIG_SCHED_CASIO_POLICY
21     if (policy == SCHED_CASIO)
22         p->deadline = param->deadline;
23         p->casio_id = param->casio_id;
24     }
25 ...
26
27 #endif
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
315
316
317
317
318
319
319
320
321
321
322
323
323
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
13
```

The screenshot shows a Linux desktop environment with two windows open. The left window is a terminal window titled 'gedit' containing the source code for 'sched.c'. The code is written in C and defines structures for scheduling, including 'rt\_rq' and 'casio\_task'. The right window is a PDF viewer showing a document titled 'Laboratorio #5 2021.pdf'. The document contains instructions and comments about modifying the kernel's scheduling policy to use a Red-Black tree for task management.

```

/* leaf cfs_rq's are those that hold tasks (lowest schedulable entity in
 * a hierarchy). Non-leaf lrqs hold other higher schedulable entities
 * (like users, containers etc.)
 */
/* leaf_cfs_rq_list ties together list of leaf cfs_rq's in a cpu. This
 * list is used during load balance.
 */
struct list_head leaf_cfs_rq_list;
struct task_group *tg; /* group that "owns" this runqueue */

/* Real-Time classes' related field in a runqueue: */
struct rt_rq {
    struct rt_prio_array active;
    int rt_load_balance_idx;
    struct list_head *rt_load_balance_head, *rt_load_balance_curr;
};

#ifndef CONFIG_SCHED_CASIO_POLICY
struct casio_task{
    struct rb_node casio_rb_node;
    unsigned long long absolute_deadline;
    struct list_head casio_list_node;
    struct task_struct* task;
};

struct casio_rq{
    struct rb_root casio_rb_root;
    struct list_head casio_list_head;
    atomic_t nr_running;
};
#endif

/*
 * This is the main, per-CPU runqueue data structure.
 *
 * Locking rule: those places that want to lock multiple runqueues
 * (such as the load balancing or the thread migration code), lock
 * acquire operations must be ordered by ascending &runqueue.
 */

```

University of the Valley of Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

p. Ahora definiremos las tareas que son calendarizables con nuestra política, y su ready queue. Recuerde que el calendarizador CFS para tareas normales en Linux usa un árbol red-black para organizar sus procesos por prioridad. En nuestra política haremos lo mismo, pero, por ser una implementación de EDF, las etiquetas de los nodos en el árbol serán las deadlines de las tareas. Siempre en sched.c aplique la siguiente modificación:

```

...
struct rt_rq{
...
};

#ifndef CONFIG_SCHED_CASIO_POLICY
struct casio_task{
    struct rb_node casio_rb_node;
    unsigned long long absolute_deadline;
    struct list_head casio_list_node;
    struct task_struct* task;
};

struct casio_rq{
    struct rb_root casio_rb_root;
    struct list_head casio_list_head;
    atomic_t nr_running;
};

#endif
/*
 * This is the main, per-CPU runqueue data structure.
 ...

Note que nuestra política se apoya en el uso de estructuras de datos provistas por el kernel en <linux/list.h> y <linux/rbtree.h>. Un árbol red-black mantendrá nuestra ready queue.
• Explique el contenido de la estructura casio_task.
q. Para que el sistema pueda referirse a las tareas calendarizadas de acuerdo con nuestra política, debemos aplicar la siguiente modificación en sched.c:
struct rq {
...
    struct rt_rq rt;
#endif CONFIG_SCHED_CASIO_POLICY
    struct casio_rq casio_rq;
#endif
...

```

- Explique el contenido de la estructura casio\_task.

La estructura casio\_task contiene el nodo de nuestro rb tree, el deadline del task, la casio\_list\_node nos ayuda a mantener una double linked list para organizar los tasks y el task en sí.

The screenshot shows a Linux desktop interface with two windows. On the left, a terminal window titled 'sched.c (/home/scheduler\_dev/linux-2.6.24-rc1/sio/kernel) - gedit' displays the source code for the 'sched.c' file. The code is written in C and includes various #ifdef directives for different kernel configurations like 'CONFIG\_NO\_HZ', 'CONFIG\_FAIR\_GROUP\_SCHED', and 'CONFIG\_SCHED\_CASIO\_POLICY'. It defines structures such as 'cfs', 'rq', and 'casio\_rq'. The code is annotated with comments explaining its purpose, such as 'This is the main, per-CPU runqueue data structure'. On the right, a PDF viewer window titled 'Laboratorio #5 2021.pdf' is open, showing a document with several questions (q.) and their answers. One question asks to explain the purpose and content of the 'casio\_rq' structure, with an answer pointing to the kernel's 'list.h' and 'rbtree.h' files. Another question asks about the 'atomic\_t' type and its use in RMW operations and MMIO mapping, with an answer providing a brief explanation.

- Explique el propósito y contenido de la estructura `casio_rq`.  
Esta nos ayuda a mantener una linked list con head `casio_list` para asignar los taks a un procesador.
- ¿Qué es y para qué sirve el tipo `atomic_t`? Describa brevemente los conceptos de operaciones RMW (read-modify-write) y mapeo de dispositivos en memoria (MMIO).  
Los tipos atomic proveen una interfaz para los medios de la arquitectura a las operaciones RMW entre CPUs.

Applications Places System sched.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - gedit

Laboratorio #5 2021.pdf

File Edit View Go Help

Previous Next 7 of 18 Fit Page Width

shed.c

```

#endif

    retval = security_task_setscheduler(p, policy, param);
    if (retval)
        return retval;
    /* make sure no PI-waiters arrive (or leave) while we are
     * changing the priority of the task:
    */
    spin_lock_irqsave(&p->pi_lock, flags);
    /*
     * 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;
    }
#endif 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);
    if (on_rq) {
        deactivate_task(rq, p, 0);
        if (running)
            p->sched_class->put_prev_task(rq, p);
    }

    oldprio = p->prio;
    _setscheduler(rq, p, policy, param->sched_priority);

```

Ln 4390, Col 7 INS

Note que esta modificación emplea un método que todavía no hemos definido.

s. Los diferentes calendarizadores de Linux se inicializan en la función `sched_init`. Modifique esta función de la siguiente forma:

```

void __init sched_init(void)
{
    ...
    rq->nrt_running = 0;
    rq->clock = 1;
#endif CONFIG_SCHED_CASIO_POLICY
    init_casio_rq(&rq->casio_rq);
    ...
    init_cfs_rq(&rq->cfs, rq);
}

```

Note, de nuevo, que la función llamada no ha sido definida todavía.

Applications Places System sched.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - gedit

Laboratorio #5 2021.pdf

File Edit View Go Help

Previous Next 7 of 18 Fit Page Width

shed.c

```

cfs_rq->tasks_timeline = RB_ROOT;
#endif CONFIG_FAIR_GROUP_SCHED
cfs_rq->rq = rq;
#endif
cfs_rq->min_vruntime = (u64)(-ILL << 20);
}

void __init sched_init(void)
{
    int highest_cpu = 0;
    int i, j;

    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->nrt_running = 0;
        rq->clock = 1;
#endif CONFIG_SCHED_CASIO_POLICY
        init_casio_rq(&rq->casio_rq);
    }
    init_cfs_rq(&rq->cfs, rq);
#endif 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);

        init_cfs_rq_p[i] = cfs_rq;
        init_cfs_rq(cfs_rq, rq);
        cfs_rq->tg = &init_task_group;
        list_add(&cfs_rq->leaf_cfs_rq_list,
                 &rq->leaf_cfs_rq_list);

        init_sched_entity_p[i] = se;
        se->cfs_rq = &rq->cfs;
    }
}

```

Ln 6818, Col 24 INS

Note que esta modificación emplea un método que todavía no hemos definido.

s. Los diferentes calendarizadores de Linux se inicializan en la función `sched_init`. Modifique esta función de la siguiente forma:

```

void __init sched_init(void)
{
    ...
    rq->nrt_running = 0;
    rq->clock = 1;
#endif CONFIG_SCHED_CASIO_POLICY
    init_casio_rq(&rq->casio_rq);
    ...
    init_cfs_rq(&rq->cfs, rq);
}

```

Note, de nuevo, que la función llamada no ha sido definida todavía.

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

UVG  
UNIVERSIDAD  
DEL VALLE  
DE COATEPEQUE

t. Todo lo que hemos hecho hasta ahora ha servido para configurar el uso de la política de calendarización EDF en el sistema. Ahora implementaremos la política como tal. Cree el archivo `kernel_dir/kernel/sched_casio.c` y programe la función de inicialización de la `ready queue` para nuestras tasks:

Laboratorio #5 2021.pdf

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

t. Todo lo que hemos hecho hasta ahora ha servido para configurar el uso de la política de calendarización EDF en el sistema. Ahora implementaremos la política como tal. Cree el archivo kernel\_dir/kernel/sched\_casio.c y programe la función de inicialización de la ready queue para nuestras tareas:

```
void init_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=R8_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_set(&casio_rq->nr_running, 0);
}
```

u. Luego programe las funciones para el manejo de la lista de casio\_tasks:

```
void add_casio_task_2_list(struct casio_rq* rq, struct task_struct* p){
    struct list_head* ptr = NULL;
    struct casio_task* new = NULL;
    struct casio_task* casio_task = NULL;
    //char msg
    if (rq && p){
        new = (struct casio_task*)kzalloc(sizeof(struct casio_task),
GFP_KERNEL);
        if (new){
            casio_task = NULL;
            new->task = p;
            new->absolute_deadline = 0;
            list_for_each(ptr, &rq->casio_list_head){
                casio_task = list_entry(ptr, struct casio_task,
casio_list_node);
                if (casio_task){
                    if (new->task->casio_id < casio_task-
>casio_id){
                        list_add(&new->casio_list_node, &rq->casio_list_head);
                        //logs
                    } else {
                        printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
                    }
                } else {
                    printk(KERN_ALERT "add_casio_task_2_list: null pointers\n");
                }
            }
            list_add(&new->casio_list_node, &rq->casio_list_head);
            //logs
        } else {
            printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
        }
    }
}

void rem_casio_task_list(struct casio_rq* rq, struct task_struct* p){
    struct list_head* ptr = NULL;
    struct list_head* next = NULL;
    struct casio_task* casio_task = NULL;
    //char msg
    if (rq && p){
        new = (struct casio_task*)kzalloc(sizeof(struct casio_task),
GFP_KERNEL);
        if (new){
            casio_task = NULL;
            new->task = p;
            new->absolute_deadline = 0;
            list_for_each(ptr, &rq->casio_list_head){
                casio_task = list_entry(ptr, struct casio_task,
casio_list_node);
                if (casio_task){
                    if (new->task->casio_id < casio_task-
>casio_id){
                        list_add(&new->casio_list_node, &rq->casio_list_head);
                        //logs
                    } else {
                        printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
                    }
                } else {
                    printk(KERN_ALERT "add_casio_task_2_list: null pointers\n");
                }
            }
            list_add(&new->casio_list_node, &rq->casio_list_head);
            //logs
        } else {
            printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
        }
    }
}
```

Laboratorio #5 2021.pdf

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

u. Luego programe las funciones para el manejo de la lista de casio\_tasks:

```
void add_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=R8_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_set(&casio_rq->nr_running, 0);
}

void init_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=R8_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_set(&casio_rq->nr_running, 0);
}

void add_casio_task_2_list(struct casio_rq* rq, struct task_struct* p){
    struct list_head* ptr = NULL;
    struct casio_task* new = NULL;
    struct casio_task* casio_task = NULL;
    //char msg
    if (rq && p){
        new = (struct casio_task*)kzalloc(sizeof(struct casio_task),
GFP_KERNEL);
        if (new){
            casio_task = NULL;
            new->task = p;
            new->absolute_deadline = 0;
            list_for_each(ptr, &rq->casio_list_head){
                casio_task = list_entry(ptr, struct casio_task,
casio_list_node);
                if (casio_task){
                    if (new->task->casio_id < casio_task-
>casio_id){
                        list_add(&new->casio_list_node, &rq->casio_list_head);
                        //logs
                    } else {
                        printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
                    }
                } else {
                    printk(KERN_ALERT "add_casio_task_2_list: null pointers\n");
                }
            }
            list_add(&new->casio_list_node, &rq->casio_list_head);
            //logs
        } else {
            printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
        }
    }
}

void rem_casio_task_list(struct casio_rq* rq, struct task_struct* p){
    struct list_head* ptr = NULL;
    struct list_head* next = NULL;
    struct casio_task* casio_task = NULL;
    //char msg
    if (rq && p){
        new = (struct casio_task*)kzalloc(sizeof(struct casio_task),
GFP_KERNEL);
        if (new){
            casio_task = NULL;
            new->task = p;
            new->absolute_deadline = 0;
            list_for_each(ptr, &rq->casio_list_head){
                casio_task = list_entry(ptr, struct casio_task,
casio_list_node);
                if (casio_task){
                    if (new->task->casio_id < casio_task-
>casio_id){
                        list_add(&new->casio_list_node, &rq->casio_list_head);
                        //logs
                    } else {
                        printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
                    }
                } else {
                    printk(KERN_ALERT "add_casio_task_2_list: null pointers\n");
                }
            }
            list_add(&new->casio_list_node, &rq->casio_list_head);
            //logs
        } else {
            printk(KERN_ALERT "add_casio_task_2_list: kzalloc\n");
        }
    }
}
```

Laboratorio #5 2021.pdf

```

    if (casio_task->task->casio_id == p->casio_id)
        return casio_task;
    }
}
return NULL;
}

v. Ahora programa las funciones para el manejo del red-black tree de casio_tasks:
void insert_casio_task_rb_tree(struct casio_rq* rq, struct casio_task* p){
    struct rb_node** node = NULL;
    struct rb_node* parent = NULL;
    struct casio_task* entry = NULL;
    node = &p->casio_rb_root.rb_node;
    while(*node != NULL){
        parent = *node;
        entry = rb_entry(parent, struct casio_task, casio_rb_node);
        if (entry->absolute_deadline < entry->absolute_deadline){
            node = &parent->rb_left;
        } else {
            node = &parent->rb_right;
        }
    }
    rb_link_node(&p->casio_rb_node, parent, node);
    rb_insert_color(&p->casio_rb_node, &rq->casio_rb_root);
}

void remove_casio_task_rb_tree(struct casio_rq* rq, struct casio_task* p){
    rb_erase(&(p->casio_rb_node), &(rq->casio_rb_root));
    p->casio_rb_node.rb_left = p->casio_rb_node.rb_right = NULL;
}

struct casio_task* earliest_deadline_casio_task_rb_tree(struct casio_rq* rq){
    struct rb_node* node = NULL;
    struct casio_task* p = NULL;
    node = rq->casio_rb_root.rb_node;
    if (node == NULL)
        return NULL;
    while (node->rb_left != NULL){
        node = node->rb_left;
    }
    p = rb_entry(node, struct casio_task, casio_rb_node);
    return p;
}

```

Ln 109, Col 1      INS

Laboratorio #5 2021.pdf

```

    }
    rb_link_node(&p->casio_rb_node, parent, node);
    rb_insert_color(&p->casio_rb_node, &rq->casio_rb_root);
}

void remove_casio_task_rb_tree(struct casio_rq* rq, struct casio_task* p){
    rb_erase(&(p->casio_rb_node), &(rq->casio_rb_root));
    p->casio_rb_node.rb_left = p->casio_rb_node.rb_right = NULL;
}

struct casio_task* earliest_deadline_casio_task_rb_tree(struct casio_rq* rq){
    struct rb_node* node = NULL;
    struct casio_task* p = NULL;
    node = rq->casio_rb_root.rb_node;
    if (node == NULL)
        return NULL;
    while (node->rb_left != NULL){
        node = node->rb_left;
    }
    p = rb_entry(node, struct casio_task, casio_rb_node);
    return p;
}

const struct sched_class casio_sched_class = {
    .next = &rt_sched_class,
    .enqueue_task = enqueue_task_casio,
    .dequeue_task = dequeue_task_casio,
    .check_preempt_curr = check_preempt_curr_casio,
    .pick_next_task = pick_next_task_casio,
    .put_prev_task = put_prev_task_casio,
#define CONFIG_SMP
    .load_balance = load_balance_casio,
    .move_one_task = move_one_task_casio,
#endif
    .set_curr_task = set_curr_task_casio,
    .task_tick = task_tick_casio,
};


```

Ln 124, Col 1      INS

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

Esta se usa para organizar los modulos por prioridad en una linked list. Como `casio_sched_class` es la de mayor prioridad, esta apunta a `rt_sched_class` que es la siguiente con mayor prioridad.

```

static void enqueue_task_casio(struct rq* rq, struct task_struct* p, int wakeup)
{
    struct casio_task* t = NULL;
    //char msg
    if (p){
        t = find_casio_task_list(&rq->casio_rq, p);
        if (t){
            t->absolute_deadline = sched_clock() + p->deadline;
            insert_casio_task_rb_tree(&rq->casio_rq, t);
            atomic_inc(&rq->casio_rq.nr_running);
            //logs
        } else {
            printk(KERN_ALERT *enqueue_task_casio\n");
        }
    }
}

static void dequeue_task_casio(struct rq* rq, struct task_struct* p, int sleep)
{
    struct casio_task* t = NULL;
    //char msg
    if(p){
        t = find_casio_task_list(&rq->casio_rq,p);
        if (t){
            //logs
            remove_casio_task_rb_tree(&rq->casio_rq, t);
            atomic_dec(&rq->casio_rq.nr_running);
            if(t->task->state == TASK_DEAD || t->task->state == EXIT_DEAD
               || t->task->state==EXIT_ZOMBIE){
                rem_casio_task_list(&rq->casio_rq, t->task);
            }
        } else {
            printk(KERN_ALERT *dequeue_task_casio\n");
        }
    }
}

const struct sched class casio_sched_class = {

```

- 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_setschedule`. 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?

Primero, crea un `casio_struct`, luego le busca una posición en la lista de tasks, le define el deadline, la inserta en el rb tree, y finalmente lo registra. Al momento de hacer el dequeue, se encuentra en la lista, se remueve del rb tree y se valida si terminó o murió para removerlo de la lista. Se guarda en una red-black tree para poder removerlos y agregarlos en tiempo O(log n).

The terminal window displays the source code for the file `sched_casio.c`. The code implements a scheduler for `casio_task` structures. It includes functions for task state transitions, preempting tasks, and managing a linked list of tasks. A comment in the code asks about when it preempts a task.

```

    if(t->task->state == TASK_DEAD || t->task->state == EXIT_DEAD
        || t->task->state==EXIT_ZOMBIE){
            rem_casio_task_list(&rq->casio_rq, t->task);
        }
    } else {
        printk(KERN_ALERT *dequeue_task_casio\n");
    }
}

static void check_preempt_curr_casio(struct rq* rq, struct task_struct* p)
{
    struct casio_task* t = NULL;
    struct casio_task* curr = NULL;
    if (rq->curr->policy != SCHED_CASIO){
        resched_task(rq->curr);
    } else {
        t = earliest_deadline_casio_task_rb_tree(&rq->casio_rq);
        if (t){
            curr = find_casio_task_list(&rq->casio_rq, rq->curr);
            if (curr){
                if (t->absolute_deadline < curr->absolute_deadline)
                    resched_task(rq->curr);
            } else {
                printk(KERN_ALERT *check_preempt_curr_casio\n");
            }
        }
    }
}

const struct sched_class casio_sched_class = {
    .next          = &rt_sched_class,
    .enqueue_task  = enqueue_task_casio,
    .dequeue_task  = dequeue_task_casio,
    .check_preempt_curr = check_preempt_curr_casio,
    .pick_next_task = pick_next_task_casio,
    .put_prev_task  = put_prev_task_casio,
#define CONFIG_SMP
    .load_balance   = load_balance_casio,
};

static void check_preempt_curr_casio(struct rq* rq, struct task_struct* p)
{
    struct casio_task* t = NULL;
    struct casio_task* curr = NULL;
    if (rq->curr->policy != SCHED_CASIO){
        resched_task(rq->curr);
    } else {
        t = earliest_deadline_casio_task_rb_tree(&rq->casio_rq);
        if (t){
            curr = find_casio_task_list(&rq->casio_rq, rq->curr);
            if (curr){
                if (t->absolute_deadline < curr->absolute_deadline)
                    resched_task(rq->curr);
            } else {
                printk(KERN_ALERT *check_preempt_curr_casio\n");
            }
        }
    }
}

static struct task_struct* pick_next_task_casio(struct rq* rq)
{
    struct casio_task* t = NULL;
    t = earliest_deadline_casio_task_rb_tree(&rq->casio_rq);
    if (t)
        return t->task;
    return NULL;
}

static void put_prev_task_casio(struct rq* rq, struct task_struct* prev)
{
}

#endif CONFIG_SMP
static unsigned long load_balance_casio(struct rq* this_rq, int this_cpu,
                                         struct rq* busiest,
                                         ...
}

```

The PDF viewer window shows a document titled "Laboratorio #5 2021.pdf". The document contains text from the University of Valle de Guatemala's Systems Operations course, mentioning Erick Pineda and Tomás Gálvez P. Semestre I, 2021. It also includes the UVG logo.

- ¿Cuándo preempea una `casio_task` a la `task` actualmente en ejecución?
- Preempea cuando hay al menos un `casio_task` en la queue y la `task` asignado no es un `casio_task` y también cuando hay un `casio_task` en ejecución y hay al menos otro `casio_task` en el rb tree con un deadline menor.

Wed Apr 21, 11:57 PM Douglas

Laboratorio #5 2021.pdf

\*sched\_casio.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - gedit

File Edit View Search Tools Documents Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

```

static struct task_struct* pick_next_task_casio(struct rq* rq)
{
    struct casio_task* t = NULL;
    t = earliest_deadline_casio_task_rb_tree(&rq->casio_rq);
    if (t){
        return t->task;
    }
    return NULL;
}

static void put_prev_task_casio(struct rq* rq, struct task_struct* prev)
{
}

#ifndef CONFIG_SMP
static unsigned long load_balance_casio(struct rq* this_rq, int this_cpu,
    struct rq* busiest,
    unsigned long max_load_move,
    struct sched_domain* sd, enum cpu_idle_type idle,
    int* all_pinned, int* this_best_prio)
{
    return 0;
}
static int move_one_task_casio(struct rq* this_rq, int this_cpu,
    struct rq* busiest,
    struct sched_domain* sd,
    enum cpu_idle_type idle)
{
    return 0;
}
#endif

static void set_curr_task_casio(struct rq* rq)
{
}

static void task_tick_casio(struct rq* rq, struct task_struct* p)

```

Ln 205, Col 2 INS

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... \*sched\_casio.c (/ho...

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
Semestre I, 2021

UVG  
UNIVERSIDAD  
DEL VALLE  
GUATEMALA

```

static void set_curr_task_casio(struct rq* rq)
{
}

static void task_tick_casio(struct rq* rq, struct task_struct* p)
{
}

```

Wed Apr 21, 11:59 PM Douglas

Laboratorio #5 2021.pdf

sched.h (/home/scheduler\_dev/linux-2.6.24-casio/include/linux) - gedit

File Edit View Go Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

```

static inline void inc_syscw(struct task_struct *tsk)
{
}
#endif

#ifndef CONFIG_SMP
void migration_init(void);
#else
static inline void migration_init(void)
{
}
#endif

#ifndef /* _KERNEL_ */
#define CONFIG_SCHED_CASIO_POLICY
#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();
struct casio_event_log* get_casio_event_log();
void register_casio_event(unsigned long long t, char* m, int a);
#endif

#endif

```

Ln 2003, Col 1 INS

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... sched.h (/home/sche...

```

static void set_curr_task_casio(struct rq* rq)
{
}

static void task_tick_casio(struct rq* rq, struct task_struct* p)
{
}

y. Habiendo llegado a este punto ya tenemos lista nuestra política de calendarización, pero vamos a agregar elementos que nos permitan llevar registro de los eventos que suceden durante la calendarización. Comenzaremos por ir a kernel_dir/include/linux/sched.h y aplicar la siguiente modificación:
...
#endif /* _KERNEL_ */
#endif /* CONFIG_SCHED_CASIO_POLICY */

#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();
struct casio_event_log* get_casio_event_log();
void register_casio_event(unsigned long long t, char* m, int a);
#endif

2. Ahora definiremos estas funciones en kernel_dir/kernel/sched_casio.c. Agregue al inicio de este archivo lo siguiente:

```

Applications Places System

Thu Apr 22, 12:54 AM

Laboratorio #5 2021.pdf

Douglas

**\*sched\_casio.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - edit**

File Edit View Search Tools Documents Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

**\*sched\_casio.c**

```

insert_casio_task_rb_tree(&rq->casio_rq, t);
atomic_inc(&rq->casio_rq.nr_running);
snprintf(msg, CASIO_MSG_SIZE, "(%d:%d:%lu)", p->casio_id, p->pid, t->absolute_deadline);
register_casio_event(sched_clock(), msg, CASIO_ENQUEUE);
} else {
    printk(KERN_ALERT "enqueue_task_casio()\n");
}
}

static void dequeue_task_casio(struct rq* rq, struct task_struct* p, int sleep)
{
    struct casio_task* t = NULL;
    char msg[CASIO_MSG_SIZE];
    if(p){
        t = find_casio_task_list(&rq->casio_rq,p);
        if (t){
            snprintf(msg, CASIO_MSG_SIZE, "(%d:%d:%lu)", t->task->casio_id, t->task->pid, t->absolute_deadline);
            register_casio_event(sched_clock(), msg, CASIO_DEQUEUE);
            remove_casio_task_rb_tree(&rq->casio_rq);
            atomic_dec(&rq->casio_rq.nr_running);
            if(t->task->state == TASK_DEAD || t->task->state == EXIT_DEAD
               || t->task->state == EXIT_ZOMBIE){
                rem_casio_task_list(&rq->casio_rq, t->task);
            }
        } else {
            printk(KERN_ALERT "dequeue_task_casio()\n");
        }
    }
}

static void check_preempt_curr_casio(struct rq* rq, struct task_struct* p)
{
    struct casio_task* t = NULL;
    struct casio_task* curr = NULL;
    if (rq->curr->policy != SCHED_CASIO){
        resched_task(rq->curr);
    } else {
        t = earliest_casio_task_rb_tree(rq->casio_rq,
        ...
    }
}

Ln 161, Col 33 INS
```

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... \*sched\_casio.c (/ho...

Next 15 of 18 Fit Page Width

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
SemestreI,2021

UVG  
UNIVERSIDAD  
DEL VALLE  
DE COLOMBIA

a. Vamos a registrar algunos eventos:

- En add\_casio\_task\_2\_list declare msg en donde está el comentario //char msg, y donde está el comentario //logs registre un evento con el mensaje "add\_casio\_task\_2\_list: %d:%d:%lu" con valores new->task->casio\_id, new->task->pid, new->absolute\_deadline; y con bandera CASIO\_MSG.
- En rem\_casio\_task\_list declare msg donde está //char msg, y donde está //logs registre un evento con el mensaje "rem\_casio\_task\_list: %d:%d:%lu" con valores casio\_id, casio\_task->absolute\_deadline; con bandera CASIO\_MSG.
- En enqueue\_task\_casio declare msg donde está //char msg, y donde está //logs registre un evento con el mensaje "(%d:%d:%lu)", con valores p->casio\_id, p->pid, t->absolute\_deadline; y con bandera CASIO\_ENQUEUE.
- En dequeue\_task\_casio declare msg donde está //char msg, y donde está //logs registre un evento con el mensaje "(%d:%d:%lu)", con valores t->task->casio\_id, t->task->pid, t->absolute\_deadline; y con bandera CASIO\_DEQUEUE.

b. Un evento que debemos registrar pero que no controlamos desde sched\_casio.c es el cambio de contexto que involucra una o dos tareas. Para ello debemos dirigirnos a kernel\_dir/kernel/sched.c y aplicar la siguiente modificación:

```

...
prev->sched_class->put_prev_task(rq, prev);
next = pick_next_task(rq, prev);

#ifdef CONFIG_SCHED_CASIO_POLICY
    char msg[CASIO_MSG_SIZE];
    if (prev->policy == SCHED_CASIO || next->policy == SCHED_CASIO){
        if (prev->policy == SCHED_CASIO && next->policy == SCHED_CASIO){
            snprintf(msg, CASIO_MSG_SIZE, "%prev->(%d:%d), next->(%d:%d)", prev->casio_id, prev->pid, next->casio_id, next->pid);
        } else {
            if (prev->policy == SCHED_CASIO){
                snprintf(msg, CASIO_MSG_SIZE, "%prev->(%d:%d), next->(-1:%d)", prev->casio_id, prev->pid, next->pid);
            } else {
                snprintf(msg, CASIO_MSG_SIZE, "%prev->(-1:%d), next->(%d:%d)", prev->pid, next->casio_id, next->pid);
            }
        }
        register_casio_event(sched_clock(), msg, CASIO_CONTEXT_SWITCH);
    }
#endif
    sched_info_switch(prev, next);
...
```

Reemplazando //logs1, //logs2 y //logs3 por llamadas a sprintf cuyos primeros dos argumentos sean msg y CASIO\_MSG\_SIZE; y cuyos últimos argumentos sean, respectivamente:

Applications Places System

Thu Apr 22, 1:00 AM

Laboratorio #5 2021.pdf

Douglas

**\*sched.c (/home/scheduler\_dev/linux-2.6.24-casio/kernel) - edit**

File Edit View Search Tools Documents Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

**\*sched.c**

```

clear_task_need_resched(prev);

if (prev->state && !(preempt_count() & PREEMPT_ACTIVE)) {
    if (unlikely((prev->state & TASK_INTERRUPTIBLE) &&
                unlikely(signal_pending(prev)))) {
        prev->state = TASK_RUNNING;
    } else {
        deactivate_task(rq, prev, 1);
    }
    switch_count = &prev->nvcsw;
}

if (unlikely(rq->nr_running))
    idle_balance(cpu, rq);

prev->sched_class->put_prev_task(rq, prev);
next = pick_next_task(rq, prev);

#ifdef CONFIG_SCHED_CASIO_POLICY
    char msg[CASIO_MSG_SIZE];
    if (prev->policy == SCHED_CASIO || next->policy == SCHED_CASIO){
        if (prev->policy == SCHED_CASIO && next->policy == SCHED_CASIO){
            snprintf(msg, CASIO_MSG_SIZE, "%prev->(%d:%d), next->(%d:%d)", prev->casio_id, prev->pid, next->casio_id, next->pid);
        } else {
            if (prev->policy == SCHED_CASIO){
                snprintf(msg, CASIO_MSG_SIZE, "%prev->(%d:%d), next->(-1:%d)", prev->casio_id, prev->pid, next->pid);
            } else {
                snprintf(msg, CASIO_MSG_SIZE, "%prev->(-1:%d), next->(%d:%d)", prev->pid, next->casio_id, next->pid);
            }
        }
        register_casio_event(sched_clock(), msg, CASIO_CONTEXT_SWITCH);
    }
#endif
    sched_info_switch(prev, next);

    if (likely(prev != next)) {
        ro_nr_switches++;
    }
}

Ln 3685, Col 1 INS
```

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... \*sched.c (/home/sch...

Next 15 of 18 Fit Page Width

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
SemestreI,2021

UVG  
UNIVERSIDAD  
DEL VALLE  
DE COLOMBIA

Reemplazando //logs1, //logs2 y //logs3 por llamadas a sprintf cuyos primeros dos argumentos sean msg y CASIO\_MSG\_SIZE; y cuyos últimos argumentos sean, respectivamente:

1. "%prev->(%d:%d), next->(%d:%d)", prev->casio\_id, prev->pid, next->casio\_id, next->pid
2. "%prev->(%d:%d), next->(-1:%d)", prev->casio\_id, prev->pid, next->pid
3. "%prev->(-1:%d), next->(%d:%d)", prev->pid, next->casio\_id, next->pid

c. Finalmente, modificaremos kernel\_dir/fs/proc/proc\_misc.c para que nuestra búfera se almacene en un archivo que como usuarios podemos abrir y leer (recordemos que nuestro log y todo lo que éste almacena están en kernel space).

Applications Places System Applications proc\_misc.c (/home/scheduler\_dev/linux-2.6.24-casio/fs/proc) - gedit

File Edit View Search Tools Documents Help

New Open Save Print... Undo Redo Cut Copy Paste Find Replace

```

proc_misc.c (x)
    create_seq_entry("diskstats", 0, &proc_diskstats_operations);
#endif
#ifndef CONFIG_MODULES
    create_seq_entry("modules", 0, &proc_modules_operations);
#endif
#ifndef CONFIG_SCHEDSTATS
    create_seq_entry("schedstat", 0, &proc_schedstat_operations);
#endif
#ifndef CONFIG_PROC_KCORE
    proc_root_kcore = create_proc_entry("kcore", S_IRUSR, NULL);
    if (proc_root_kcore) {
        proc_root_kcore->proc_fops = &proc_kcore_operations;
        proc_root_kcore->size =
            (size_t)high_memory - PAGE_OFFSET + PAGE_SIZE;
    }
#endif
#ifndef CONFIG_PROC_VMCORE
    proc_vmcore = create_proc_entry("vmcore", S_IRUSR, NULL);
    if (proc_vmcore)
        proc_vmcore->proc_fops = &proc_vcore_operations;
#endif
#ifndef CONFIG_MAGIC_SYSRQ
    {
        struct proc_dir_entry *entry;
        entry = create_proc_entry("sysrq-trigger", S_IWUSR, NULL);
        if (entry)
            entry->proc_fops = &proc_sysrq_trigger_operations;
    }
#endif
#ifndef CONFIG_SCHED_CASIO_POLICY
    {
        struct proc_dir_entry* casio_entry;
        casio_entry = create_proc_entry("casio_event", 0666, &proc_root);
        if (casio_entry){
            casio_entry->proc_fops = &proc_casio_operations;
            casio_entry->data = NULL;
        }
    }
#endif
Ln 791, Col 25 INS

```

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021... proc\_misc.c (/home/...)

Thu Apr 22, 1:02 AM Douglas

Laboratorio #5 2021.pdf

View Go Help

Next 17 of 18 Fit Page Width

```

    .read = casio_read,
    .release = casio_release,
}
#endif
extern struct seq_operations fragmentation_op;
...

```

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda; Tomás Gálvez P.  
SemestreI, 2021

UVG  
UNIVERSIDAD DEL VALLE DE GUATEMALA

```

...
    entry->proc_fops = &proc_sysrq_trigger_operations;
#endif
#ifndef CONFIG_SCHED_CASIO_POLICY
    {
        struct proc_dir_entry* casio_entry;
        casio_entry = create_proc_entry("casio_event", 0666, &proc_root);
        if (casio_entry){
            casio_entry->proc_fops = &proc_casio_operations;
            casio_entry->data = NULL;
        }
    }
#endif

```

Con esto terminamos las modificaciones al sistema que implementan la nueva política de calendarización. Antes de compilar el kernel acceda al Makefile en kernel\_dir y asigne a la variable EXTRAVERSION el valor -casio. Además, copie el archivo de configuración del kernel actual a esta carpeta con el siguiente comando:

```
sudo cp /boot/config-2.6.24-26-generic .config
```

Note el espacio antes de .config. Ahora copie todo el contenido de scheduler\_dev/linux-2.6.24-casio a scheduler (use la opción -a del comando cp). Se recomienda crear una snapshot (al menos) en este punto. Diríjase a scheduler/linux-2.6.24-casio y ejecute lo siguiente:

```
sudo make oldconfig
```

Este proceso de compilación toma un archivo de configuración existente y crea uno nuevo, pidiendo input

Applications Places System Applications root@douglas-laptop: /home/scheduler/linux-2.6.24-casio

File Edit Terminal Tab Help

```

LRW support (EXPERIMENTAL) (CRYPTO_LRW) [M/n/y/?] m
XTS support (EXPERIMENTAL) (CRYPTO_XTS) [M/n/y/?] m
Software async crypto daemon (CRYPTO_CRYPTD) [M/n/y/?] m
DES and Triple DES EDE cipher algorithms (CRYPTO_DES) [M/y/?] m
FCrypt cipher algorithm (CRYPTO_FCRYPT) [M/y/?] m
Blowfish cipher algorithm (CRYPTO_BLOWFISH) [M/n/y/?] m
Twofish cipher algorithm (CRYPTO_TWOFISH) [M/n/y/?] m
Twofish cipher algorithms (i586) (CRYPTO_TWOFISH_586) [M/n/y/?] m
Serpent cipher algorithm (CRYPTO_SERPENT) [M/n/y/?] m
AES cipher algorithms (CRYPTO_AES) [M/y/?] m
AES cipher algorithms (i586) (CRYPTO_AES_586) [M/n/y/?] m
CAST5 (CAST-128) cipher algorithm (CRYPTO_CAST5) [M/y/?] m
CAST6 (CAST-256) cipher algorithm (CRYPTO_CAST6) [M/n/y/?] m
TEA, XTEA and XETA cipher algorithms (CRYPTO_TEA) [M/n/y/?] m
ARC4 cipher algorithm (CRYPTO_ARC4) [M/y/?] m
Khazad cipher algorithm (CRYPTO_KHAZAD) [M/n/y/?] m
Anubis cipher algorithm (CRYPTO_ANUBIS) [M/n/y/?] m
SEED cipher algorithm (CRYPTO_SEED) [M/n/y/?] m
Deflate compression algorithm (CRYPTO_DEFLATE) [M/y/?] m
Michael MIC keyed digest algorithm (CRYPTO_MICHAEL_MIC) [M/y/?] m
CRC32c CRC algorithm (CRYPTO_CRC32C) [M/y/?] m
Camellia cipher algorithms (CRYPTO_CAMELLIA) [M/n/y/?] m
Testing module (CRYPTO_TEST) [M/n/?] m
Authenc support (CRYPTO_AUTHENC) [M/n/y/?] m
*
* Hardware crypto devices
*
Hardware crypto devices (CRYPTO_HW) [Y/n/?] y
Support for VIA Padlock ACE (CRYPTO_DEV_PADLOCK) [Y/n/m/?] y
Padlock driver for AES algorithm (CRYPTO_DEV_PADLOCK_AES) [M/n/y/?] m
Padlock driver for SHA1 and SHA256 algorithms (CRYPTO_DEV_PADLOCK_SHA) [M/n/y/?] m
*
Support for the Geode LX AES engine (CRYPTO_DEV_GEODE) [M/n/y/?] m
*
Library routines
*
CRC-CCITT functions (CRC_CCITT) [M/y/?] m
CRC16 functions (CRC16) [M/y/?] m
CRC ITU-T V.41 functions (CRC_ITU_T) [M/y/?] m
CRC32 functions (CRC32) [Y/?] y
CRC7 functions (CRC7) [M/n/y/?] m
CRC32c (Castagnoli, et al) Cyclic Redundancy-Check (LIBCRC32C) [M/y/?] m
#
# configuration written to .config
#
root@douglas-laptop:/home/scheduler/linux-2.6.24-casio#

```

root@douglas-laptop... lab-5 - File Browser Laboratorio #5 2021...

Thu Apr 22, 1:10 AM Douglas

Laboratorio #5 2021.pdf

File Edit View Go Help

Previous Next 17 of 18 Fit Page Width

```

...
    entry->proc_fops = &proc_sysrq_trigger_operations;
#endif
#ifndef CONFIG_SCHED_CASIO_POLICY
    {
        struct proc_dir_entry* casio_entry;
        casio_entry = create_proc_entry("casio_event", 0666, &proc_root);
        if (casio_entry){
            casio_entry->proc_fops = &proc_casio_operations;
            casio_entry->data = NULL;
        }
    }
#endif

```

Con esto terminamos las modificaciones al sistema que implementan la nueva política de calendarización. Antes de compilar el kernel acceda al Makefile en kernel\_dir y asigne a la variable EXTRAVERSION el valor -casio. Además, copie el archivo de configuración del kernel actual a esta carpeta con el siguiente comando:

```
sudo cp /boot/config-2.6.24-26-generic .config
```

Note el espacio antes de .config. Ahora copie todo el contenido de scheduler\_dev/linux-2.6.24-casio a scheduler (use la opción -a del comando cp). Se recomienda crear una snapshot (al menos) en este punto. Diríjase a scheduler/linux-2.6.24-casio y ejecute lo siguiente:

```
sudo make oldconfig
```

Este proceso de compilación toma un archivo de configuración existente y crea uno nuevo, pidiendo input al usuario sobre las características nuevas o desconocidas que tenga el kernel a compilarse. Para cada pregunta que se le realice habrá un valor entre corchetes y, en caso de ser una pregunta con respuesta "si" o "no", se señalará con una letra mayúscula la opción por defecto. Asegúrese de que CASIO Scheduler sea configurada con 'y' y todas las demás opciones con su valor por defecto. Al terminar, compile el kernel con el siguiente comando:

```
sudo make-kpkg --initrd kernel_image 2.../errors
```

Cualquier error detectado durante la compilación se almacenará en el archivo errors, en el directorio scheduler. Una vez termine la compilación, instale el kernel con el siguiente comando:

```
sudo dpkg -i linux-image-...deb
```

Al terminar este proceso, reinicie su máquina. Si todo salió bien, al iniciar el sistema podrá presionar una tecla para acceder al menú de GRUB, desde donde podrá entrar a su nuevo sistema.

```

root@douglas-laptop: /home/scheduler/linux-2.6.24-casio#
File Edit View Terminal Help
-e 's=/I/YES/g' -e 's.=D./boot.g' \
-e '/s=MD/initramfs-tools (:= 0.53) | yaird (:= 0.0.11) | linux-initramfs-tool. /g'
-e 's=@MK@mkinitafs-kpkg aknitrdf.yaird@g' -e 's=@A@i386@g' \
-e '@M@q' -e 's=@OF//g' \
-e 's=@S@/g' -e 's=@B@386@g' \
/debian/pkgs/image/preinst > /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-
-casio/DEBIAN/preinst
chmod 755 /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio/DEBIAN/preinst
sed -e 's=/V2.6.24-casio/g' -e 's=/IB/g' \
-e 's=/ST/linux/g' -e 's=/R/g' \
-e 's=@K/bzImage/g' -e 's=@L/lilo/g' \
-e '@MK@mkinitramfs-kpkg aknitrdf.yaird@g' -e 's=@A@i386@g' \
-e 's=/I/YES/g' -e 's.=D./boot.g' \
-e 's=/MD/initramfs-tools (:= 0.53) | yaird (:= 0.0.11) | linux-initramfs-tool. /g'
-e 's=@M@q' -e 's=@OF//g' \
-e 's=@S@/g' -e 's=@B@386@g' \
/debian/pkgs/image/preinst > /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-c
asio/DEBIAN/preinst
chmod 755 /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio/DEBIAN/prem
sed -e 's=/V2.6.24-casio/g' -e 's=/IB/g' \
-e 's=/ST/linux/g' -e 's=/R/g' \
-e 's=@K/bzImage/g' -e 's=@L/lilo/g' \
-e '@MK@mkinitramfs-kpkg aknitrdf.yaird@g' -e 's=@A@i386@g' \
-e 's=/MD/initramfs-tools (:= 0.53) | yaird (:= 0.0.11) | linux-initramfs-tool. /g'
-e 's=@M@q' -e 's=@OF//g' \
-e 's=@S@/g' -e 's=@B@386@g' \
/debian/templates.in > ./debian/templates.master
echo using old template
using old template
install -c -m 644 ./debian/templates.master /home/scheduler/linux-2.6.24-casio/debian/linux-i
mage-2.6.24-casio/DEBIAN/templates
dpkg-gencontrol -A'Debian-1386...isp' \
-plinux-image-2.6.24-casio -P/home/scheduler/linux-2.6.24-casio/debian/lin
ux-image-2.6.24-casio/
create_md5sums_fn() { cd $1 ; find . -type f ! -regex '.*\./DEBIAN/*' ! -rege
x '\.lib/modules/\*/modules\.list' -print0 | xargs -r0 md5sum > DEBIAN/md5sums ; if [
-x DEBIAN/md5sums ] ; then rm -f "DEBIAN/md5sums" fi } ; create_md5sums_fn
/home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio
chmod -R o+rwx /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio
chown -R root:root /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio
dpkg --build /home/scheduler/linux-2.6.24-casio/debian/linux-image-2.6.24-casio
...
dpkg-deb: building package 'linux-image-2.6.24-casio' in `..../linux-image-2.6.24-casio_2.6.24-casio
-10.00.Custom.i386.deb'
make[1]: Leaving directory `/home/scheduler/linux-2.6.24-casio'
=====
making target stamp-kernel-image [new prereqs: linux-image-2.6.24-casio linux-image-2.6.24-
casio]=====
This is kernel package version 11.00L.
echo done > stamp-kernel-image
=====
This is kernel package version 11.00L.
root@douglas-laptop:/home/scheduler/linux-2.6.24-casio#

```

Universidad del Valle de Guatemala  
Sistemas Operativos  
Docentes: Erick Pineda, Tomás Gálvez P.  
Seminario 1, 2021

UVG  
UNIVERSIDAD DEL VALLE  
DE QUITO, ECUADOR

Este proceso de compilación toma un archivo de configuración existente y crea uno nuevo, pidiendo input al usuario sobre las características nuevas o desconocidas que tenga el kernel a compilarse. Para cada pregunta que se le realice habrá un valor entre corchetes y, en caso de ser una pregunta con respuesta "si" o "no", se señalará con una letra mayúscula la opción por defecto. Asegúrese de que CASIO Scheduler sea configurada con 'y' y todas las demás opciones con su valor por defecto. Al terminar, compile el kernel con el siguiente comando:

```

sudo make-kpxp --initrd kernel_image 2>../.errors
Cualquier error detectado durante la compilación se almacenará en el archivo errors, en el directorio scheduler. Una vez termine la compilación, instale el kernel con el siguiente comando:
sudo dpkg -i linux-image-...
Al terminar este proceso, reinicie su máquina. Si todo salió bien, al iniciar el sistema podrá presionar una tecla para acceder al menú de GRUB, desde donde podrá entrar a su nuevo sistema.

```

- pre\_casio vs post\_casio:**

Las principales diferencias son que el post\_casio maneja más procesos. Podemos ver que empieza por Task(1) y sigue con Task(3), mientras que el pre\_casio empieza con Task(2) y no hace nada más. Sumado a esto, podemos ver que tienen un deadline diferente.