Universitat Oberta de Catalunya

LIBRERÍA PARA ANÁLISIS DINÁMICO DE PROGRAMAS

TFC - PLATAFORMA GNU/LINUX

JUNIO 2011

Oscar Mira Sánchez

Consultor: Joaquin López Sánchez

Enginyeria Tècnica en Informàtica de Sistemes

Agenda

- Introducción
- Tracing e interposición de librerías
- Desarrollo realizado
- Demostración
- Conclusiones

Introducción

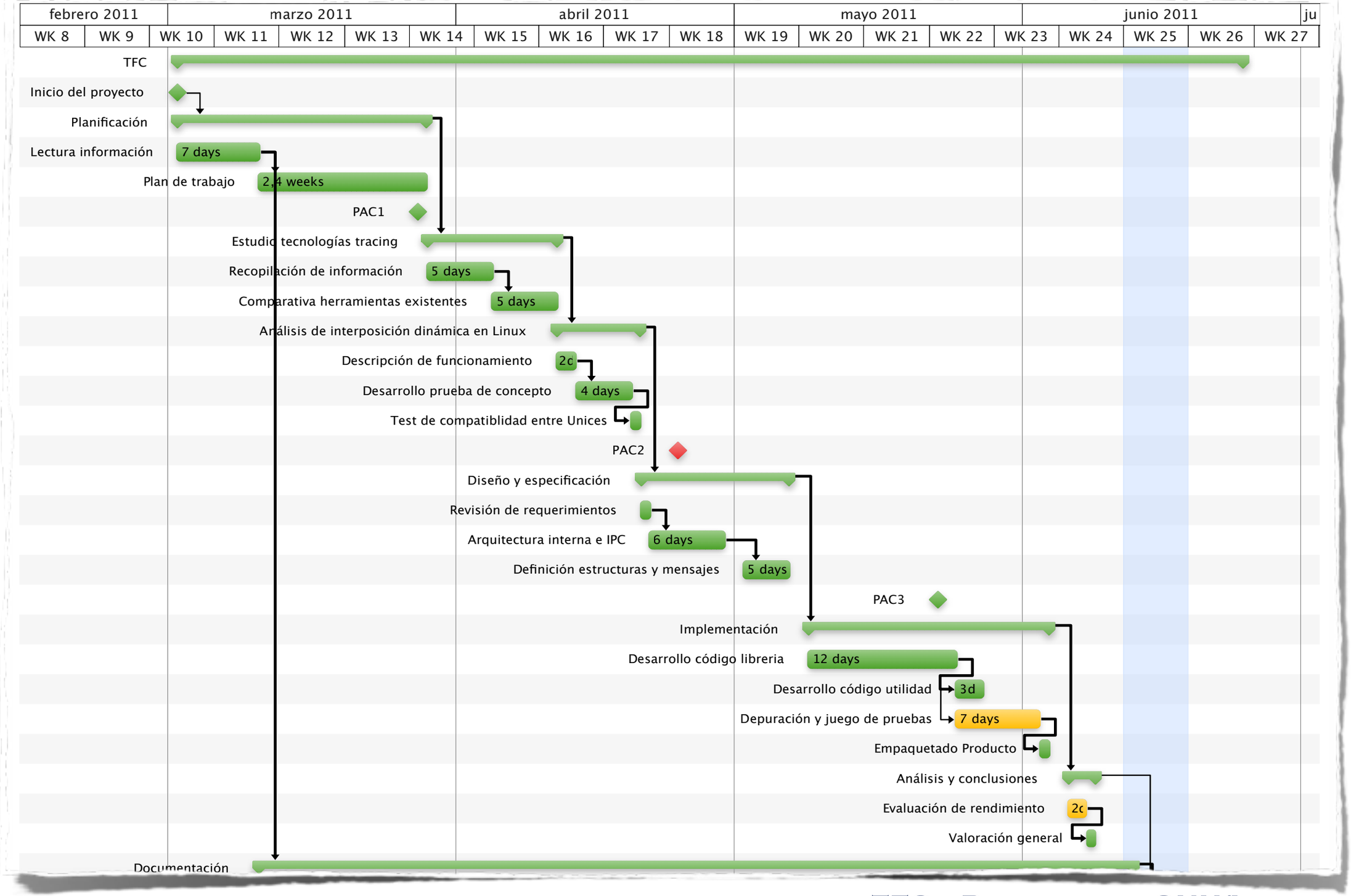
- Análisis dinámico de programas
  - Optimización
  - Corrección de errores
  - Detección de vulnerabilidades
  - Monitorización
  - *Tracers*, *profilers* y depuradores
- En Linux se pueden interceptar las peticiones al sistema:
  - Desde el Kernel
  - Desde otras aplicaciones (espacio de usuario)
- La idea: diseñar un sistema que registre dichas llamadas y las ponga a disposición de los analistas para realizar otros proyectos.

  ➔ Librería de interposición

Objetivos

- Estudiar las herramientas de tracing existentes en Linux.
- Analizar la técnica de interposición.
- Diseñar una librería de interposición en C.
- Desarrollar un prototipo.
- Evaluar la funcionalidad y el rendimiento de la librería.

Planificación



Agenda

- Introducción
- Tracing e interposición de librerías
- Desarrollo realizado
- Demostración
- Conclusiones

Tracing

- Registro de información significativa (evento) durante la ejecución de un programa:
  - Entrada / salida regiones de código (funciones, bucles, ...)
  - Interacción del proceso con el S.O. (*syscalls*)
- Registro de evento:
  - *Timestamp*
  - Identificador del proceso (*PID*)
  - Tipo de eventos + información específica
- Una traza es una secuencia de eventos ordenada en el tiempo.
- Cuando el objetivo es la optimización de software = *profiling*

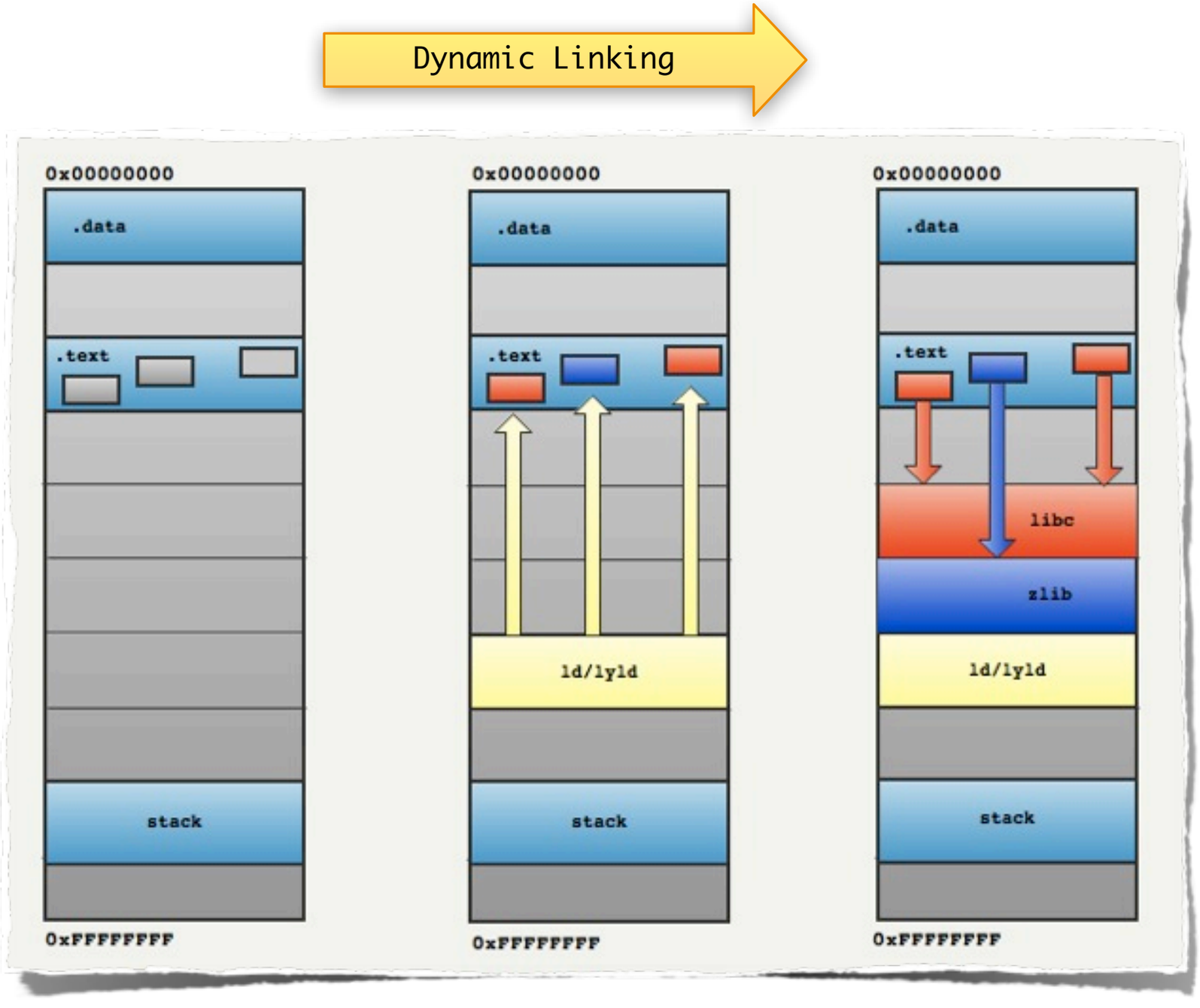
Adquisición de datos

- Mecanismos que permiten determinar el comportamiento del programa, identificar los eventos relevantes y capturarlos:
  - Instrumentalización del código fuente
    - GCC (*gcov*)
    - Lenguajes de alto nivel (*Java JVMPI*)
    - API de bajo nivel (*UST*)
  - Emulación binaria (*valgrind*)
  - Muestreo (*gprof*)
  - Instrumentalización dinámica (*dtrace, systemtap*)

  ➔ Nuestro proyecto: interposición con LD_PRELOAD

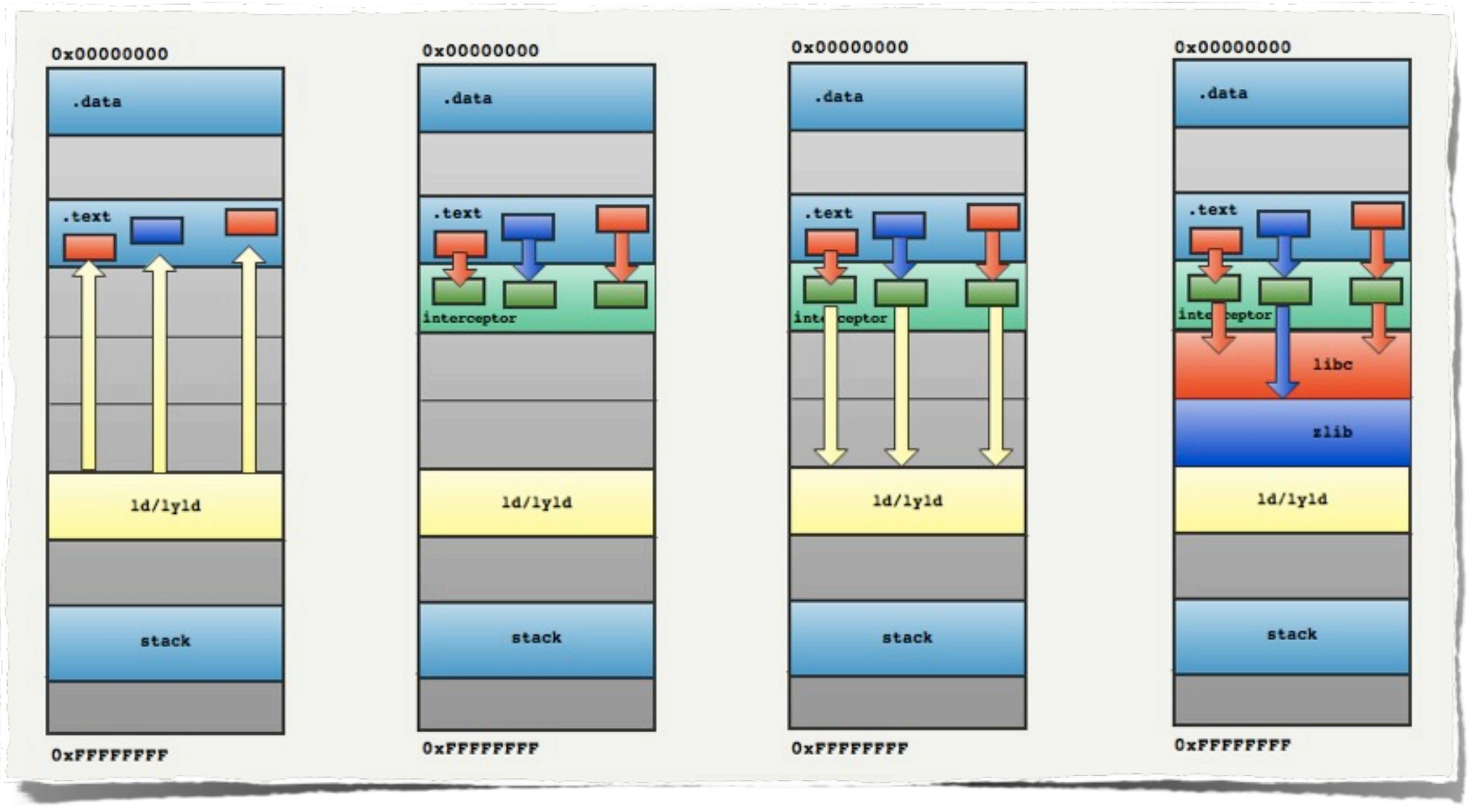
Librerias compartidas

- En Linux, los programas referencian funciones de librerías (*libc*, *zlib*, ...)
- El Kernel integra el cargador ***ld*** en el espacio de memoria virtual del proceso y le pasa el control.
- El cargador mapea en memoria las librerías y enlaza los símbolos externos del programa.
- Si se resuelve todos los símbolos, el programa comienza.



Interposición

- Configurando el cargador se puede alterar la carga de librerías e interponer código:



LD_PRELOAD

- Crear una librería con una función prototipo:

```
void *malloc(size_t size))
{
    ...; printf("malloc(%d) is called\n", size); ...;
}
```

- Compilar como librería compartida:

```
$: gcc -rdynamic -shared -ldl malloc_interposer.c
```

- Fijar la variable de entorno *LD_PRELOAD*

```
$: LD_PRELOAD=/lib/malloc_interposer.so cat /dev/null
“malloc(20) is called”
```

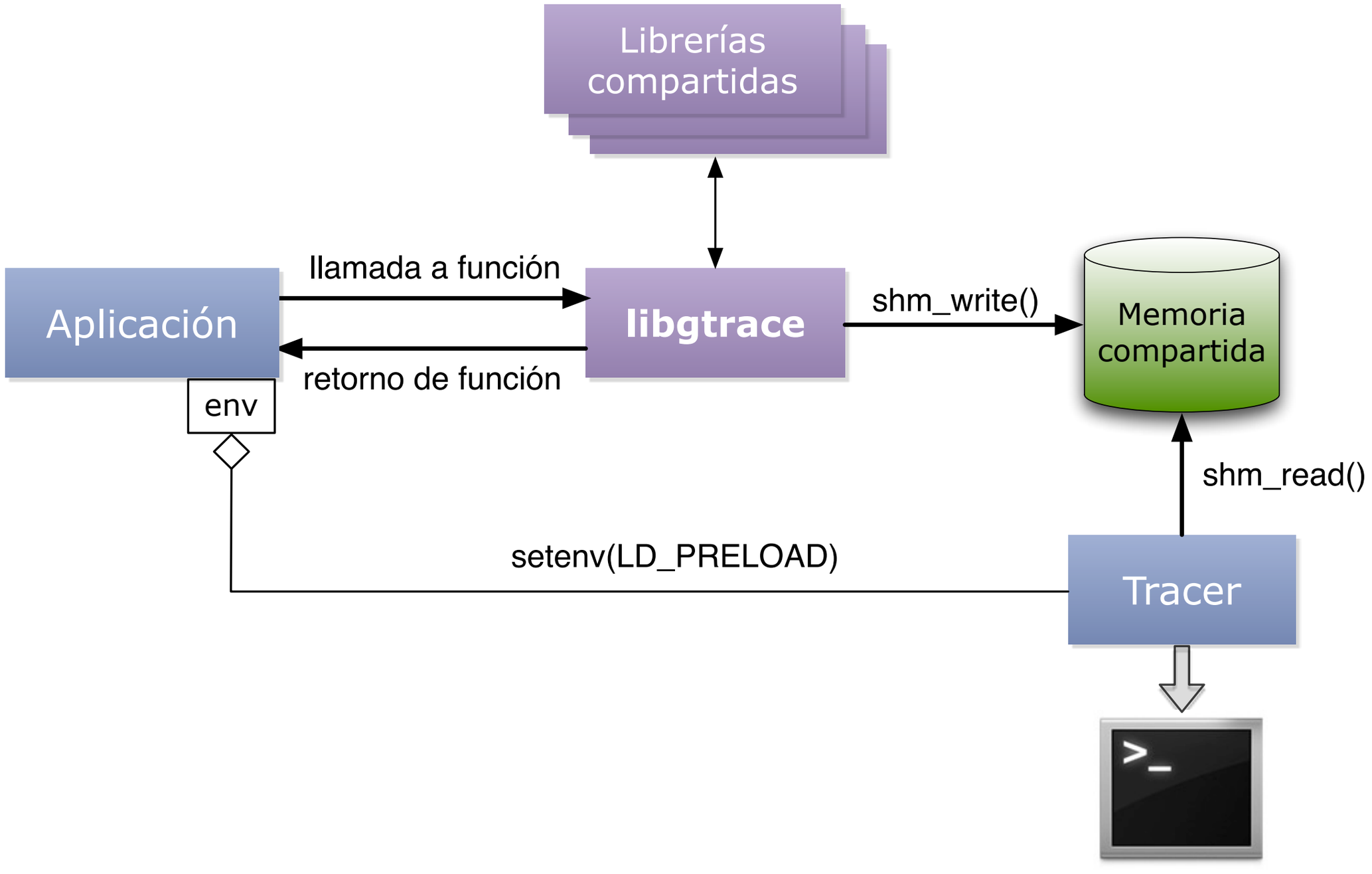
Agenda

- Introducción
- Tracing e interposición de librerías
- Desarrollo realizado
- Demostración
- Conclusiones

Requisitos de diseño

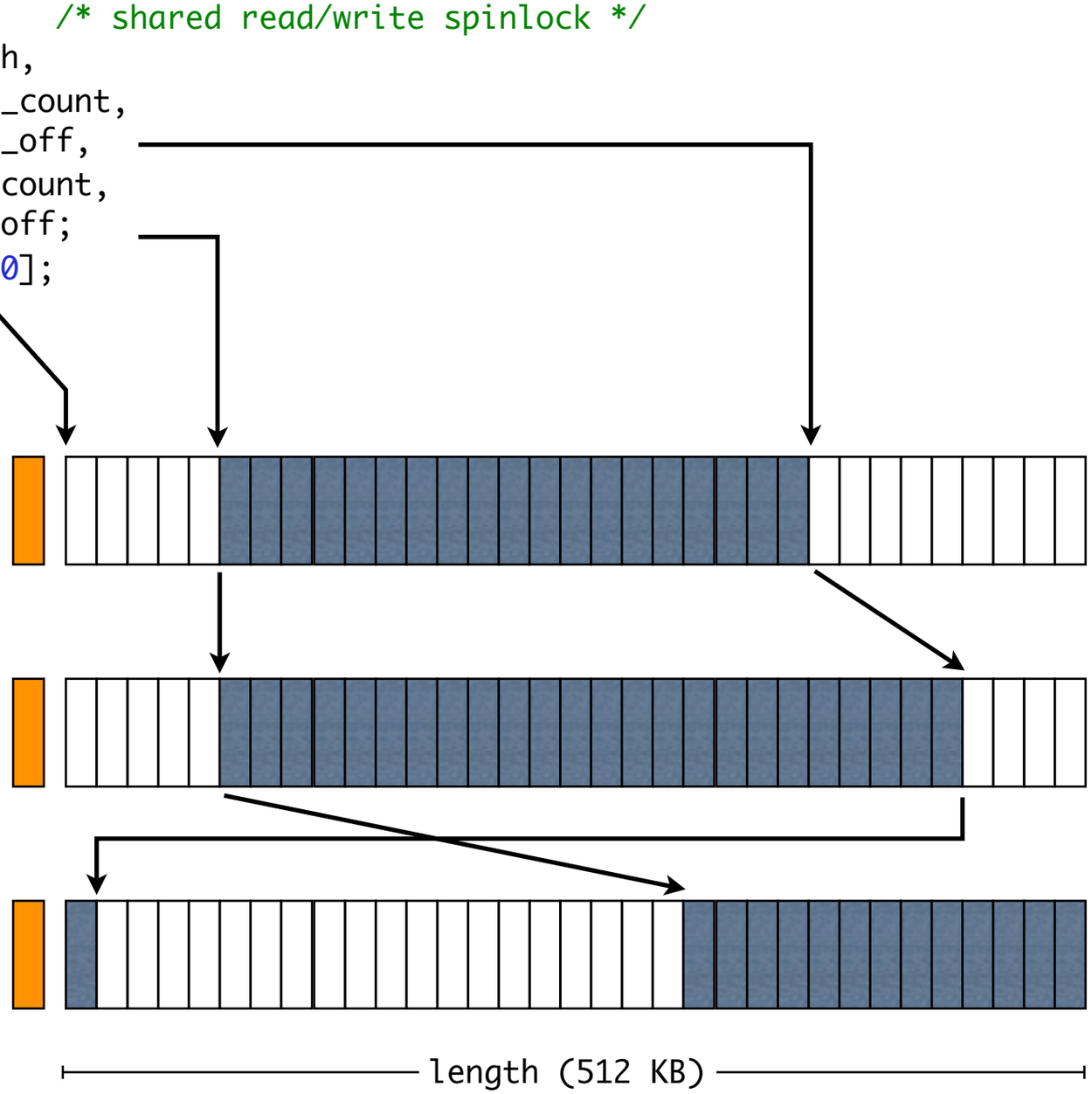
- Requisitos de la librería:
  - Capturar las llamadas a funciones, parámetros y valores de retorno, con un impacto mínimo en el rendimiento de la aplicación auditada.
  - Permitir subscribirse a eventos y responder en tiempo real a las llamadas de funciones capturadas.
  - Soportar múltiples procesos y *threads* simultáneamente.
  - Utilizar mecanismos IPC del estándar POSIX.
  - Funcionar en el espacio de usuario.
- Requisitos del prototipo:
  - Imprimir en pantalla los eventos recibidos.

Arquitectura



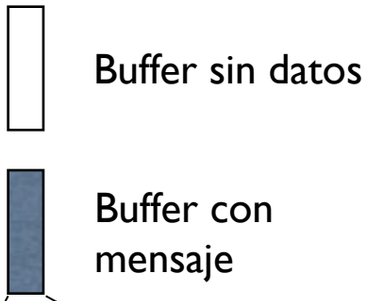
Memoria compartida

```
struct shm_record {
    volatile s32  lock;   /* shared read/write spinlock */
    volatile u32  length,
                  write_count,
                  write_off,
                  read_count,
                  read_off;
    volatile u8   data[0];
};
```



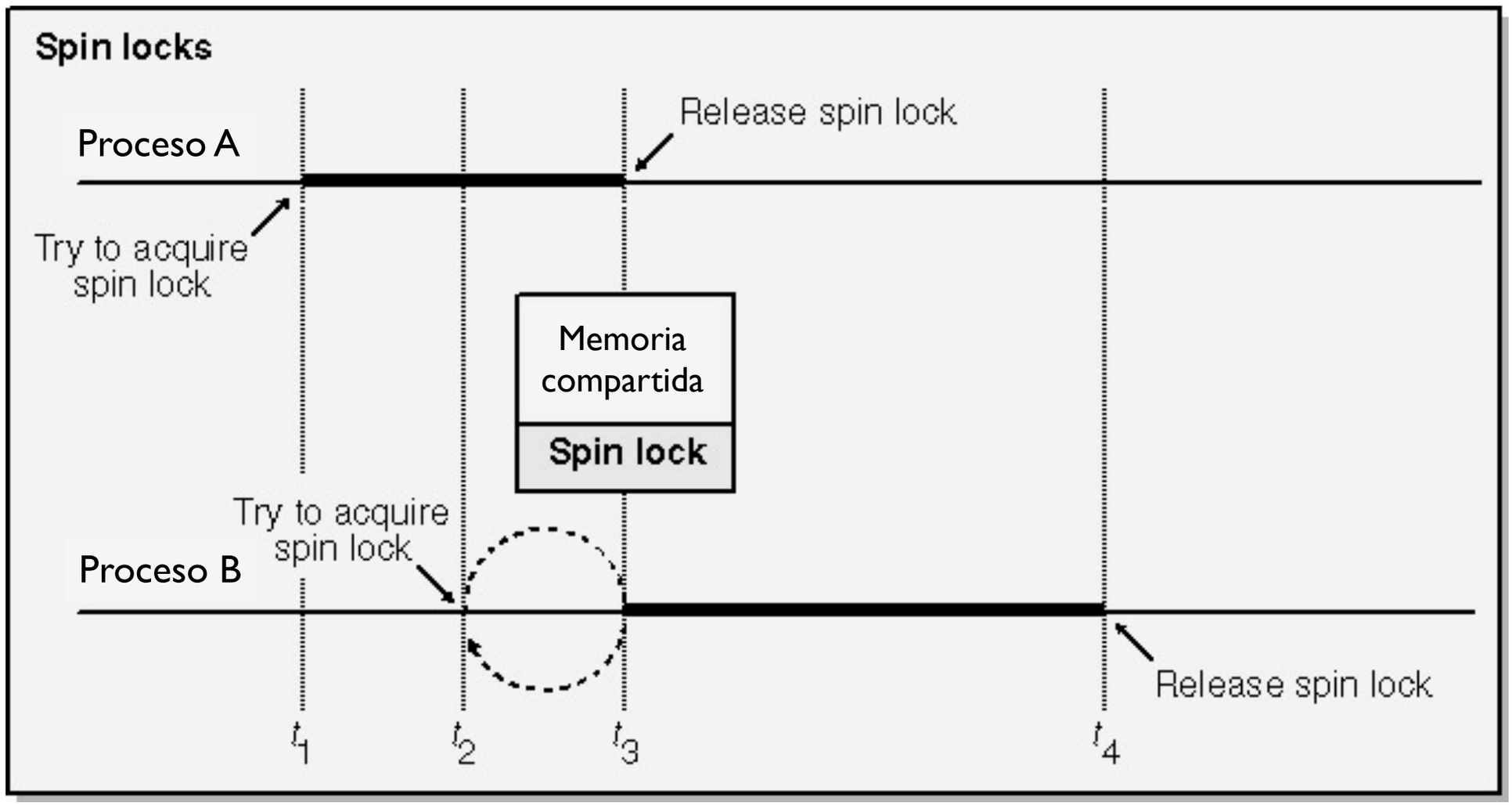
Escritura de datos
`write_off` se incrementa

Lectura y escritura de datos
`write_off` vuelve a cero



gtr_pid	3111
gtr_time	11:32:08.247322
gtr_type	GTR_LIBCALL
gtr_value	0x55642948 (fopen)
gtr_buffer	"/etc/passwd re"

- Las operaciones *shm_read()* y *shm_write()* esperan en un bucle hasta hacerse con el semáforo (variable *spinlock)*



Agenda

- Introducción
- Tracing e interposición de librerías
- Desarrollo realizado
- Demostración
- Conclusiones

GTRACE

- Utilidad prototipo que instrumentaliza 22 funciones (*fopen*, *puts*, ...)
- Compatible con GNU *automake* y *autoconf*
  - `$: ./configure && make install`
- Puede analizar múltiples procesos simultáneamente.

```
Trace library calls of a given program.

Usage:
  gtrace [options] -- [command [args ...]]

Example:
  gtrace -- ls -al

Options:
  -l path   Interposer library path. Default is /usr/local/lib/
libgtrace.so
  -f list   Trace only specified functions (comma separated list)
```

Gtrace

xoms@Hideo:~ — ssh — 102×38

```
[xoms@Hideo ~]$ gtrace -- uname -a
gtrace 0.1 by <xoms@uoc.edu>
06:43:32.901318 +++ trace of 'uname' started +++
06:43:32.901803 pid(24237): fopen (path "/proc/cpuinfo" mode r)
06:43:32.901823 pid(24237): fopen: returns 0
06:43:32.901825 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901861 pid(24237): fscanf: returns 3
06:43:32.901862 pid(24237): strcmp (s1 "processor" s2 "model name")
06:43:32.901863 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901864 pid(24237): fscanf: returns 3
06:43:32.901864 pid(24237): strcmp (s1 "vendor_id" s2 "model name")
06:43:32.901865 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901865 pid(24237): fscanf: returns 3
06:43:32.901866 pid(24237): strcmp (s1 "cpu family" s2 "model name")
06:43:32.901866 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901867 pid(24237): fscanf: returns 3
06:43:32.901867 pid(24237): strcmp (s1 "model" s2 "model name")
Linux Hideo 2.6.31-ARCH #1 SMP PREEMPT Tue Nov 10 19:01:40 CET 2009 x86_64 Inte
E8400 @ 3.00GHz GenuineIntel GNU/Linux
06:43:32.901867 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901868 pid(24237): fscanf: returns 3
06:43:32.901868 pid(24237): strcmp (s1 "model name" s2 "model name")
06:43:32.901871 pid(24237): fclose (fp 0x10e3030)
06:43:32.901879 pid(24237): fopen (path "/proc/cpuinfo" mode r)
06:43:32.901885 pid(24237): fopen: returns 0
06:43:32.901885 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901908 pid(24237): fscanf: returns 3
06:43:32.901909 pid(24237): strcmp (s1 "processor" s2 "vendor_id")
06:43:32.901909 pid(24237): fscanf (stream 0x10e3030 format "%64[^.:].:%256[^.]
06:43:32.901910 pid(24237): fscanf: returns 3
06:43:32.901910 pid(24237): strcmp (s1 "vendor_id" s2 "vendor_id")
06:43:32.901911 pid(24237): fclose (fp 0x10e3030)
06:43:32.901923 pid(24237): exit (status 0)
06:43:32.901927 pid(24237): fclose (fp 0x4826a780)
06:43:32.901930 pid(24237): fclose (fp 0x4826a860)
06:43:32.902004 +++ exited (code 0) +++
06:43:32.902009 +++ 31 messages processed (31 received) +++
[xoms@Hideo ~]$
```

xoms@Hideo:~/gtrace-0.1 — ssh — 80×2

xoms@Hide...e-0.1 — ssh | xoms@Hideo:~ — ssh

```
[xoms@Hideo gtrace-0.1]$ gtrace -f fork,exit
gtrace 0.1 by <xoms@uoc.edu>
LD_PRELOAD=/usr/local/lib/libgtrace.so; export LD_PRELOAD;
06:34:46.351340 +++ waiting for processes +++
06:34:55.65326 pid(24194): fork: returns 0
06:34:55.65551 pid(24195): fork: returns 0
06:34:55.65987 pid(24195): exit (status 0)
06:34:55.66054 pid(24194): fork: returns 24195
06:34:55.66126 pid(24194): exit (status 0)
06:34:55.66215 pid(24193): fork: returns 24194
06:34:55.86545 pid(24196): fork: returns 0
06:34:55.86831 pid(24196): fork: returns 24197
06:34:55.86956 pid(24198): fork: returns 0
06:34:55.87130 pid(24196): fork: returns 24198
06:34:55.87154 pid(24193): fork: returns 24196
06:34:55.106911 pid(24197): fork: returns 0
06:34:55.133481 pid(24198): exit (status 0)
06:34:55.133672 pid(24196): exit (status 0)
06:34:55.157764 pid(24199): fork: returns 0
06:34:55.158265 pid(24199): exit (status 1)
06:34:55.158370 pid(24193): fork: returns 24199
06:34:56.517247 pid(24193): exit (status 0)
```

Agenda

- Introducción
- Tracing e interposición de librerías
- Desarrollo realizado
- Demostración
- Conclusiones

Conclusiones

- He realizado un estudio sobre las tecnologías de tracing en Linux.
- He diseñado e implementado una librería de interposición, cumpliendo con los objetivos planteados.
- He mostrado la funcionalidad del prototipo y la viabilidad de la solución adoptada para futuro desarrollos.
- He aprendido el desafío técnico y organizativo que supone la realización de proyecto como este.