Universidad del Valle Guatemala
Sistemas Operativos
Tomás Gálvez

Ivette Cardona
Carné 16020
Fecha: 25/02/2019

Laboratorio 4

Conocimientos necesarios

- **Funcionamiento y sintaxis de uso de structs**
  En C, se tiene el tipo de dato *structure* el cual permite agrupar datos de diferentes tipos. Para definir una *structure* es necesario hacer uso de la palabra clave ***struct***.
  La sintaxis es la siguiente:

```
struct mystruct
{
    int int_member;
    double double_member;
    char string_member[25];
} variable;
```

  En donde variable hace referencia a una o más variables que conforman la estructura.

  Un ejemplo de su uso es el siguiente:

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
```

- **Propósito y directivas del preprocesador**
  El cpp es el preprocesador para C. Es el primer programa invocado por el compilador y procesa directivas, las cuales son:

  ➢ #if
  ➢ #else

- ➢ #elif
- ➢ #define
- ➢ #undef
- ➢ #warning
- ➢ #error
- ➢ #line
- ➢ #region
- ➢ #endregion
- ➢ #pragma
- ➢ #pragma warning
- ➢ #pragma checksum

Estas directivas se encargan de mirar la información del documento antes de completar su compilación, verificando que no existan errores.

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

  En C, al contrario que en otros lenguajes de programación, se puede obtener directamente la dirección de memoria de cualquier variable. Esto se logra al utilizar el operador unitario "**&**".

```
char a;         /* Variable 'a' de tipo char */

printf("la direccion de memoria de 'a' es: %p \n", &a);
```

  En cambio, el operador unitario "*****" se utiliza para obtener lo apuntado por un puntero.

```
char a;         /* Variable 'a' de tipo char */
char *pchar;    /* Puntero a char 'pchar' */

pchar = &a;     /* 'pchar' <- @ de 'a' */

printf("la direccion de memoria de 'a' es: %p \n", &a);
printf("y su contenido es : %c \n", *pchar);
```

- **Propósito y modo de uso de APT y dpkg**

  APT (Advanced Packaging Tool) está basando en una biblioteca que contiene la aplicación central y **apt-get** fue la primera interfaz desarrollada dentro del proyecto. Con APT se puede agregar o eliminar paquetes del sistema. Para poder utilizarlo es necesario actualizar la lista de paquetes; esto puede realizarse simplemente con **apt update**.

  Ahora bien, **dpkg** es un gestor de paquetes Debian de medio nivel. Es utilizado para instalar, construir, borrar y gestionar paquetes de Debian GNU/Linux. Se llama con parámetros desde línea de órdenes, especificando una acción y cero o más opciones. La acción dice a dpkg qué hacer y las opciones controlan de alguna manera su comportamiento. También se puede usar dkpg como interfaz a dpkg−deb. Si aquél se encuentra alguna de las opciones de éste, se limita a llamarlo con esas mismas opciones. Las opciones de dpkg−deb pueden ser:

  −b, −−build,
  −c, −−contents,
  −I, −−info,
  −f, −−field,
  −e, −−control,
  −x, −−extract,

−X, −−vextract, and
−−fsys−tarfile.

Modificando sched.h

- **¿Cuál es el propósito de los archivos sched.h modificados?**
  En el sched.h se definen las diferentes clases de procesos. Por ejemplo:

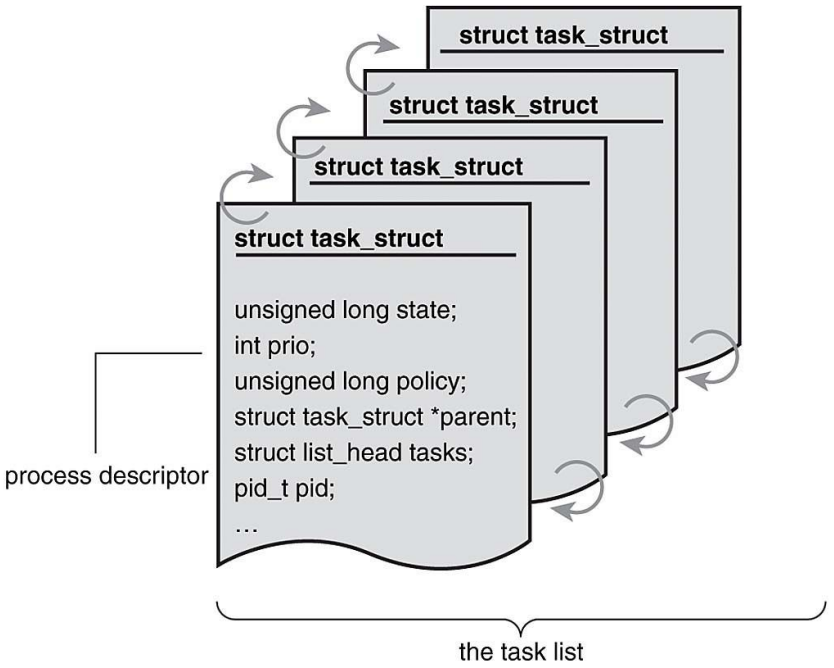
```
/* Scheduling Policies
*/
#define SCHED_OTHER  0
#define SCHED_FIFO   1
#define SCHED_RR     2
```

  Al modificarlo se pretende definir una macro para identificar la política de calendarización(scheduling). Es por eso que se modifican ambos archivos sched.h.

- **¿Cuál es el propósito de la definición incluida y las definiciones existentes en el archivo?**
  SCHED_NORMAL (SCHED_OTHER): son las tareas normales realizadas por el usuario (por defecto).
  SCHED_FIFO: las tareas ejecutadas nunca se anularan. Salen del CPU solo para esperar eventos del kernel de sincronización, si se ha solicitado una suspensión explícita o una reprogramación desde el espacio del usuario.
  SCHED_RR: estas tareas son ejecutadas en tiempo real, pero dejarán la CPU si hay otra tarea en tiempo real en la cola de ejecución. Por lo tanto, la potencia del CPU se distribuirá entre todas las tareas de SCHED_RR. Si al menos una tarea de tiempo real se está ejecutando, ninguna otra tarea de SCHED_NORMAL podrá ejecutarse.
  SCHED_BATCH: es una variante del SCHED_IDLE, en donde se podrá hacer uso del CPU solo si los procesos en SCHED_NORMAL no necesitan usarlo.
  SCHED_IDLE: utilizado para ejecutar trabajos de background de muy baja prioridad.
  SCHED_CASIO: nueva política añadida para admitir tareas en tiempo real programadas de acuerdo con el algoritmo de programación de EDF.

Modificando task_struct

- **¿Qué es una task en Linux?**
  Todo en GNU/Linux es una ***task***. Todo lo que se intercambia en el CPU es una *task_entity*. Una task es programada a través del comando de kernel: *schedule*() y *pick_up_next_task*().
  Por tal razón, en algunos casos cuando se dice *task*, se refiere a un proceso (programa en ejecución).
- **¿Cuál es el propósito de task_struct y cuál es su análogo en Windows?**
  Un process descriptor del tipo task_struct, es un elemento en la lista de tasks. Contiene toda la información sobre un proceso en específico.



Los procesos en Linux son implementados en el kernel como instancias de task_struct. El campo mm en el task_struct apunta al “memory descriptor”, mm_struct, el cual es un resumen ejecutivo de la memoria de un programa. En Windows, el bloque **EPROCESS** es una mezcla entre el task_struct y mm_struct.

Modificando sched_param

- **¿Qué información contiene sched_param?**
  Esta estructura describe los parámetros de calendarización(scheduling).

Modificando rt_policy

- **¿Para qué sirve la función rt_policy y para qué sirve la llamada unlikely en ella?**
  Determina qué política de calendarizador se dará a lugar. La llamada unlikely es una sugerencia para que el compilador emita instrucciones que favorezcan el lado improbable del condicional.
- **¿Qué tipo de tareas calendariza la política EDF, en vista del método modificado?**
  Con EDF siempre que ocurra un evento de programación (tarea finalizada, nueva tarea liberada, etc.) se buscará en la cola el proceso más cercano a su fecha límite. Este proceso es el siguiente en ser programado para su ejecución.
  El tipo de tareas que calendariza son las que maneja CASIO per se (casio_task).

Implementando sched_casio.c

- **Describa la precedencia de prioridades para las políticas EDF, RT y CFS, de acuerdo con los cambios realizados hasta ahora.**
  EDF le da prioridad a la task con la deadline que llega primero; no requiere de procesos periódicos, solamente el deadline de los procesos. En cambio, CFS trata de dividir el procesamiento de forma justa entre los procesos dados, para esto utiliza el árbol red-black y RT los ordena por tiempo real.

Estructura casio_task

- **Explique el contenido de la estructura casio_task.**

```
struct casio_task{
        struct rb_node casio_rb_node;
        unsigned long long absolute_deadline;
        struct list_head casio_list_node;
        struct task_struct* task;
};
```

Contiene una estructura para los nodos del árbol red-black, la etiqueta de los nodos que es el absolute_deadline y otras dos estructuras correspondientes a los nodos recorridos y las tareas que maneja.

Estructura casio_rq

- **Explique el propósito de la estructura casio_rq.**

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

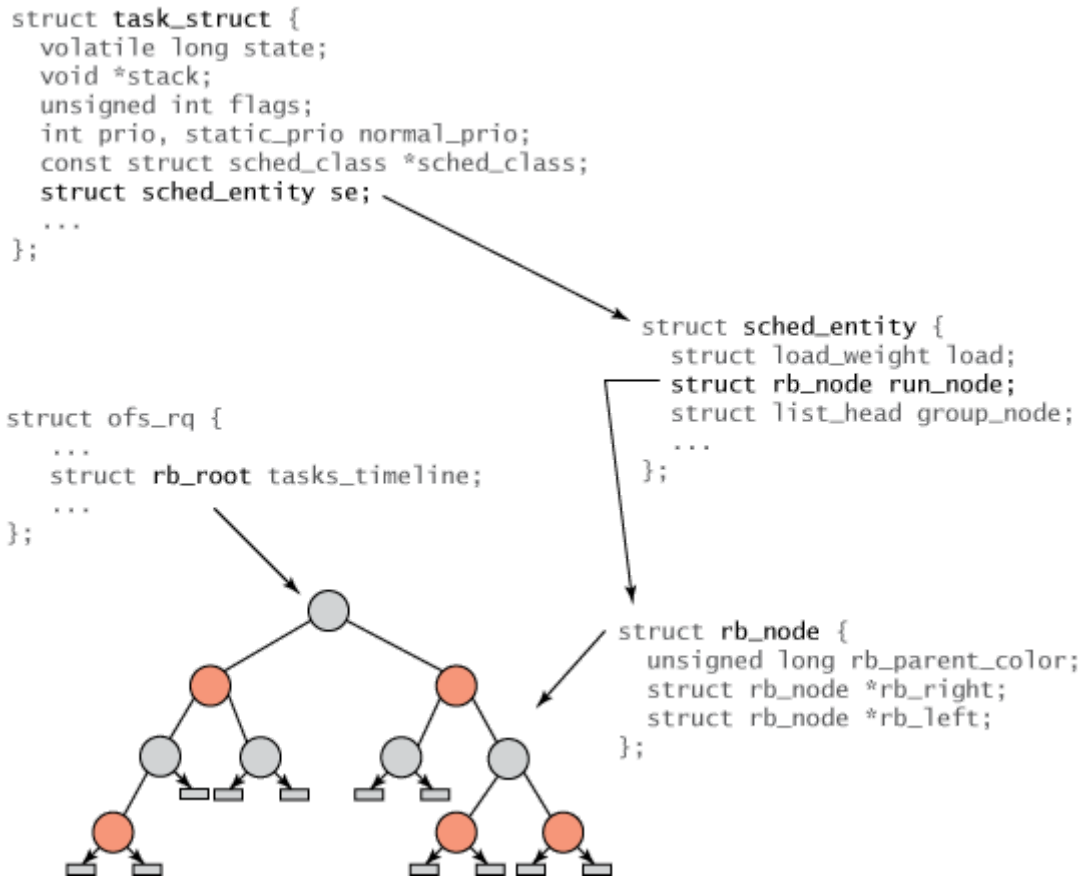
El propósito de esta es ordenar los procesos por prioridad, de manera que se tiene una lista con procesos y su prioridad, la cual será recorrida y ordenada constantemente.

Programando sched_casio.c

- **¿Qué indica el campo .next de esta estructura?**
  Apunta a &rt_sched_class, lo que indica a la dirección de memoria de esta variable. Se utiliza para organizar los módulos del planificador por prioridad en una lista vinculada y el núcleo del calendarizador, comenzando por el módulo del calendarizador de mayor prioridad. Buscará una tarea ejecutable de cada módulo en un orden decreciente por prioridad.

- **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?**
  El enqueue_task_casio es llamado cada vez que que una CASIO task entra en un estado de ejecución. Esta función invoca find_casio, luego lleva el puntero a la task strcut casio en la lista vinculada. Después, actualiza la fecha límite absoluta e inserta casio_task en el árbol red-black.

  Las tasks se guardan en el árbol red-black debido que se deben ordenar dependiendo del tiempo. Esto debido a la prioridad dada por CFS.



Y se guarda en una lista encadenada con el fin de tener la capacidad de que se pueda iterar sobre ellas. Por ejemplo, en el caso de que se desee habilitar o deshabilitar el procesador dada a la tarea.

- **¿Cuándo preemptea una casio_task a la task actualmente en ejecución?**
  Esto se da al momento de llamar a la función pick_next_casio.
  Esta función selecciona la tarea que ejecutará el procesador actual. La función es invocada por el núcleo del planificador siempre que la tarea que se está ejecutando actualmente se anule. Si no hay una casio_task que necesite ser ejecutada, la función devuelve NULL y el planificador intenta encontrar otra tarea en la siguiente calendarización de prioridad menor.

Post Casio

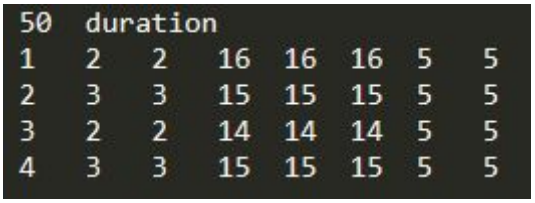
Nota: El archivo log de eventos se encuentra en el vector struct casio_event casio_event, el cual es manejado como un queue circular.

- **Diferencias entre pre_casio.txt y post_casio.txt**

pre_casio	post_casio
• pid = 1 • deadline = 15000000000 • Task(2)	• pid = 2 • deadline = 16000000000 • Task(1)

En general, las prioridades en las ejecuciones de las tareas es diferente en el pre y post.

- **¿Qué información contiene el archivo system que se especifica como argumento en la ejecución de casio_system?**



  Contiene la duración de las tasks.

- **Investigue el concepto de aislamiento temporal en relación a procesos. Explique cómo el calendarizador SCHED_DEADLINE, introducido en la versión 3.14 del kernel de Linux, añade al algoritmo EDF para lograr aislamiento temporal.**

  El aislamiento de procesos es un mecanismo para ejecutar programas con seguridad y de manera separada. A menudo se utiliza para ejecutar código nuevo, o software de dudosa procedencia. El aislamiento permite controlar de cerca recursos proporcionados a los programas “cliente” a ejecutarse, tales como espacio temporal de disco y memoria.

  SCHED_DEADLINE está basado en el algoritmo Earliest Deadline First (EDF), el cual soporta reservar recursos. Con SCHED_DEADLINE las tareas declaran independientemente sus requisitos de tiempo, en términos de un tiempo de ejecución que se necesite por tarea, y el kernel los acepta en el calendarizador después de una prueba de calendarización. Ahora, en caso de que una tarea intente ejecutarse por más tiempo que su presupuesto asignado, el kernel suspenderá esa tarea y deberá aplazar su ejecución hasta su próximo período de activación. Esta propiedad de conservación no programable del programador le permite proporcionar **aislamiento temporal** entre las tareas.

Anexos

Evidencias de compilación

```
CC      kernel/time/ntp.o
CC      kernel/time/clocksource.o
CC      kernel/time/jiffies.o
CC      kernel/time/timer_list.o
CC      kernel/time/clockevents.o
CC      kernel/time/tick-common.o
CC      kernel/time/tick-broadcast.o
CC      kernel/time/tick-oneshot.o
CC      kernel/time/tick-sched.o
CC      kernel/time/timer_stats.o
LD      kernel/time/built-in.o
CC      kernel/futex.o
CC      kernel/rtmutex.o
CC      kernel/dma.o
CC      kernel/cpu.o
CC      kernel/spinlock.o
CC      kernel/uid16.o
CC      kernel/module.o
CC      kernel/kallsyms.o
CC      kernel/acct.o
CC      kernel/kexec.o
CC      kernel/cgroup.o
CC      kernel/cpuset.o
CC      kernel/ns_cgroup.o
CC      kernel/stop_machine.o
CC      kernel/audit.o
```

```
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_ib.o
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_multicast.o
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_verbs.o
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_vlan.o
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_cm.o
CC [M]  drivers/infiniband/ulp/ipoib/ipoib_fs.o
LD [M]  drivers/infiniband/ulp/ipoib/ib_ipoib.o
CC [M]  drivers/infiniband/ulp/iser/iser_verbs.o
CC [M]  drivers/infiniband/ulp/iser/iser_initiator.o
CC [M]  drivers/infiniband/ulp/iser/iser_memory.o
CC [M]  drivers/infiniband/ulp/iser/iscsi_iser.o
LD [M]  drivers/infiniband/ulp/iser/ib_iser.o
CC [M]  drivers/infiniband/ulp/srp/ib_srp.o
CC [M]  drivers/input/ff-memless.o
CC [M]  drivers/input/input-polldev.o
CC [M]  drivers/input/joydev.o
CC [M]  drivers/input/evdev.o
CC [M]  drivers/input/evbug.o
CC [M]  drivers/input/joystick/a3d.o
CC [M]  drivers/input/joystick/adi.o
CC [M]  drivers/input/joystick/analog.o
CC [M]  drivers/input/joystick/cobra.o
CC [M]  drivers/input/joystick/db9.o
CC [M]  drivers/input/joystick/gamecon.o
CC [M]  drivers/input/joystick/gf2k.o
CC [M]  drivers/input/joystick/grip.o
```

```
install -p    -m 644 ./debian/templates.master /home/scheduler/linux-2.6.24-casi
o/debian/linux-image-2.6.24-casio/DEBIAN/templates
dpkg-gencontrol -DArchitecture=i386 -isp         \
                      -plinux-image-2.6.24-casio -P/home/scheduler/linux-2.6.2
4-casio/debian/linux-image-2.6.24-casio/
create_md5sums_fn () { cd $1 ; find . -type f ! -regex '.*/DEBIAN/.*' ! -regex '
./etc/.*'      ! -regex '.*lib/modules/[^/]*/modules\..*' -printf '%P\0' | xargs
 -r0 md5sum > DEBIAN/md5sums ; if [ -z "DEBIAN/md5sums" ] ; then rm -f "DEBIAN/m
d5sums" ; fi ; } ; create_md5sums_fn              /home/scheduler/linux-2.6.24-c
asio/debian/linux-image-2.6.24-casio
chmod -R og=rX                 /home/scheduler/linux-2.6.24-casio/debian/linux-i
mage-2.6.24-casio
chown -R root:root             /home/scheduler/linux-2.6.24-casio/debian/linux-i
mage-2.6.24-casio
dpkg --build                   /home/scheduler/linux-2.6.24-casio/debian/linux-i
mage-2.6.24-casio ..
dpkg-deb: construyendo el paquete `linux-image-2.6.24-casio' en `../linux-image-
2.6.24-casio_2.6.24-casio-10.00.Custom_i386.deb'.
make[1]: se sale del directorio `/home/scheduler/linux-2.6.24-casio'
====== making target stamp-kernel-image [new prereqs: linux-image-2.6.24-casio l
inux-image-2.6.24-casio]======
This is kernel package version 11.001.
echo done > stamp-kernel-image
====== making target kernel_image [new prereqs: stamp-configure stamp-build-kern
el stamp-kernel-image]======
This is kernel package version 11.001.
ivette@ivette-laptop:/home/scheduler/linux-2.6.24-casio$
```

```
ivette@ivette-laptop:/home/scheduler$ sudo dpkg -i linux-image-2.6.24-casio_2.6.
24-casio-10.00.Custom_i386.deb
Seleccionando el paquete linux-image-2.6.24-casio previamente no seleccionado.
(Leyendo la base de datos ...
99860 ficheros y directorios instalados actualmente.)
Desempaquetando linux-image-2.6.24-casio (de linux-image-2.6.24-casio_2.6.24-cas
io-10.00.Custom_i386.deb) ...
Done.
Configurando linux-image-2.6.24-casio (2.6.24-casio-10.00.Custom) ...
Running depmod.
Finding valid ramdisk creators.
Using mkinitramfs-kpkg to build the ramdisk.
Running postinst hook script update-grub.
Searching for GRUB installation directory ... found: /boot/grub
Searching for default file ... found: /boot/grub/default
Testing for an existing GRUB menu.lst file ... found: /boot/grub/menu.lst
Searching for splash image ... none found, skipping ...
Found kernel: /boot/vmlinuz-2.6.24-casio
Found kernel: /boot/vmlinuz-2.6.24-26-generic
Found kernel: /boot/memtest86+.bin
Replacing config file /var/run/grub/menu.lst with new version
Updating /boot/grub/menu.lst ... done


ivette@ivette-laptop:/home/scheduler$
```

```
Starting up ...
Uncompressing Linux... Ok, booting the kernel.
_
```