Platform-independent L4 version 2 system-call bindings

Michael Hohmuth

Dresden University of Technology
Department of Computer Science
D-01062 Dresden, Germany
email: hohmuth@inf.tu-dresden.de

January 5, 2000

1 Rationale

`XXX!` ***define new portable L4v2 bindings that are also forward-compatible with L4v4; therefore, we have to wait for Jochen's new L4v4 refman***

`XXX!` ***in the course, fix a number of problems present in existing bindings***

2 Experience with existing bindings

A quick poll among a few L4 developers revealed the following commonly-felt problems with the current L4 system-call bindings:

Portability. The bindings are not portable. The interface to the IPC system calls is optimized towards each architecture, and their platform-incompatibility is deliberately expressed in their architecture-specific names (e. g., `l4_i386_ipc_call`). The other interfaces have platform-independent names, but have subtle incompatibilities nonetheless. Table 1 lists some of the incompatibilities.

When we worked on L^{4}Linux, we felt we needed portable wrappers around the system-specific system-call bindings, so making a new set of bindings platform independent seems to be a worthwhile goal.

In–out parameters. It was a bad idea in the first place to come up with C bindings that use in–out parameters. This design decision has resulted in a lot of confusion and an uncountable number of bugs, and it should be reversed. All in–out parameters should be replaced by two parameters: an in parameter and an out parameter.

Specialized bindings. We should not only standardize on system-call bindings that are most general, but also on more specialized ones that use only selected aspects

L4/x86 system call	L4/MIPS incompatibility
l4_task_new l4_thread_ex_regs	have an extra argument (exception-handler thread)
l4_thread_schedule	MIPS replaces an in–out parameter with an in parameter plus an out parameter
l4_nchief	is called l4_id_nearest

Table 1 System-call incompatibilities between x86 and MIPS bindings

of one of the heavily overloaded L4 system calls. We should extend on the already existing specialized IPC system-call bindings and on `l4_myself` that hides aspects of the `l4_nchief` system call.

For example, it would be nice to have simple bindings for "short" (register-only) IPC that didn't require the often-unused message-buffer and message-dope parameters.

Optional parameters. Often, the values of out parameters are not required. However, as it is illegal to pass null pointers to the current bindings; the programmer has to declare dummy variables in order to pass valid references to the bindings. I suggest making most out parameters optional and allowing to pass null pointers to the default bindings. If desired, we can also provide "unsafe" bindings that are faster but do not test for null pointers. The default bindings' implementations then either have to test for null pointers, or they can use different inline-assembly stubs based on the value of out arguments. Modern compilers like Gcc provide facilities like `builtin_constant_p` that can be used to test for null pointers at compile time and optimize out unused code.

Consistency. Some bindings use both function-return values and out parameters to return data. This is confusing. Bindings should return data only either through return values or through out parameters. If out parameters are used, the return value should either be `void`, or it can optionally be used to return status information (error code, or just "success" (1) or "failure" (0)).

Cleanups. There also is the occasional small ugliness that can be eliminated while we are at changing the system-call bindings.

For example, the `l4_thread_ex_regs` binding takes an `l4_threadid_t` argument even though the system call only needs the local thread number. This results in a lot of code that takes great pains to correctly construct a valid thread ID, often using `l4_myself`, only for the binding to throw away most of the ID and pull out only the local thread number.

Where ugliness cannot be removed, it should at least be documented.

3 Proposal for new bindings

This proposal is based on the problems listed in Section 2. It is a **Work in progress!**

3.1 Types

We need some compiler-dependent and platform-dependent type definitions. The following definition shows example declarations for x86 hardware:

```
typedef signed int         l4_int_t; /* register-sized signed int */
typedef unsigned int       l4_uint_t; /* register-sized unsigned int */
typedef unsigned char      l4_bool_t; /* int type most effective as bool */

typedef signed char        l4_s8_t; /* 8-bit signed int */
typedef signed short int   l4_s16_t; /* etc. */
typedef signed int         l4_s32_t;
typedef signed long long   l4_s64_t;
typedef unsigned char      l4_u8_t; /* 8-bit unsigned int */
typedef unsigned short int l4_u16_t; /* etc. */
typedef unsigned int       l4_u32_t;
typedef unsigned long long l4_u64_t;

typedef unsigned char      l4_bool_t;

typedef l4_u32_t l4_addr_t;     /* memory address (ptr-sized unsigned int) */
typedef l4_u32_t l4_size_t;     /* difference between two addresses */
typedef l4_s32_t l4_ssize_t;    /* difference with sign */
```

The old types `dword_t` etc. have been removed because they cluttered the type name space and did not have a clear definition.

3.2 System-call interfaces

In the following subsections, I will propose binding interfaces for each system call, and I will give a rationale on each interface change compared to previous interfaces.

3.2.1 fpage_unmap

```
void          l4_fpage_unmap (l4_fpage_t fpage,
                              l4_uint_t map_mask);
```

The `map_mask` parameter is a bit set and does not have to be a special type—so use the native integer type for it.

3.2.2 id_nearest

```
void          l4_id_nearest (l4_threadid_t destination,
                             /* out */ l4_threadid_t *next_chief,
                             /* out */ l4_uint_t *type);
```

This binding before was called `l4_nchief` on the x86 platform which did not reflect the name of the L4 system call it binds to.

The `type` out parameter used to be a return value; this was inconsistent.

Specializations:

```
l4_threadid_t l4_myself (void);
```

No changes.

3.2.3 lthread_ex_regs

```
void          l4_thread_ex_regs (l4_uint_t lthread,
                                 l4_addr_t eip,
                                 l4_addr_t esp,
                                 l4_threadid_t excepter,
                                 l4_threadid_t pager,
                                 l4_threadid_t preempter,
                                 /* out */ unsigned int *old_eflags,
                                 /* out */ l4_addr_t *old_eip,
                                 /* out */ l4_addr_t *old_esp,
                                 /* out */ l4_threadid_t *old_excepter,
                                 /* out */ l4_threadid_t *old_preempter,
                                 /* out */ l4_threadid_t *old_pager);
```

> `XXX!` ***maybe rename to l4_lthread_ex_regs if this makes sense with L4v4***

Changed in–out parameters to pairs of in and out parameters.

The destination-thread argument used to be of type `l4_threadid_t` even though the kernel just wants a thread number. Changed to `l4_uint_t` so that users know they don't have to construct a valid thread id here.

> `XXX!` ***Jochen indicates that staying with a*** *`l4_threadid_t`* ***parameter type might be a good idea for L4v4 compatibility.***

The `eip` and `esp` arguments used to be of type `dword_t` which did not have a clear definition. Changed the type to `l4_addr_t` which is a platform-specific pointer-sized unsigned integer type.

Provided in and out parameters for both excepter for the MIPS architecture, and for preempter for x86. These arguments are ignored on architectures which don't use them.

3.2.4 thread_switch

```
void          l4_thread_switch (l4_threadid_t destination);
```

No changes.

Specializations:

```
void          l4_yield (void);
```

This was in use on x86 before, but is new for other architectures. Otherwise, no changes.

3.2.5 thread_schedule

```
int           l4_thread_schedule (l4_threadid_t dest,
                                  l4_sched_param_t param,
                                  l4_threadid_t ext_preempter,
                                  /* out */ l4_sched_param_t *old_param,
                                  /* out */ l4_threadid_t *old_ext_preempter,
                                  /* out */ l4_threadid_t *partner,
                                  /* out */ l4_cpu_time_t *time);
```

The CPU time used to be a return value. Changed this to be an out parameter.
The new return value is a boolean that flags successful execution.
Changed an in–out parameter (`partner`) to pair of in and out parameters.
Renamed `cpu_time_t` to `l4_cpu_time_t`.

3.2.6 task_new

```
int           l4_task_new (l4_taskid_t destination,
                           l4_uint_t mcp_or_new_chief,
                           l4_addr_t esp,
                           l4_addr_t eip,
                           l4_threadid_t excepter,
                           l4_threadid_t pager,
                           /* out */ l4_taskid_t *new_task);
```

The new task ID used to be a return value. Changed this to be an out parameter.
The new return value is a boolean that flags successful execution.
Added new `excepter` argument on x86. This parameter is required in MIPS but is ignored on other architectures.
The `dword_t` data type has been replaced by `l4_uint_t`.

Specializations:

```
int          l4_task_delete (l4_taskid_t destination,
                             l4_taskid_t new_chief,
                             /* out */ l4_taskid_t *new_task);
```

A new binding that just contains the task-deletion portion of `task_new`. This binding conveniently takes a `l4_taskid_t` argument instead of just the low doubleword of a task ID.

3.2.7 ipc

This system call does not have a generic binding—just specializations.

The system-specific IPC system-call bindings should be kept in their current form. However, we should standardize on portable bindings for both long-message IPC and a least-common-denominator version of short (register-only) IPC.

I this draft report I will give a just a small number of examples. More work is necessary to define standard bindings for all kind of IPC including deceiving IPC.

Least-common-denominator functions for short IPC: These functions transfer exactly two register values or one flexpage per IPC.

```
int          l4_ipc_send2 (l4_threadid_t dest,
                           l4_bool_t snd_fpage_p,
                           l4_uint_t snd_dword0,
                           l4_uint_t snd_dword1,
                           l4_timeout_t timeout,
                           /* out */ l4_msgdope_t *result);

int          l4_ipc_receive2
                          (l4_threadid_t src,
                           l4_fpage_t rcv_fp_option,
                           /* out */ l4_uint_t *rcv_dword0,
                           /* out */ l4_uint_t *rcv_dword1,
                           l4_timeout_t timeout,
                           /* out */ l4_msgdope_t *result);
```

Instead of a full-blown message descriptor, these bindings only accept a flag indicating that a flexpage is being sent (`snd_fpage_p`) or a receive-flexpage option, respectively.

All the other short-IPC bindings work likewise:

```
int          l4_ipc_wait2 (/* out */ l4_threadid_t *src,
                           l4_fpage_t rcv_fp_option,
                           /* out */ l4_uint_t *rcv_dword0,
                           /* out */ l4_uint_t *rcv_dword1,
                           l4_timeout_t timeout,
```

```
                         /* out */ l4_msgdope_t *result);

int           l4_ipc_call2 (l4_threadid_t dest,
                         l4_bool_t snd_fpage_p,
                         l4_uint_t snd_dword0,
                         l4_uint_t snd_dword1,
                         l4_fpage_t rcv_fp_option,
                         /* out */ l4_uint_t *rcv_dword0,
                         /* out */ l4_uint_t *rcv_dword1,
                         l4_timeout_t timeout,
                         /* out */ l4_msgdope_t *result);

int           l4_ipc_reply_and_wait2
                        (l4_threadid_t dest,
                         l4_bool_t snd_fpage_p,
                         l4_uint_t snd_dword0,
                         l4_uint_t snd_dword1,
                         /* out */ l4_threadid_t *src,
                         l4_fpage_t rcv_fp_option,
                         /* out */ l4_uint_t *rcv_dword0,
                         /* out */ l4_uint_t *rcv_dword1,
                         l4_timeout_t timeout,
                         /* out */ l4_msgdope_t *result);
```

Long-message IPC: The portable long-message IPC bindings hide the number of message words transferable in registers. Instead, the user is expected to put the whole message into the memory buffer. The bindings will then pull the register-transferred words out of the buffer before invoking the system call, and write the words received in registers into the buffer before returning to the user.

```
int           l4_ipc_send (l4_threadid_t dest,
                        const void *snd_buffer,
                        l4_timeout_t timeout,
                        /* out */ l4_msgdope_t *result);

int           l4_ipc_receive
                       (l4_threadid_t src,
                        void *rcv_buffer,
                        l4_timeout_t timeout,
                        /* out */ l4_msgdope_t *result);

int           l4_ipc_wait (/* out */ l4_threadid_t *src,
                        void *rcv_buffer,
                        l4_timeout_t timeout,
                        /* out */ l4_msgdope_t *result);
```

```
int            l4_ipc_call (l4_threadid_t dest,
                            const void *snd_buffer,
                            void *rcv_buffer,
                            l4_timeout_t timeout,
                            /* out */ l4_msgdope_t *result);

int            l4_ipc_reply_and_wait
                            (l4_threadid_t dest,
                             const void *snd_buffer,
                             /* out */ l4_threadid_t *src,
                             void *rcv_buffer,
                             l4_timeout_t timeout,
                             /* out */ l4_msgdope_t *result);
```

4 Open problems

- The new bindings probably should be in a different name space than `l4_*` to avoid incompatible changes to existing bindings.
- We also must define standard macros like `L4_IPC_TIMEOUT()`.
- Should we introduce a new data type like `l4_msghead_t` to be used instead of `void*` for message-buffer pointers?
- Which of the out-parameter references should be allowed be passed as null pointers?