

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# █
```

The screenshot shows a dual-pane interface. The left pane is a text editor for a Kconfig file, which contains the following code:

```

bool
depends on IA32_EMULATION
default y

config COMPAT_FOR_U64_ALIGNMENT
    def_bool COMPAT
    depends on X86_64

config SYSVIPC_COMPAT
    bool
    depends on X86_64 && COMPAT && SYSVIPC
    default y

endmenu

menu "CASIO Scheduler"
config SCHED_CASIO_POLICY
    bool "CASIO scheduling policy"
    default y
endmenu

source "net/Kconfig"
source "drivers/Kconfig"
source "drivers/firmware/Kconfig"
source "fs/Kconfig"
source "kernel/Kconfig.instrumentation"
source "arch/x86/Kconfig.debug"
source "security/Kconfig"
source "crypto/Kconfig"
source "lib/Kconfig"

```

The right pane is a PDF document titled "Laboratorio #5 2021.pdf". It contains several sections of text, with some parts highlighted in yellow. One section, labeled 'g.', instructs the user to download the kernel:

g. Ingrese a `scheduler_dev` y descargue el `kernel 2.6.24` de Linux con el siguiente comando:

```
sudo wget https://mirrors.edge.kernel.org/pub/linux/kernel/v2.6/linux-2.6.24.tar.bz2 --no-check-certificate
```

Another section, labeled 'h.', describes how to add a scheduling policy to the kernel:

h. Para agregar una política de calendarización a Linux primero será necesario registrarla como una opción en el menú de configuración del `kernel`. Cree un `backup` de, y abra para modificación, el archivo `kernel_dir/arch/x86/Kconfig`, y agregue en alguna ubicación fácil de hallar las siguientes instrucciones:

```
menu "CASIO Scheduler"
config SCHED_CASIO_POLICY
    bool "CASIO scheduling policy"
    default y
endmenu
```

A note below this states: "Nota: CASIO son las siglas para el nombre del curso que desarrolló esta política de calendarización, correspondientes a *Cursos Avanzados de Sistemas Operativos*".

Below the note, there is a bulleted list of topics:

- Funcionamiento y sintaxis de uso de structs.
- Propósito y directivas del preprocesador.
- Diferencia entre * y & en el manejo de referencias a memoria (punteros).
- Propósito y modo de uso de APT y dpkg.

A yellow box highlights the instruction "Incluya esta información con sus entregables."

- Funcionamiento y sintaxis de uso de structs
- Propósito y directivas del preprocesador
- Diferencia entre * y & en el manejo de referencias a memoria (punteros).
- Propósito y modo de uso de APT y dpkg.

Laboratorio #5 2021.pdf

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

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

A continuación, cree un backup de, y abra, el archivo kernel_dir/include/linux/sched.h. Modifíquelo de la siguiente manera:

```

#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

#endif /* _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/timer.h>
#include <linux/jiffies.h>

```

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.`

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

```

#define SCHED_BATCH 3

#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:

Laboratorio #5 2021.pdf

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

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

A continuación, cree un backup de, y abra, el archivo kernel_dir/include/linux/sched.h. Modifíquelo de la siguiente manera:

```

#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

#endif /* _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/timer.h>
#include <linux/jiffies.h>

```

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.`

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

```

#define SCHED_BATCH 3

#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:

- ¿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?

The terminal window shows the content of the file `sched.h` from the Linux kernel source code. The PDF viewer window shows a document with the following text:

Laboratorio #5 2021.pdf

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

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

• ¿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 {
...
#endif
 struct prop_local_single dirties;
#ifdef CONFIG_SCHED_CASIO_POLICY
 unsigned int casio_id;
 unsigned long long deadline;
#endif
};`

• ¿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 `sched_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 `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))
#ifdef CONFIG_SCHED_CASIO_POLICY`

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

Laboratorio #5 2021.pdf

shed.h (/usr/include/bits) - gedit

```

/* 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 */

#ifndef __defined_schedparam
#define __defined_schedparam 1
#endif /* __need_schedparam */

/* Data structure to describe a process' schedulability. */
struct __sched_param
{
    int __sched_priority;
    unsigned int casio_id;
    unsigned long long deadline;
};

#ifndef __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 __NPUBITS ((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) / __NPUBITS)

```

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

UVG
UNIVERSIDAD DEL VALLE GUATEMALA

En este mismo archivo busca también la estructura `shed_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 `shed_param` (en realidad, una es para `_sched_param`, incluya estos cambios en ellas también, pero sin encerrárslos en un bloque `#ifdef`). Grabe y cierre `shed.h`.

- ¿Qué información contiene `shed_param`?

- ¿Qué información contiene `shed_param`?

Laboratorio #5 2021.pdf

*shed.c (/home/scheduler/dev/linux-2.6.24-casio/kernel) - gedit

```

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))
#ifdef 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];
};

```

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

UVG
UNIVERSIDAD DEL VALLE GUATEMALA

En este mismo archivo busca también la estructura `shed_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 `shed_param` (en realidad, una es para `_sched_param`, incluya estos cambios en ellas también, pero sin encerrárslos en un bloque `#ifdef`). Grabe y cierre `shed.h`.

- ¿Qué información contiene `shed_param`?

m. Diríjase al archivo `kernel_dir/kernel/sched.c`. La política de calendarización que empleamos 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 modifíquela 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))
#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?
- ¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?

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

```

...
#include "shed_debug.c"
#endif

#ifdef CONFIG_SCHED_CASIO_POLICY
#include "shed_casio.c"
#endif

#ifdef CONFIG_SCHED_CASIO_POLICY
#define __shed_casio_shed_class
#endif

```

- ¿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?

The screenshot shows a dual-monitor setup. The left monitor displays a terminal window titled "Applications Places System" with the command "gedit sched.c (/home/scheduler_dev/linux-6.24-casio/kernel)". The right monitor displays a PDF viewer window titled "Liberatorio #5 2021.pdf". The PDF contains instructions for modifying the Linux scheduler source code.

```

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 (smoothened) load is used
 * during load balance.
 */

```

que nuestra política pertenece a esta clase. Busque la función `rt_policy` y modifíquela 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:

```

...
#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.
...

```

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

UVG
UNIVERSIDAD
DEL VALLE

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

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

```
*sched.c
1      return cpu_rq(cpu)->idle;
}

/**
 * find_process_by_pid - find a process with a matching PID value.
 * @pid: the pid in question.
 */
static struct task_struct *find_process_by_pid(pid_t pid)
{
    return pid ? find_task_by_vpid(pid) : current;
}

/* Actually do priority change: must hold rq lock. */
static void
_setscheduler(struct rq *rq, struct task_struct *p, int policy, int prio)
{
    BUG_ON(p->se.on_rq);

    p->policy = policy;
    switch (p->policy) {
    case SCHED_NORMAL:
    case SCHED_BATCH:
    case SCHED_IDLE:
        p->sched_class = &fair_sched_class;
        break;
    case SCHED_FIFO:
    case SCHED_RR:
        p->sched_class = &rt_sched_class;
        break;
#define CONFIG_SCHED_CASIO_POLICY
    case SCHED_CASIO:
        p->sched_class = &casio_sched_class;
        break;
#endif
    }

    p->rt_priority = prio;
    p->normal_prio = normal_prio(p);
    /* we are holding navi lock already */
}
```

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:

```
...
p->policy = policy;
switch(p->policy){
...
#ifndef CONFIG_SCHED_CASIO_POLICY
    case SCHED_CASIO:
        p->sched_class = &casio_sched_class;
        break;
#endif
}

Y en la función sched_setscheduler realiza las siguientes modificaciones:
...
if (policy < 0)
    policy = oldpolicy = p->policy;
else if ((policy != SCHED_FIFO && policy != SCHED_RR &&
          policy != SCHED_NORMAL && policy != SCHED_BATCH &&
          policy != SCHED_IDLE) ...
#endif
...
/* can't change other user's priorities */
if ((current->euid != p->euid) &&
   (current->euid != p->uid))
    return -EPERM;
}

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

```

Laboratorio #5 2021.pdf

Wed Apr 21, 11:16 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

```
sched.c
1      return -ESRCH;
}

/*
 * if (!capable(CAP_SYS_NICE)) {
    if (rt_policy(policy)) {
        unsigned long rlim_rtprio;

        if (!lock_task_sighand(p, &flags))
            return -ESRCH;
        rlim_rtprio = p->signal->rlim[RLIMIT_RTPRIO].rlim_cur;
        unlock_task_sighand(p, &flags);

        /* can't set/change the rt policy */
        if (policy != p->policy && !rlim_rtprio)
            return -EPERM;

        /* can't increase priority */
        if (param->sched_priority > p->rt_priority &&
            param->sched_priority > rlim_rtprio)
            return -EPERM;

        /*
         * 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;
    }
}

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

retval = security_task_setscheduler(p, policy, param);
```

Ln 4343, Col 14 INS

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

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

UVG
UNIVERSIDAD
DEL VALLE

The screenshot shows a dual-monitor setup. The left monitor displays a terminal window titled 'gedit' containing the file 'sched.c' from the Linux kernel source code. The right monitor displays a PDF viewer titled 'Laboratorio #5 2021.pdf' showing a LaTeX document from the University of Valle de Guatemala.

```

/* 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.
 */

```

The PDF document contains the following text:

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

p. Ahora definiremos las tareas que son calendarizables con nuestra política, y su ready queue. Recuerde que el calendador 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;
};

#ifndef CONFIG_SCHED_CASIO_POLICY
    struct casio_rq casio_rq;
#endif
...
```

- Explique el contenido de la estructura casio_task.

The screenshot shows a dual-monitor setup. The left monitor displays a terminal window titled 'gedit' containing the file 'sched.c' from the Linux kernel source code. The right monitor displays a PDF viewer titled 'Laboratorio #5 2021.pdf' showing a LaTeX document from the University of Valle de Guatemala.

```

unsigned long cpu_load[CPU_LOAD_IDX_MAX];
unsigned char idle_at_tick;
#ifndef CONFIG_NO_HZ
    unsigned char in_nohz_recently;
#endif

/* capture load from *all* tasks on this cpu: */
struct load_weight load;
unsigned long nr_load_updates;
u64 nr_swatches;

struct cfs_rq cfs;
#ifndef CONFIG_FAIR_GROUP_SCHED
    /* list of leaf cfs_rq on this cpu: */
    struct list_head leaf_cfs_rq_list;
#endif
    struct rt_rq rt;
#endif
    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:
 */
unsigned long nr_uninterruptible;

struct task_struct *curr, *idle;
unsigned long next_balance;
struct mm_struct *prev_mm;

u64 clock, prev_clock_raw;
s64 clock_max_delta;

unsigned int clock_warp, clock_overflows;
u64 idle_clock;
unsigned int clock_deep_idle_events;
u64 tick_timestamp;

atomic_t nr_inwait;

```

The PDF document contains the following text:

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

...

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;
};

#ifndef CONFIG_SCHED_CASIO_POLICY
    struct casio_rq casio_rq;
#endif
...
```

• Explique el propósito y contenido de la estructura casio_rq.

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

r. Cuando un proceso cambie su política de calendarización, para usar nuestra política debe ser agregado a la lista. Modifique nuevamente la función sched_setscheduler para que refleje los siguientes cambios:

```

...
    if (unlikely(oldpolicy != -1 && oldpolicy != p->policy)){
        policy = oldpolicy = -1;
        task_rq_unlock(rq);
        spin_unlock_irqrestore(&p->pi_lock, flags);
    }
...
```

- Explique el propósito y contenido de la estructura casio_rq.
 - ¿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).

The screenshot shows a Linux desktop environment with two windows open. The left window is a terminal window titled "ched.c (/home/scheduler_de_linux-2.6.24-casio/kernel) - gedit" containing the source code for the scheduler. The right window is a PDF viewer window titled "Labradorio #5 2021.pdf" displaying a LaTeX document from the University of Valle de Guatemala.

Scheduler Code (ched.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);
```

LaTeX Document (Labradorio #5 2021.pdf):

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

UVG
UNIVERSIDAD
DEL VALLE
DOUGLAS

r. Cuando un proceso cambie su política de calendarización, para usar nuestra política debe ser agregado a la lista. Modifique nuevamente la función `sched_setscheduler` para que refleje los siguientes cambios:

```
...
    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);
...
```

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->nr_running = 0;
    rq->clock = 1;
#endif CONFIG_SCHED_CASIO_POLICY
    init_casio_rq(&rq->casio_rq);
#endif
    init_cfs_rq(&rq->cfs, rq);
...
```

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

Applications Places System **Wed Apr 21, 11:26 PM** **Laboratorio #5 2021.pdf** **Douglas**

sched.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

```

sched.c x
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->n_running = 0;
        rq->clock = 1;
#ifdef CONFIG_SCHED_CASIO_POLICY
        init_casio_rq(&rq->casio_rq);
#endif
        init_cfs_rq(&rq->cfs);
#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);

            init_cfs_rq_pj1 = 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

```

File Edit View Go Help

Previous Next 7 of 18 Fit Page Width

```

#endif
update_rq_clock(rq);
...

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->n_running = 0;
    rq->clock = 1;
#ifdef CONFIG_SCHED_CASIO_POLICY
    init_casio_rq(&rq->casio_rq);
#endif
    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 GUATEMALA

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:

Applications Places System **Wed Apr 21, 11:32 PM** **Laboratorio #5 2021.pdf** **Douglas**

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

```

sched_casio.c x
void init_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=RB_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_c_set(&casio_rq->r_running, 0);
}

```

File Edit View Go Help

Previous Next 8 of 18 Fit Page Width

```

void init_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=RB_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_c_set(&casio_rq->r_running, 0);
}

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:

```

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 tasks:

```

void init_casio_rq(struct casio_rq* casio_rq){
    casio_rq->casio_rb_root=RB_ROOT;
    INIT_LIST_HEAD(&casio_rq->casio_list_head);
    atomic_c_set(&casio_rq->r_running, 0);
}

u. Luego programe las funciones para el manejo de la lista de casio_tasks:
void add_casio_task(struct casio_rq* rq, struct task_struct* p1)
{
    struct list_head* new_task_ptr = NULL;
    struct casio_task* new_task = NULL;
    struct casio_task* casio_task = NULL;
    //char msg;
    if (rq->rq->list_head == NULL) {
        new_task = (struct casio_task*)kzalloc(sizeof(struct casio_task), GFP_KERNEL);
        if (new_task == NULL)
            return;
        casio_task = NULL;
        new_task->ptr = p1;
        new_task->deadline = 0;
        list_for_each_safe(p1, new_task_ptr, &rq->casio_list_head) {
            casio_task = p1;
            if (new_task->task->casio_id < casio_task->task->casio_id)
                if (new_task->task->casio_id < casio_task->task->casio_id)
                    list_add(new_task, &casio_task->list_node, &p1);
                else
                    list_add(new_task, &casio_task->list_node, &new_task_ptr);
            else
                list_add(new_task, &casio_task->list_node, &new_task_ptr);
        }
    } else {
        new_task = (struct casio_task*)kzalloc(sizeof(struct casio_task), GFP_KERNEL);
        if (new_task == NULL)
            return;
        casio_task = NULL;
        new_task->ptr = p1;
        new_task->deadline = 0;
        list_for_each_safe(p1, new_task_ptr, &rq->casio_list_head) {
            casio_task = p1;
            if (new_task->task->casio_id < casio_task->task->casio_id)
                if (new_task->task->casio_id < casio_task->task->casio_id)
                    list_add(new_task, &casio_task->list_node, &p1);
                else
                    list_add(new_task, &casio_task->list_node, &new_task_ptr);
            else
                list_add(new_task, &casio_task->list_node, &new_task_ptr);
        }
    }
}

void rem_casio_task(list_struct casio_rq* rq, struct task_struct* p1)
{
    struct list_head* list_head_ptr = NULL;
    struct casio_task* casio_task = NULL;
    struct casio_task* casio_task2 = NULL;
    //char msg;
    if (rq->rq->list_head == NULL)
        return;
    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
        casio_task = p1;
        if (casio_task->task->casio_id == p1->task->casio_id)
            list_del(&casio_task->list_node);
        else
            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                casio_task2 = p1;
                if (casio_task->task->casio_id == p1->task->casio_id)
                    list_del(&casio_task->list_node);
                else
                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                        casio_task2 = p1;
                        if (casio_task->task->casio_id == p1->task->casio_id)
                            list_del(&casio_task->list_node);
                        else
                            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                casio_task2 = p1;
                                if (casio_task->task->casio_id == p1->task->casio_id)
                                    list_del(&casio_task->list_node);
                                else
                                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                        casio_task2 = p1;
                                        if (casio_task->task->casio_id == p1->task->casio_id)
                                            list_del(&casio_task->list_node);
                                        else
                                            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                casio_task2 = p1;
                                                if (casio_task->task->casio_id == p1->task->casio_id)
                                                    list_del(&casio_task->list_node);
                                                else
                                                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                        casio_task2 = p1;
                                                        if (casio_task->task->casio_id == p1->task->casio_id)
                                                            list_del(&casio_task->list_node);
                                                        else
                                                            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                casio_task2 = p1;
                                                                if (casio_task->task->casio_id == p1->task->casio_id)
                                                                    list_del(&casio_task->list_node);
                                                                else
                                                                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                        casio_task2 = p1;
                                                                        if (casio_task->task->casio_id == p1->task->casio_id)
                                                                            list_del(&casio_task->list_node);
                                                                        else
                                                                            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                                casio_task2 = p1;
                                                                                if (casio_task->task->casio_id == p1->task->casio_id)
                                                                                    list_del(&casio_task->list_node);
                                                                                else
                                                                                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                                        casio_task2 = p1;
                                                                                        if (casio_task->task->casio_id == p1->task->casio_id)
                                                                                            list_del(&casio_task->list_node);
                                                                                        else
                                                                                            list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                                                casio_task2 = p1;
                                                                                                if (casio_task->task->casio_id == p1->task->casio_id)
                                                                                                    list_del(&casio_task->list_node);
                                                                                                else
                                                                                                    list_for_each_safe(p1, list_head_ptr, &rq->casio_list_head) {
                                                                                                        casio_task2 = p1;
................................................................

```

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

UVG
UNIVERSIDAD DEL VALLE GUATEMALA

Laboratorio #5 2021.pdf

```

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

u. Luego programas 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-
>task->casio_id){
                        list_add(&new->casio_list_node, &casio_task-
>casio_list_node, ptr);
                        return;
                    }
                }
            }
            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");
    }
}
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){
        struct list_head* list_head;
        struct list_head* pte = NULL;
        struct list_head* next_head = NULL;
        struct casio_task* casio_task = NULL;
        if (rq && p){
            list_for_each_safe(ptr, next, &rq->casio_list_head){
                casio_task = list_entry(ptr, struct casio_task,
casio_list_node);
                if (casio_task)
                    if (casio_task->task->casio_id == p->task->casio_id)
                        list_del_init(&ptr);
                    else
                        printk(KERN_ALERT "rem_casio_task_list: kfree\n");
            }
        }
    }
}

```

Ln 71, Col 1 INS

Laboratorio #5 2021.pdf

```

v. Ahora programas 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 = &rq->casio_rb_root.rb_node;
    while(*node != NULL){
        parent = *node;
        entry = rb_entry(parent, struct casio_task, casio_rb_node);
        if (entry){
            if (p->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 99, Col 1 INS

```

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

void remove_casio_task_rb_tree(struct casio_rq* rq, struct casio_task* p){
    rb_erase(&p->casio_rb_node), (&q->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?

```

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);
            //log
        } 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){
            //log
            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 = {
Ln 126, Col 1           INS

```

x. Ahora definiremos las funciones que conforman nuestra clase de calendarización. Asegúrese de incluir este código antes de la declaración de `casio_sched_class`:

```

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);
            //log
        } 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){
            //log
            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");
            }
        }
    }
}

Ln 126, Col 1           INS

```

• 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?

- 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?

The screenshot shows a Linux desktop environment with two windows open. On the left is a terminal window titled 'gedit' containing the file 'sched_casio.c'. The code in this file is related to task scheduling, specifically for the 'casio' class. It includes functions like 'enqueue_task_casio', 'dequeue_task_casio', 'check_preempt_curr_casio', and 'pick_next_task_casio'. The code uses structures like 'task_struct' and 'rb_node' for managing tasks. On the right is a PDF viewer window titled 'Laboratorio #5 2021.pdf'. The PDF contains information about the University of Valle de Guatemala (UVG), the course 'Sistemas Operativos', and the authors 'Erick Pineda; Tomás Gálvez P.'. It also includes a section on 'Semestre I, 2021' and some code snippets from the kernel source.

```

*sched_casio.c (*sched_casio.c) +-----+-----+
File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
*sched_casio.c
int(t->task->state == TASK_DEAD || t->task->state == EXIT_DEAD
    || t->task->state==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,
};

Ln 166, Col 1      INS

```

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

UVG
UNIVERSIDAD DEL VALLE
GUATEMALA

```

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");
            }
        }
    }
}

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

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,
                                         struct rq* next_rq);

```

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

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 //log1, //log2 y //log3 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 //log1, //log2 y //log3 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.