Moroso IPC Specification

Amanda M. Watson, Carnegie Mellon University

October 13, 2014

This document defines the behavior of the IPC mechanism for the Moroso Architecture.

1 Introduction

Like many other microkernels, the Moroso userland uses Interprocess Communication (IPC) to enable communication across address spaces. In the Moroso operating system, IPC is used for the following:

Requests from Servers In Moroso, programs can request resources or services from application servers: this might be the status of a file, several pages of data, or a request from another program to relinquish a resource to them. With this in mind, IPC should allow threads to exchange both messages (statuses, requests) and resources (large amounts of memory), and make it possible for a requesting thread to block until the request or resource is complete – a requesting thread should be aware of when they alone have a resource without any ambiguity of ownership.

Synchronization The Moroso userland should provide developers with core thread synchronization primitives, such as `yield(), sleep,` and some approximation of Pebbles' `deschedule()/make_runnable()`. IPC should make it possible for these primitives to make guarantees about atomicity and synchronization between threads.

Signaling With many device drivers in userland, IPC should make it possible to forward interrupts and resources.

Performance While research suggests that IPC in modern microkernels is not the bottleneck it once was, as IPC is a core service, performance remains an important consideration. While its not clear yet how good performance should be defined, designs should attempt to not hinder it, optimizing for common cases as well as remaining configurable for the wide variety of workloads it might support.

While the userland may not have all of these features in place, these use cases represent the spirit of IPC, and the design is made with these in mind.

1.1 Blocking on Messages

In Moroso IPC, there are two different protocols for blocking on message exchange:

Synchronous When a message is sent synchronously, the sender blocks until a receiver either sends a reply or dies. A synchronous receive (note: all receives are synchronous in Moroso IPC) blocks until a sender completes a message transfer.

Asynchronous When a message is sent asynchronously, the sender is not blocked on a reply: instead, if no receiver is waiting, it returns as soon as the message is made available to the receiver: otherwise, it performs the transfer directly, but returns without a reply.

1.2 Types of Messages

To enable the exchange of both information and memory resources, IPC can send two different types of messages: long and short. Since each has different abilities and limitations, they are dealt with differently.

Long Message A long IPC message is a message that is larger than one page. To avoid the overhead of making copies of so much data or creating additional storage, long messages are specified by the page and transferred directly between virtual memory spaces: as such, all long messages are transferred synchronously.

Short Message a short IPC message is a message smaller than a page in size – this is what might be used for an exchange of requests, acknowledgements, or other short status information. As short messages are small enough to be copied into the kernel, they can be sent synchronously or asynchronously. Note that performance is less of a concern with asynchronous sends, and will almost certainly be slower.

Special Note: "Fast Path" While not a type of message per se, synchronous messages small enough to fit in the gpr's can be sent can be sent through the "fast path" without being buffered in the kernel. Inspired by the L4 Fast Path.

1.3 Additional Notes:

"Mailboxes" there is a 1-to-1 mapping between threads and "mailboxes", where messages are received. mailboxes are initialize when on thread creation, and are identified by the user tid.

Timeouts It's important to note that synchronous sends and receives have timeouts associated with them – timeouts can range from 0 (essentially polling) to infinite (no timeout). When the timeout is reached, the message expires, and the invoking thread returns failure.

2 Interface

The interfaces for the IPC functions are the following:

```
fn ipc_send(id: u32, message_long: *u8, message_short: *u8, len_long: u32,
            len_short: *u32, timeout: u32, shared: bool, option_sync: u32) -> i32
```

Sends a long and/or short message to thread `id`. Messages are received in the order in which they are sent.

Parameters:
`id`: The tid of the receiver thread. Function returns with error if thread `id` does not exist

`message_long`: The starting page of the long IPC message. Function returns with error if `message_long` is not page-aligned, or if it does not denote an existing user page (unless the message length is 0. See below)

`message_short`: The short IPC message. Function returns with error if `message_short` points to invalid memory (unless the message length is 0. See below). On success on a synchronous send, `message_short` is replaced with the receiver's reply.

`len_long`: length of the long message: on success `len_long` bytes worth of pages, starting at `message_long` are transferred to the receiver. When `len_long` is zero, the contents of `message_long` is ignored. Function returns with error if `len_long` is not a multiple of 4KB, or if the specified page range does not represent valid user pages in memory.

`len_short`: points to an address containing the length of `message_short`. if `*len_short` is 0, the contents of `message_short` is ignored. Function returns with error if `len_short` is an invalid pointer, or `*len_short` is larger than a 4KB page. On success on a synchronous send, `*len_short` is replaced with the length of the receiver's reply.

`timeout`: The number of ticks a synchronous send should wait for the receiver to reply before returning failure. This field is ignored if `option_sync` is 0 and `len_long` is zero. Otherwise, if `timeout` is set to 0, send returns immediately if the receiver is not currently blocked; if `timeout` is set to -1, there is no timeout, and the send will block infinitely until the receiver replies or dies.
Note: If the timeout occurs between the time when the receiver receives the message and the receiver replies, the receiver will not be notified of fthe timeout immediately: rather, `reply` will return with failure to indicate a timeout. This means no attempt should be made to reclaimor access resources relinquished to the receiver.

`shared`: denotes whether the pages being transferred for the long message should be shared. when `shared` is set to `true`, the pages remain mapped into

the sender's virtual memory, and the pages are shared between sender and receiver. Otherwise, they are unmapped from the sender's virtual memory, and the invoking thread will no longer have access to them.

`option_sync`: denotes whether send should be synchronous or asynchronous. If `len_long` is non-zero, this field is ignored. Otherwise, when `option_sync` is 1, messages are sent synchronously, blocking until they expire on `timeout` or the receiver sends a reply. If `option_sync` is 2, the same occurs except no reply is expected. If `option_sync` is 0, the send is asynchronous. Asynchronous messages do not wait for a reply and instead return as soon as a) a waiting receiver receives `message_short`, or, in the case that no receiver is waiting b) that `message_short` is accessible to the reciever.

Return Value:
On success, if the send was synchronous, or if the receiver was already waiting, `ipc_send` returns 0: this means the transfer to the receiver has successfully completed and, in the case of a synchronous send, the receiver replied. If the send was asynchronous and no receiver was waiting, returns 1. Returns a negative error code on failure.

```
fn ipc_recv(long_dest: *u8, short_dest: *u8, len_long: *u32, len_short:
            *u32, timeout: u32) -> i32
```

Blocks until it receives a long and/or short message from the next available sender, or the receive times out. Only one sender is received per invocation. Long messages are received starting at address `long_dest` up to `len_long` bytes, and short messages are written to `short_dest` up to `len_short` bytes. Note that, when `ipc_recv` returns from a synchronous send, the sender will still be blocked until `ipc_reply` is called.

Parameters:
`long_dest` The starting page where a long IPC message is mapped in. Function returns failure if `long_dest` is not page-aligned, or not a valid, unmapped userspace address (unless `len_long` is 0: see below)

`short_dest`: The address at which a short message is copied. Function returns failure if `short_dest` is not valid user memory (unless `len_short` is 0: see below)

`len_long`: Points to the maximum length in bytes of the memory range, starting at `long_dest`, that a long IPC message can be mapped into. if `*len_long` is 0, `long_dest` is ignored and no long IPC transfer occurs. On success, the length of the transfer is written to `*len_long`. Returns with failure if `len_long` points to invalid memory, the memory range contains existing pages, or if `*len_long` is not a multiple of 4KB.

`len_short`: Points to the length of the buffer a short message can be copied

into. A short message will only be copied into `short_dest` up to `len_short` bytes. If `*len_short` is 0, `dest_short` is ignored and no short IPC transfer occurs. On success, the length of the transfer is written to `*len_short`. Function returns with failure if `len_short` points to invalid memory.

`timeout`: the number of ticks before a receive expires, and the receiver returns with failure. If `timeout` is set to 0, receive returns immediately if no sender is currently blocked; if `timeout` is set to -1, there is no timeout, and the receive will block infinitely until it receives something.

Return Value:
Returns 0 if a synchronous sender is successfully received, 1 for an asynchronous send, a negative error code otherwise. Returns with failure if invoked before replying to the last synchronous sender.

`fn ipc_reply(message: *u8, message_len: u32) -> i32`
Replies to the last blocked sending thread with a short message `message`.

Parameters:
`message` the short reply to the sending thread. Function returns with error is `message` points to invalid memory (unless `message_len` is 0 – see below).

`message_len` the length of `message`. If `message_len` is 0, `message` is not copied to the sender, but the sender is still unblocked. Function returns with error if `message_len` exceeds 4KB.

Return Value:
Returns 0 if the sender was successfully replied to, a negative error code otherwise. An error is returned if the last synchronous sender has already been acknlowedged, or if the sender timed out and returned before `ipc_reply` could be invoked.