

Universidad del Valle de Guatemala
Sistemas Operativos
ANDRES QUINTO
Carné 18288

Laboratorio #5

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

The image shows two windows side-by-side. The left window is a text editor displaying a portion of a Kconfig file. The right window is a web browser displaying a document with instructions for modifying the Linux kernel scheduler.

Left Window (Kconfig Content):

```

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"

```

Ln 1612, Col 1 INS

Right Window (Document Content):

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

```

Extraiga el contenido de este paquete usando el comando `tar -xvf` seguido del nombre del archivo descargado. Cámbiele el nombre a la carpeta que se produce con la extracción a `linux-2.6.24-casio`. En adelante nos referiremos a esta carpeta donde está el kernel extraído como `kernel_dir`.

h. Para agregar una política de calendarización a Linux primero será necesario registrala 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

```

Nota: CASIO son las siglas para el nombre del curso que desarrolló esta política de calendarización, correspondientes a *Concursos Avanzados de Sistemas Operativos*.

Aunque ya ha visto bastante código en C y probablemente tenga experiencia previa con el lenguaje, es importante conocer algunas de sus características para que el código que se prueba en este laboratorio no sea copiado ciegamente, sino entendido en el proceso. Por ello, investigue y resuma:

- Funcionamiento y sintaxis de uso de `struct`.
- 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.

Incluya esta información con sus entregables.

Universidad del Valle de Guatemala

- 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 dual-pane editor window. The left pane displays the source code for the file `sched.h`. The right pane displays a LaTeX document with sections and notes.

```

/* clear the TID in the child */
#define CLONE_CHILD_CLEARTID 0x00200000
#define CLONE_DETACHED 0x00400000 /* Unused, ignored */
#define CLONE_UNTRACED 0x00800000 /* 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 utnsname group? */
#define CLONE_NEWIPC 0x08000000 /* New ipcs */
#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

The right pane contains the following LaTeX document:

Universidad del Valle de Guatemala
Sistemas Operativos
UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

i. 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

#ifndef __KERNEL__
...
```

Note que lo que este extracto de código le indica 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
#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 image shows two windows side-by-side. The left window is a text editor displaying the contents of the file `sched.h`. It includes the GNU Lesser General Public License notice at the top, followed by several #define statements for scheduling algorithms (SCHED_OTHER, SCHED_FIFO, SCHED_RR, SCHED_BATCH, SCHED_IDLE, SCHED_CASIO) and clone flags (CLONE_VM, CLONE_FS, CLONE_FILES, CLONE_SIGHAND, CLONE_PTRACE, CLONE_CHILD_CLEARTID, CLONE_PARENT, CLONE_THREAD, CLONE_NEWNS, CLONE_SYSVSEM, CLONE_SETTLS). The right window is also a text editor showing a modified version of `sched.h`. It adds new policy definitions: `SCHED_NORMAL`, `SCHED_FIFO`, `SCHED_RR`, `SCHED_BATCH`, and `SCHED_CASIO`. It also includes a section for the kernel with `SCHED_IDLE` and `SCHED_CASIO`. A text box on the right provides instructions for modifying the kernel's task_struct definition.

```

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_UNDO semantics. */
#define CLONE_SETTLS 0x00080000 /* Set TLS info. */

```

Above the code, there is a note: "A continuación, cree un backup de, y abra, el archivo kernel_dir/include/linux/sched.h. Modifíquelo de la siguiente manera:" followed by a series of #define statements:

```

#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

#define CONFIG_SCHED_CASIO_POLICY
#define SCHED_CASIO    6
#endif

#ifndef KERNEL
...
```

Below this, there is a note: "Note que lo que este extracto de código te indica con los colores es que agregue la parte de #ifdef luego de #define SCHED_IDLE 5." followed by another set of code modifications:

```

j. En el archivo /usr/include/bits/sched.h realice la siguiente modificación:
...
#define SCHED_BATCH 3
#endif

#define SCHED_CASIO 6
```

Finally, there is a list of questions:

- ¿Cuáles 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?
- La definición incluída es nuestra política de calendarización SCHED_CASIO, mientras que las existentes son las que trae el kernel por defecto. Entre las incluídas están Round Robin y FIFO que vimos en clase.

The image shows two windows side-by-side. The left window is a code editor displaying the file `*sched.h`. It contains C code related to scheduling, including definitions for structures like `compat_robust_list_head`, `list_head`, and `task_struct`, and policies like `SCHED_RT` and `SCHED_NORMAL`. The right window is a LaTeX document titled "Sistemas Operativos" from the "Universidad del Valle de Guatemala". It includes sections k and l, which discuss the `task_struct` and its analogies in Windows, and the `sched_param` structure.

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Cut Copy Paste Find Replace
* *sched.h *
#ifndef CONFIG_COMPAT
    struct compat_robust_list_head __user *compat_robust_list;
#endif
    struct list_head pi_state_list;
    struct futex_pi_state *pi_state_cache;
#endif
    atomic_t fs_excl; /* holding fs exclusive resources */
    struct rcu_head rcu;

/*
 * cache last used pipe for splice
 */
    struct pipe_inode_info *splice_pipe;
#endif
CONFIG_TASK_DELAY_ACCT
    struct task_delay_info *delays;
#endif
CONFIG_FAULT_INJECTION
    int make_it_fail;
#endif
    struct prop_local_single dirties;
#endif
CONFIG_SCHED_CASIO_POLICY
    unsigned int casio_id;
    unsigned long long deadline;
#endif
};

/*
 * Priority of a process goes from 0..MAX_PRIO-1, valid RT
 * priority is 0..MAX_RT_PRIO-1, and SCHED_NORMAL..SCHED_BATCH
 * tasks are in the range MAX_RT_PRIO..MAX_PRIO-1. Priority
 * values are inverted: lower p->prio value means higher priority.
 *
 * The MAX_USER_RT_PRIO value allows the actual maximum
 * RT priority to be separate from the value exported to
 * user-space. This allows kernel threads to set their
 * priority to a value higher than any user task. Note:
 * MAX_RT_PRIO must not be smaller than MAX_USER_RT_PRIO.
 */
#define MAX_USER_RT_PRIO      100
Ln 1188, Col 7      INS

```

File Edit View Go Help
Previous Next 3 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

k. ¿Cuáles 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 linea 921). Se agregarán a ella los parámetros con los que se relacionará una tarea general con una tarea calendarizada por nuestra nueva política. Para ello su modificación al archivo debe ser la siguiente:
`struct task_struct {`
 `...`
 `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 busque también la estructura `sched_param` (línea 47) y agregúele 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?
 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.

sched.h

```

/* 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;
};

#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 __NCPUBITS (8 * sizeof (__cpu_mask))

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

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

```

Ln 97, Col 10 INS

kernel/include/linux/sched.h

3 of 18 Fit Page Width

• ¿Cuál es el propósito de los archivos sched.h modificados?
 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 {
...
};

#ifndef __SCHED_CASIO_POLICY
struct __sched_local_single_dirties;
#endif

```

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

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

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

sched.c

```

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))
#endif
        || 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];
};

```

Ln 148, Col 41 INS

kernel/include/linux/sched.h

4 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

l. En este mismo archivo busca también la estructura sched_param (línea 47) y agrégale 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). Incluye estos cambios en ellas también, pero sin encerrarlos en un bloque #ifdef. Graba y cierra sched.h.
 m. ¿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 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))
#endif
    #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

#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 dual-pane code editor. The left pane displays the file `sched.c` with its source code. The right pane shows a modified version of the `rt_policy` function, specifically the part related to the `SCHED_EDF` policy. The modification includes the `unlikely` macro and a note explaining its purpose. The code editor has a toolbar at the top with standard file operations like New, Open, Save, Print, Undo, Redo, Cut, Copy, Paste, Find, and Replace. The status bar at the bottom indicates "Ln 891, Col 49" and "INS". The footer of the window includes the text "Universidad del Valle de Guatemala" and the UVG logo.

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
File Edit View Go Help
Previous Next 4 of 18 Fit Page Width
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 cpuaacct_charge(struct task_struct *tsk, u64 cputime);
#else
static inline void cpucacct_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.
 */
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;
}

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.

File Edit View Search Tools Documents Help

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:
        p->sched_class = &rt_sched_class;
        break;
    #ifdef 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 p->rq_lock already */
}

```

Ln 4253, Col 31 INS

File Edit View Go Help

Previous Next 5 of 18 Fit Page Width

Semestre,

- 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 calendario con llamadas a sistema durante su ejecución. En la función __setscheduler realice la siguiente modificación:

```

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

Y en la función sched_setscheduler realice 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
    /* */
#ifndef CONFIG_SCHED_CASIO_POLICY
    && policy != SCHED_CASIO
#endif
    )
    return -EINVAL;
...
    /* 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
...

```

File Edit View Search Tools Documents Help

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

*sched.c

```

/*
 * 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, don't 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;
 * }
 *
#endif 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

File Edit View Go Help

Previous Next 5 of 18 Fit Page Width

```

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

Y en la función sched_setscheduler realice 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
    /* */
#ifndef CONFIG_SCHED_CASIO_POLICY
    && policy != SCHED_CASIO
#endif
    )
    return -EINVAL;
...
    /* 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
...

```

Universidad del Valle de Guatemala

UVG
UNIVERSIDAD
DEL VALLE DE GUATEMALA

The image shows two computer monitors side-by-side. The left monitor displays a code editor with the file 'sched.c' open. The code is part of the Linux kernel's scheduler. It defines several structures, including 'rt_rq' and 'casio_rq'. The 'casio_rq' structure includes fields like 'rb_root', 'casio_list_head', and 'atomic_t nr_running'. The right monitor displays a LaTeX document with the title 'Universidad del Valle de Guatemala' and 'Sistemas Operativos'. It contains a question (p) asking about modifying the 'rt_rq' structure to use a red-black tree for scheduling tasks. The LaTeX code shown includes parts of the 'rt_rq' and 'casio_rq' structures, along with comments explaining the modifications.

```

* 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 */

#endif
};

/* 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.
 */

```

Ln 277, Col 41 INS

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

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;
#ifndef 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 displays two windows side-by-side. The left window is a code editor showing the content of the file `sched.c`. The right window is a document viewer showing a question from a document.

Left Window (Code Editor):

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
 sched.c
unsigned long cpu_load[CPU_LOAD_IDX_MAX];
unsigned char idle_at_tick;
#endif
#define 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_switches;

struct cfs_rq cfs;
#endif
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
CONFIG_SCHED_CASIO_POLICY
struct casio_rq casio_rq;
#endif

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

```

Ln 325, Col 7 INS

Right Window (Document Viewer):

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 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).

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

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`.
Esta nos ayuda a mantener una linked list con head `casio_list` para asignar los tasks 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.

File Edit View Search Tools Documents Help

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

sched.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;
    }

#ifndef 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);

    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;
#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.

Ln 4390, Col 7 INS
```

File Edit View Go Help

Previous Next 7 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

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;
    }

#ifdef 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;
#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.
```

File Edit View Search Tools Documents Help

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

sched.c

```
cfs_rq->tasks_timeline = RB_ROOT;
#ifndef CONFIG_FAIR_GROUP_SCHED
    cfs_rq->rq = rq;
#endif
    cfs_rq->min_vruntime = (u64)(- (1LL << 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->nr_running = 0;
        rq->clock = 1;
    }

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

        init_cfs_rq_pil = 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_pil = 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->nr_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

UVG
UNIVERSIDAD
DEL VALLE
DE 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:

File Edit View Tools Documents Help

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

sched_casio.c

```
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->nr_running, 0);
}
```

Ln 6, Col 1 INS

File Edit View Go Help

Previous Next 8 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

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:

```
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->nr_running, 0);
}

u. Luego programe las funciones para el manejo de la lista de casio_tasks:
```

```
#include "casio_task.h"
#include "casio_rq.h"
#include "casio_rb.h"
#include "list.h"
#include "spinlock.h"
#include "atomic.h"
#include "kern.h"
#include "syscalls.h"
#include "casio.h"

struct casio_task_struct {
    struct list_head list;
    int casio_id;
    int deadline;
    void (*function)(void* arg);
    void* arg;
};

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);
                        //log
                    } 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) {
        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->casio_id) {
                    list_del(ptr);
                    //log
                    kfree(casio_task);
                }
            }
        }
    }
}
```

File Edit View Tools Documents Help

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

sched_casio.c

```
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);
                        //log
                    } 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) {
        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->casio_id) {
                    list_del(ptr);
                    //log
                    kfree(casio_task);
                }
            }
        }
    }
}
```

Ln 71, Col 1 INS

File Edit View Go Help

Previous Next 8 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD DEL VALLE GUATEMALA

u. Luego programe las funciones para el manejo de la lista de casio_tasks:

```
#include "casio_task.h"
#include "casio_rq.h"
#include "casio_rb.h"
#include "list.h"
#include "spinlock.h"
#include "atomic.h"
#include "kern.h"
#include "syscalls.h"
#include "casio.h"

struct casio_task_struct {
    struct list_head list;
    int casio_id;
    int deadline;
    void (*function)(void* arg);
    void* arg;
};

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);
                        //log
                    } 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) {
        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->casio_id) {
                    list_del(ptr);
                    //log
                    kfree(casio_task);
                }
            }
        }
    }
}
```

File Edit View Tools Documents Help

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

```
*sched_casio.c
}

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->b_left;
            } else {
                node = &parent->b_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->b_left != NULL){
        node = node->b_left;
    }
    p = rb_entry(node, struct casio_task, casio_rb_node);
    return p;
}
}
Ln 109, Col 1 INS
```

File Edit View Go Help

Previous Next 9 of 18 Fit Page Width

```
1. CASIO_TASKS
    if (casio_task->task->casio_id == p->casio_id)
        return casio_task
    }

    }

    return NULL;
}

v. Ahora programé 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->b_left;
            } else {
                node = &parent->b_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->b_left != NULL){
        node = node->b_left;
    }
    p = rb_entry(node, struct casio_task, casio_rb_node);
    return p;
}
}
Ln 109, Col 1 INS
```

File Edit View Tools Documents Help

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

```
*sched_casio.c
}

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->b_left != NULL){
        node = node->b_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,
};

#endif 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
```

File Edit View Go Help

Previous Next 10 of 18 Fit Page Width

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
GUATEMALA

w. Las funciones que recién definimos son como el *backend* de nuestra política de calendarización. Recomendamos que en la clase `&casio_sched_class` agregamos una condicional para que se tomara esta clase, pero nótese que no hablamos de una clase del paradigma de orientación a objetos sino de una clase de calendarización. Esta clase es en realidad la declaración de una constante de tipo `struct sched_class`, que requiere la definición de ciertos valores para funcionar como una calendarización en el sistema (similar a una interfaz en Java). Incluya el siguiente código en `sched_casio.c`:

```
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,
};

#ifdef 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,
};

• ¿Qué indica el campo .next de esta estructura?
```

- ¿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.

The left pane shows the `*sched_casio.c` file with C code for enqueueing and dequeuing tasks from a red-black tree. The right pane shows a LaTeX document from the UVG Sistemas Operativos course, section 11, page 18, titled "x. Ahora definiremos las funciones que conforman nuestra clase de calendarización. Asegúrese de incluir este código ANTES de la declaración de `casio_sched_class`:". It contains the same C code for enqueueing and dequeuing tasks, followed by a bullet point explaining the purpose of the functions.

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
File Edit View Go Help
Previous Next 11 of 18 Fit Page Width
Universidad del Valle de Guatemala
Sistemas Operativos
UVG
UNIVERSIDAD DEL VALLE DE GUATEMALA
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);
            //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 = {

```

Ln 126, Col 1 INS

UVG
UNIVERSIDAD DEL VALLE DE GUATEMALA

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);
            //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");
        }
    }
}

```

• 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?

- 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 screenshot shows two panes of a code editor. The left pane displays the `sched_casio.c` file with its source code. The right pane shows a printed or displayed version of the same code, which includes a header from the University of Valle de Guatemala (UVG) and some annotations.

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
*sched_casio.c
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          = &t_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,
};

#endif CONFIG_SMP
.load balance      = load_balance_casio.
Ln 166, Col 1      INS

```

Universidad del Valle de Guatemala
Sistemas Operativos
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)
{
}

#ifndef CONFIG_SMP
static unsigned long load_balance_casio(struct rq* this_rq, int this_cpu,

```

- ¿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 asignada 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.

File Edit View Tools Documents Help

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

*sched_casio.c

```

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

File Edit View Go Help

Previous Next 12 of 18 Fit Page Width

* ¿Cuándo preempesta 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)
{
}

#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

```

Universidad del Valle de Guatemala
Sistemas Operativos

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)

Ln 189, Col 2 INS

File Edit View Tools Documents Help

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

*sched.h

```

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

#endif /* __KERNEL__ */

#ifndef 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

File Edit View Go Help

Previous Next 13 of 18 Fit Page Width

```

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 el kernel_dir/include/linux/sched.h aplicar la siguiente modificación:
...
#endif /* __KERNEL__ */

#ifndef 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

```

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

File Edit View Search Tools Documents Help

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

View Go Help

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

```
*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, 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");
        }
    }
}

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 {
        + = earliest_deadline_casio_task_rq(rq->casio_rq);
    }
}

Ln 161, Col 33 INS
```

aa. 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_task->task->casio_id, casio_task->task->pid, casio_task->absolute_deadline;y 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.
- Finalmente 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.

bb. Un evento que debemos registrar pero que no controlamos desde sched_casio.c es el cambio de contexto que involucra una o dos casio tasks. 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);

#ifndef 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){
        //log1
        if (prev->policy == SCHED_CASIO){
            //log2
        } else {
            //log3
        }
    }
    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:
```

File Edit View Search Tools Documents Help

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

View Go Help

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD
DEL VALLE
DE GUATEMALA

```
*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);

#ifndef 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){
        sprintf(msg, CASIO_MSG_SIZE, "prev->(%d:%d), next->(%d:%d)", prev->casio_id, prev->pid, next->casio_id, next->pid);
    } else {
        sprintf(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)) {
    rq->nrc_switches++;
}

Ln 3685, Col 1 INS
```

bb. Un evento que debemos registrar pero que no controlamos desde sched_casio.c es el cambio de contexto que involucra una o dos casio tasks. 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);

#ifndef 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){
        //log1
        if (prev->policy == SCHED_CASIO){
            //log2
        } else {
            //log3
        }
    }
    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:
```

cc. Finalmente modificaremos kernel_dir/fs/proc/proc_misc.c para que nuestra bitácora 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).

proc_misc.c

```

File Edit View Search Tools Documents Help
New Open Save Print... Undo Redo Cut Copy Paste Find Replace
View Go Help
17 of 18 Fit Page Width
proc_misc.c
    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_vmcore_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

```

Universidad del Valle de Guatemala
Sistemas Operativos

UVG
UNIVERSIDAD DEL VALLE DE GUATEMALA

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 compilar. 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 configurado 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-2.6.24-casio
```

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

File Edit View Terminal Tabs 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 (i586) (CRYPTO_AES_586) [M/n/y/?] m
CAST5 (CAST-128) cipher algorithm (CRYPTO_CAST5) [M/n/y/?] m
CAST6 (CAST-256) cipher algorithm (CRYPTO_CAST6) [M/n/y/?] m
TEA, XTEA and XETA cipher algorithms (CRYPTO_TEAB) [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
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#

```

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 compilar. 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 configurado 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-2.6.24-casio
```

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

File	Edit	View	Go	Help
Previous	Next	17	of 18	Fit Page Width <input type="button" value="▼"/>
<p><code>sudo cp /boot/config-2.6.24-26-generic .config</code></p> <p>Note el espacio antes de <code>.config</code>. Ahora copie todo el contenido de <code>scheduler.dev/linux-2.6.24-casio</code> a <code>scheduler</code> (use la opción <code>-a</code> del comando <code>cp</code>). Se recomienda crear una <i>snapshot</i> (al menos) en este punto. Diríjase a <code>scheduler/linux-2.6.24-casio</code> y ejecute lo siguiente:</p> <pre><code>sudo make oldconfig</code></pre> <p>Este proceso de compilación toma un archivo de configuración existente y crea uno nuevo, pidiendo <i>input</i> al usuario sobre las características nuevas o desconocidas que tenga el <i>kernel</i> a compilarse. Para cada pregunta que se le realice habrá un <i>valor</i> entre corchetes y, en caso de ser una pregunta con respuesta “<i>sí</i>” o “<i>no</i>”, se señalará con una letra mayúscula la opción por defecto. Asegúrese de que <code>CASIO Scheduler</code> sea configurada con ‘<i>y</i>’ y todas las demás opciones con su valor por defecto. Al terminar, compile el <i>kernel</i> con el siguiente comando:</p> <pre><code>sudo make-kpkg --initrd kernel_image 2>> .errors</code></pre> <p>Cualquier error detectado durante la compilación se almacenará en el archivo <code>.errors</code>, en el directorio <code>scheduler</code>. Una vez termine la compilación, instale el <i>kernel</i> con el siguiente comando:</p> <pre><code>sudo dpkg -i linux-image-...deb</code></pre> <p>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.</p>				

- 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.

Referencias bibliográficas: