μ-Kernel Construction (4)

IPC Functionality & Interface
IPC Primitives

- Send to (a specified thread)
- Receive from (a specified thread)

- Two threads communicate
- No interference from other threads
- Other threads block until it’s their turn

Problem
- How to communicate with a thread unknown a priori (e.g., a server’s clients)
IPC Primitives

- **Send to**
  (a specified thread)

- **Receive from**
  (a specified thread)

- **Receive**
  (from any thread)

**Scenario**
- A client thread sends a message to a server expecting a response
- The server replies expecting the client thread to be ready to receive

**Problem**
- The client might be preempted between the *send to* and *receive from*
IPC Primitives

- Send to
  (a specified thread)
- Receive from
  (a specified thread)
- Receive
  (from any thread)
- Call
  (send to, receive from specified thread)
- Send to & Receive
  (send to, receive from any thread)
- Send to & Receive from
  (send to, receive from specified different thread)

- Are other combinations appropriate?

  **Atomic** operation to ensure that server’s (callee’s) reply cannot arrive before client (caller) is ready to receive.

  **Atomic** operation for optimization reasons. Typically used by servers to reply and wait for the next request (from anyone).
Message Types

- Registers
  - Short messages, avoid memory during IPC
  - Guaranteed to avoid user-level page faults during IPC

- Strings (optional)
  - In-memory messages copied from sender to receiver
  - May incur user-level page faults during copy operation

- Mappings (optional)
  - Messages that map pages from sender to receiver
  - Can map other resources too
IPC – API

- Operations
  - Send to
  - Receive from
  - Receive
  - Call
  - Send to & Receive
  - Send to & Receive from

- Message Types
  - Registers
  - Strings
  - Mappings
Problem

- How to deal with threads that are
  - Uncooperative
  - Malfunctioning
  - Malicious?

- How to prevent an IPC operation from never completing?
IPC – API

- Timeouts (v2, vX.0)
  - snd timeout, rcv timeout
IPC – API

- **Timeouts** (v2, vx.0)
  - snd timeout, rcv timeout
    - snd-pf timeout
      - specified by sender

- Attack through receiver’s pager
 IPC – API

- **Timeouts (v2, vx.0)**
  - snd timeout, rcv timeout
  - snd-pf / rcv-pf timeout
    - specified by receiver

- Attack through sender’s pager

![Diagram](image.png)
Timeout Problem

- Worst case IPC transfer time is high
  - Potential worst-case is a page fault per memory access
    - IPC time = send timeout + $n \times$ page fault timeout

- Worst-case for a careless implementation is unbound
  - Pager might respond with null mapping that does not resolve the fault
IPC – API

Timeouts \( (vX.2, v4) \)

- snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv

(time)

**wait for send** ➔ **send message** (xfer) ➔ **wait for reply** ➔ **receive message** (xfer)

- snd to
  - min (xfer to snd, xfer to rcv)
  - rcv to
  - min (xfer to rcv, xfer to snd)

*(specified by the partner thread)*
Timeout Issues

- What timeout values are typical or necessary?
- How do we encode timeouts to minimize space needed to specify all four values?

Timeout values

- $\infty$ (infinite)
  - Client waiting for a (trusted) server
- 0 (zero)
  - Server responding to a client
  - Polling
- Specific time
  - 1 us – 610 h (log)
Timeout Issues

- Assume short timeouts need finer granularity than long timeouts
  - Timeouts can always be combined to achieve long fine-grain timeouts

Timeout values
- $\infty$ (infinite)
  - Client waiting for a (trusted) server
- 0 (zero)
  - Server responding to a client
  - Polling
- Specific time
  - 1 us – 610 h (log)
IPC – API

- **Timeouts** *(vx.2, v4)*
  - snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv
  - relative timeout values
    - 0
    - infinite
    - 1 us ... 610 h (log) $2^e m \mu s$
 IPC – API

- **Timeouts** (vX.2, v4)
  - snd timeout, rcv timeout, xfer timeout snd, xfer timeout rcv

- **relative timeout values**
  - 0
  - infinite
  - 1 us ... 610 h (log)

- **absolute timeout values**

\[ \text{clock} = m_{(10)} + 0 \]

\[ \text{clock} + 2^{(e+10)} = m_{(10)} \]

© 2009 Universität Karlsruhe, System Architecture Group
Clarification of the “c” Bit

- User gives absolute timeouts relative to the current epoch (:= all but the least significant 10+e bits of clock).
- Kernel computes absolute timeout via “(clock’ & (~0ull << (10+e))) | (m << e)”, i.e., “epoch’ | (m << e)”.
  - The clock readings of the client and the kernel are different!

(a) Timeout 09:50, clock 09:45 => epoch 09:00 => delta := m << e = 50’
  - Kernel reached at clock’ 09:48 => epoch’ 09:00 => timeout 09:50 (ok)

(b) Timeout 10:12, clock 09:55 => epoch 09:00 => delta 1:12 (must be able to specify “next epoch”)
  (1) Kernel reached at clock’ 09:59 => epoch 09:00 => timeout 10:12 (ok)
  (2) Kernel reached at clock’ 10:01 => epoch 10:00 => timeout 11:12 (wrong)

Instead of specifying “this vs. next epoch” specify least significant bit (LSB) of target epoch:

(a) Timeout 09:50, clock 09:45 => epoch 09:00 => c = LSB(09) == 1, delta 50’
  - Kernel reached at clock’ 09:48 => epoch’ 09:00 => LSB(09) == 1 == c => epoch” 09:00 => timeout 09:50 (ok)

(b) Timeout 10:12, clock 09:55 => epoch 09:00 => c = LSB(10) == 0, delta 12’
  (1) Kernel reached at clock’ 09:59 => epoch’ 09:00 => LSB(09) == 1 != c => epoch” 10:00 => timeout 10:12 (ok)
  (2) Kernel reached at clock’ 10:01 => epoch’ 10:00 => LSB(10) == 0 == c => epoch” 10:00 => timeout 10:12 (ok)

Errors occur only if the epoch changes more than once between the client and the kernel reading the clock, i.e., if more than one complete epoch ((1<<(10+e)) μs ≈ (1<<(e) ms) passed in between.

As can be seen in (b1), using c is different from using more bits for the delta (effectively specifying the LSB of the target epoch): epoch’ is 09, having delta include the LSB would decrease this to 08 (LSB forced to 0); considering c != LSB(09) increases epoch’ to 10.

Do not waste your time understanding this - informational only!
# Timeout Range of Values (seconds) [v4, vx.2]

<table>
<thead>
<tr>
<th>$e$</th>
<th>$m = 1$</th>
<th>$m = 1023$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.000001</td>
<td>0.001023</td>
</tr>
<tr>
<td>1</td>
<td>0.000002</td>
<td>0.002046</td>
</tr>
<tr>
<td>3</td>
<td>0.000008</td>
<td>0.008184</td>
</tr>
<tr>
<td>5</td>
<td>0.000032</td>
<td>0.032736</td>
</tr>
<tr>
<td>7</td>
<td>0.000128</td>
<td>0.130944</td>
</tr>
<tr>
<td>9</td>
<td>0.000512</td>
<td>0.523776</td>
</tr>
<tr>
<td>11</td>
<td>0.002048</td>
<td>2.095104</td>
</tr>
<tr>
<td>13</td>
<td>0.008192</td>
<td>8.380416</td>
</tr>
<tr>
<td>15</td>
<td>0.032768</td>
<td>33.521664</td>
</tr>
<tr>
<td>17</td>
<td>0.131072</td>
<td>134.086656</td>
</tr>
<tr>
<td>19</td>
<td>0.524288</td>
<td>536.346624</td>
</tr>
<tr>
<td>21</td>
<td>2.097152</td>
<td>2145.386496</td>
</tr>
<tr>
<td>23</td>
<td>8.388608</td>
<td>8581.545984</td>
</tr>
<tr>
<td>25</td>
<td>33.554432</td>
<td>34326.18394</td>
</tr>
<tr>
<td>27</td>
<td>134.217728</td>
<td>137304.7357</td>
</tr>
<tr>
<td>29</td>
<td>536.870912</td>
<td>549218.943</td>
</tr>
<tr>
<td>31</td>
<td>2147.483648</td>
<td>2196875.772</td>
</tr>
</tbody>
</table>

1µs – 1023µs with 1µs granularity

Up to ~610h with ~35min granularity
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string

- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC
Ideally Encoded in Registers

- Parameters in registers whenever possible
- Make frequent/simple operations simple and fast

Sender Registers

- EAX
- ECX
- EDX
- EBX
- EBP
- ESI
- EDI

Receiver Registers
Example: Call-Reply

Thread A

pre

IPC call

pre

post

IPC reply & wait

Thread B

pre

IPC reply & wait

post

pre
Send and Receive Encoding

- 0 (Nil ID) is a reserved thread ID
- Define -1 as a wildcard thread ID

**Sender Registers**

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>destination</td>
</tr>
<tr>
<td>ECX</td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td>receive specifier</td>
</tr>
<tr>
<td>EBX</td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td></td>
</tr>
</tbody>
</table>

- **Nil ID** means “no send operation”
- **Wildcard** is not allowed (no broadcast support)

- **Nil ID** means “no receive operation”
- **Wildcard** means “receive from any thread”
Why use a single call instead of many?

- The implementation of the individual send and receive is very similar to the combined send and receive
  - We can use the same code
    - We reduce cache footprint of the code
    - We make applications more likely to be in cache
- L4 only implements combined “send to A and receive from B” syscall
  - A may but need not be equal to B
  - A or B may be 0 to avoid a send or receive phase
    - A == B == 0 is just a costly no-operation
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string

IPC syscall

- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC
Message Transfer

- Assume that **64 extra registers** are available
  - Name them $\text{MR}_0 \ldots \text{MR}_{63}$ (message register 0 ... 63)
  - All message registers are transferred during IPC
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceived IPC
Message Construction

- Messages are stored in registers ($MR_0 \ldots MR_{63}$)
- First register ($MR_0$) acts as message tag
- Subsequent registers contain
  - Untyped words ($u$)
  - Typed words ($t$) (e.g., map item, string item)
Message Construction

- Messages are stored in registers ($MR_0 \ldots MR_{63}$)
- First register ($MR_0$) acts as message tag
- Subsequent registers contain
  - Untyped words ($u$)
  - Typed words ($t$) (e.g., map item, string item)
Message Construction

- Typed items occupy one or more words
- Three currently defined items
  - Map item (2 words)
  - Grant item (2 words)
  - String item (2+ words)
- Typed items can have arbitrary order
Map and Grant Items

- Two words
  - Send base
  - Fpage
- Lower bits of send base indicates map or grant item

Semantics will be explained during memory management lecture
String Items

- Up to 4 MB (per string)
- Compound strings supported
  - Allows scatter-gather
- Incorporates cacheability hints
  - Reduce cache pollution for long copy operations

"hh" indicates cacheability hints for the string

E.g., only use L2 cache, or do not use cache at all
String Items

- New string specifier may of course contain substrings
- Different size compound strings require a new string specifier
- All substrings are of same size

"hh" indicates cacheability hints for the string
**IPC Parameters**

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceived IPC
String Reception

- Assume that **34 extra registers** are available
  - Name them $BR_0 \ldots BR_{33}$ (buffer register 0 \ldots 33)
  - Buffer registers specify
    - Receive strings
    - Receive window for mappings
Receiving Messages

- Receiver buffers are specified in registers ($BR_0 \ldots BR_{33}$)

- First BR ($BR_0$) contains "Acceptor"
  - May specify receive window (if not nil-fpage)
  - May indicate presence of receive strings/buffers (if s-bit set)
Receiving Messages

If **C**-bit in string item is cleared, it indicates that no more receive buffers are present.

A receive buffer can of course be a compound string.

If **C**-bit in string item is set, it indicates presence of more receive buffers.

The **s**-bit set indicates presence of string items acting as receive buffers.
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string

- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceived IPC
Timeouts

- Send and receive timeouts are the important ones
  - Xfer timeouts only needed during string transfer
  - Store xfer timeouts in predefined memory location

Sender Registers

- EAX: destination
- ECX: snd/rcv timeouts
- EDX: receive specifier

Receiver Registers

- Timeout values are only 16 bits
- Store send and receive timeout in single register
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string

- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceived IPC
Error conditions are exceptional
- Not common case
- No need to optimize for error handling

Bit in received message tag indicates error
- Fast check

Exact error code store in predefined memory location
IPC Result

- IPC errors flagged in $\text{MR}_0$
- Sender’s thread ID stored in register

### Sender Registers
- EAX: destination
- ECX: snd/rcv timeouts
- EDX: receive specifier

### Receiver Registers
- from
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- **IPC result code**
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceived IPC
IPC Redirection

- Redirection/deceiting IPC flagged by bit in the message tag
  - Fast check
- When redirection bit set
  - Thread ID to deceit as and intended receiver ID stored in predefined memory locations
IPC Parameters

- Send to
- Receive from
- Receive
- Call
- Send to & Receive
- Send to & Receive from
- Destination thread ID
- Source thread ID
- Send registers
- Receive registers
- Number of map pages
- Page range for each map page
- Number of send strings
- Send string start for each string
- Send string size for each string
- Receive window for mappings
- Number of receive strings
- Receive string start for each string
- Receive string size for each string
- Send timeout
- Receive timeout
- Send xfer timeout
- Receive xfer timeout
- IPC result code
- Sender thread ID
- Specify deceiting IPC
- Thread ID to deceit as
- Intended receiver of deceited IPC
Virtual Registers

- What about message and buffer registers?
  - Most architectures do not have 64+34 spare registers

- What about predefined memory locations?
  - Must be thread local
Virtual Registers

- What about message and buffer registers?
  - Most architectures do not have 64+34 spare registers

- What about predefined memory locations?
  - Must be thread local
What are Virtual Registers?

- Virtual registers are backed by either
  - Physical registers, or
  - Non-pageable memory

- UTCBs hold the memory backed registers
  - UTCBs are thread local
  - UTCBs cannot be paged
    - No page faults
    - Registers always accessible
Switching UTCBs (IA-32)

- Locating UTCB must be fast
  (avoid using system call)

- Use separate segment for UTCB pointer
  \[\text{movl} \%gs:0, \%edi\]

- Switch pointer on context switches
Switching UTCBs (IA-32)

- Locating UTCB must be fast  
  (avoid using system call)

- Use separate segment for UTCB pointer
  
  \[
  \text{movl} \ %gs:0, \ %edi
  \]

- Switch pointer on context switches
Message Registers and UTCB

- Some MRs are mapped to physical registers
- Kernel will need UTCB pointer anyway – pass it

Sender Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>destination</td>
</tr>
<tr>
<td>ECX</td>
<td>snd/rcv timeouts</td>
</tr>
<tr>
<td>EDX</td>
<td>receive specifier</td>
</tr>
<tr>
<td>EBX</td>
<td>( \text{MR}_1 )</td>
</tr>
<tr>
<td>EBP</td>
<td>( \text{MR}_2 )</td>
</tr>
<tr>
<td>ESI</td>
<td>( \text{MR}_0 )</td>
</tr>
<tr>
<td>EDI</td>
<td>UTCB</td>
</tr>
</tbody>
</table>

Receiver Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>from</td>
</tr>
<tr>
<td></td>
<td>( \text{MR}_1 )</td>
</tr>
<tr>
<td></td>
<td>( \text{MR}_2 )</td>
</tr>
<tr>
<td></td>
<td>( \text{MR}_0 )</td>
</tr>
<tr>
<td></td>
<td>UTCB</td>
</tr>
</tbody>
</table>
Free Up Registers for Temporary Values

- Kernel needs registers for temporary values
- $\text{MR}_1$ and $\text{MR}_2$ are the only values that the kernel may not need

<table>
<thead>
<tr>
<th>Sender Registers</th>
<th>Receiver Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>destination</td>
</tr>
<tr>
<td>ECX</td>
<td>snd/rcv timeouts</td>
</tr>
<tr>
<td>EDX</td>
<td>receive specifier</td>
</tr>
<tr>
<td>EBX</td>
<td>$\text{MR}_1$</td>
</tr>
<tr>
<td>EBP</td>
<td>$\text{MR}_2$</td>
</tr>
<tr>
<td>ESI</td>
<td>$\text{MR}_0$</td>
</tr>
<tr>
<td>EDI</td>
<td>UT CB</td>
</tr>
</tbody>
</table>
Free Up Registers for Temporary Values

- **Sysexit** instruction requires
  - ECX = user IP
  - EDX = user SP

<table>
<thead>
<tr>
<th>Sender Registers</th>
<th>Receiver Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>destination</td>
</tr>
<tr>
<td>ECX</td>
<td>snd/rcv timeouts</td>
</tr>
<tr>
<td>EDX</td>
<td>receive specifier</td>
</tr>
<tr>
<td>EBX</td>
<td>~</td>
</tr>
<tr>
<td>EBP</td>
<td>~</td>
</tr>
<tr>
<td>ESI</td>
<td>MR₀</td>
</tr>
<tr>
<td>EDI</td>
<td>UTCB</td>
</tr>
<tr>
<td></td>
<td>from</td>
</tr>
<tr>
<td></td>
<td>MR₁, MR₂, MR₀, UTCB</td>
</tr>
</tbody>
</table>
IPC Register Encoding

- Parameters in registers whenever possible
- Make frequent/simple operations simple and fast

<table>
<thead>
<tr>
<th>Sender Registers</th>
<th>Receiver Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>from</td>
</tr>
<tr>
<td>ECX</td>
<td>~</td>
</tr>
<tr>
<td>EDX</td>
<td>~</td>
</tr>
<tr>
<td>EBX</td>
<td>MR₁</td>
</tr>
<tr>
<td>EBP</td>
<td>MR₂</td>
</tr>
<tr>
<td>ESI</td>
<td>MR₀</td>
</tr>
<tr>
<td>EDI</td>
<td>UTCB</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MR₀</td>
</tr>
<tr>
<td></td>
<td>UTCB</td>
</tr>
</tbody>
</table>
Case study: IA-64

IPC Register Usage
Register Encoding on IA-64

All other registers are undefined
Register Stack Engine

Dirty partition (must be saved before use)
Clean partition (can be used directly)
Register Stack Engine
Register Stack Engine

Invalid partition (can be used directly)
Register Stack Engine
Register Stack Engine

Register Stack

Proc c
gr32
Proc b
Proc a

gr31
gr32

gr127

Physical registers

Proc c
Proc c

Proc b
Proc a

gr0

Backing storage
Register Stack Engine

Register Stack

Physical registers

Backing storage

Proc a

Proc b

gr_{32}

g_{r_{31}}

gr_{0}

gr_{32}

gr_{31}

gr_{127}
Register Stack Engine

Register Stack Physical registers

Proc a

gr_{32}  gr_{31}  gr_{30}  gr_{127}

Proc a

Proc a

gr_{0}  gr_{31}

Proc a

Register Stack  Physical registers  Backing storage
Register Stack Engine

Register Stack
Proc a

Physical registers
Proc a

Backing storage

gr_{32}
gr_{31}
gr_{0}
gr_{127}
gr_{32}
gr_{31}
Backing Store Switch for System Calls

- User backing store
- Register stack
- Kernel backing store

bspstore
Backing Store Switch for System Calls

User backing store

Register stack

Kernel backing store

bspstore
Backing Store Switch for System Calls

- User backing store
- Register stack
- Kernel backing store

bspstore
Register Stack During IPC

Backing store A  Register stack  Backing store B

bspstore  
gr_39
Msg

g_r32
Register Stack During IPC

First 8 message registers (64 bytes) are not saved and restored to/from memory.