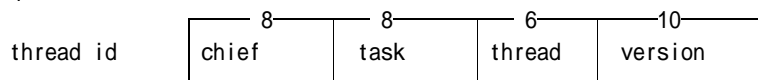


## § 1. 基本データタイプ

## Unique Ids

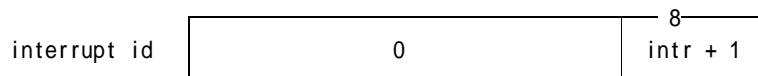
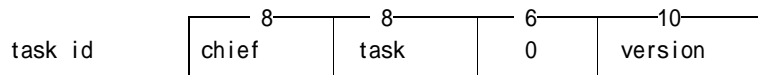


chief: チーフタスクの番号

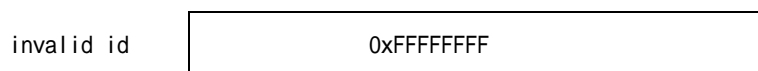
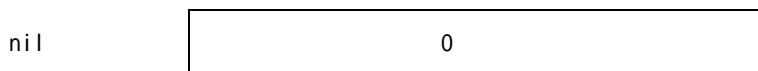
task: タスク番号

thread: タスク内ローカルのスレッド番号

version: L4-Kaでは、この値は使われない？



上位24ビットが 0、下位 8ビットが割り込み番号 + 1



## Fpages



× 4K番地から始まり、サイズが 2 の 乗のメモリ域を `fpage( , )` と表現する。

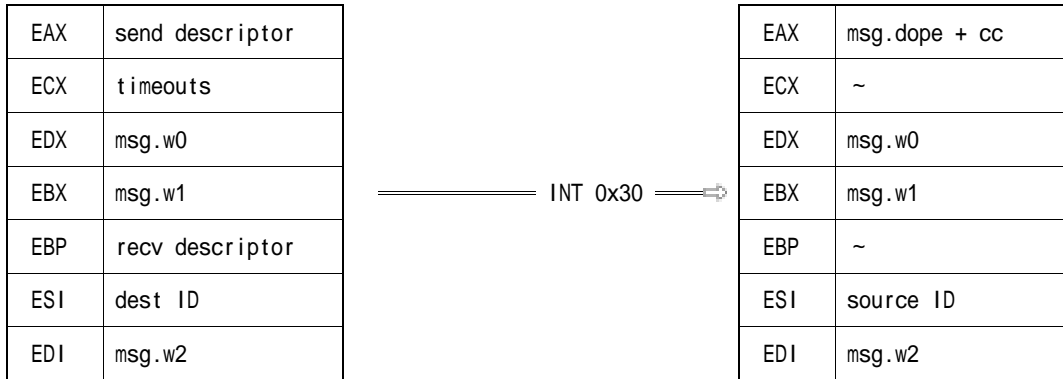


ページマッピングなどに使用する。

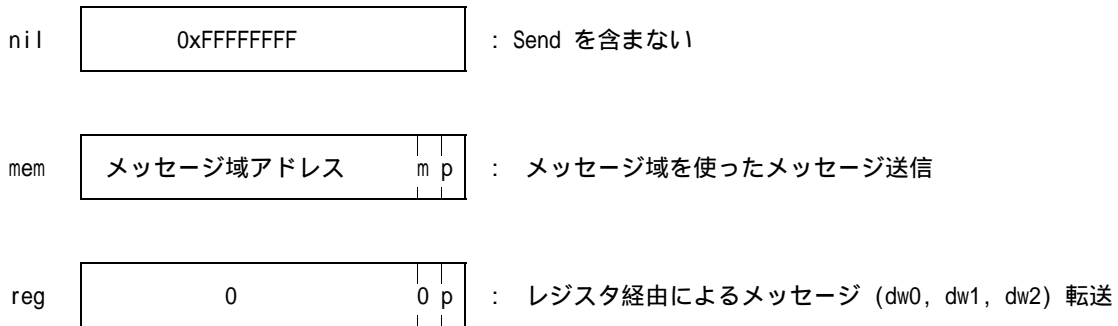
## § 2. メッセージ

- 同期型通信
- IPC = SEND + RECV
- メッセージの宛て先は、スレッドの UniquIDで指定。

### § 2.1 IPC 命令



### § 2.1 Send descriptor



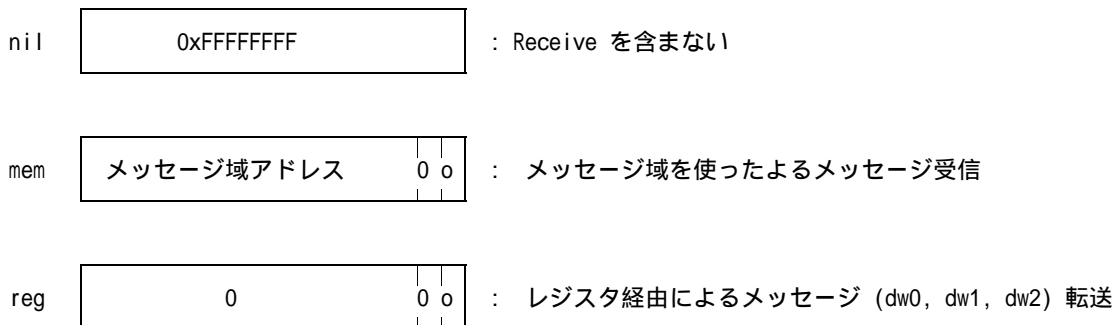
m {

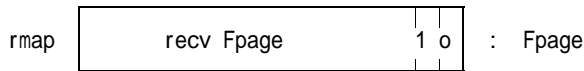
- 0 : 値コピーによるメッセージ転送
- 1 : Page mapping によるメッセージ転送

p {

- 0 : Unpropagated send operation
- 1 : Propagated send operation

### § 2.2 Recv descriptor



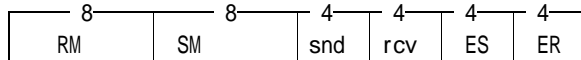


- 0 {
- 0 : Closed wait ( destIDで指定したスレッドからのみ受信)
  - 1 : Open wait (任意のスレッドから受信)

## § 2.2 Timeout

```
typedef struct {
    unsigned rcv_exp:4;
    unsigned snd_exp:4;
    unsigned rcv_pfault:4;
    unsigned snd_pfault:4;
    unsigned snd_man:8;
    unsigned rcv_man:8;
} l4_timeout_struct_t;

typedef union {
    dword_t raw;
    l4_timeout_struct_t timeout;
} l4_timeout_t;
```



$$\text{timeout} = 4 \cdot (15 - E) \cdot M \text{ } \mu\text{sec.}$$

```
#define L4_IPC_NEVER ((l4_timeout_t) { raw: 0})

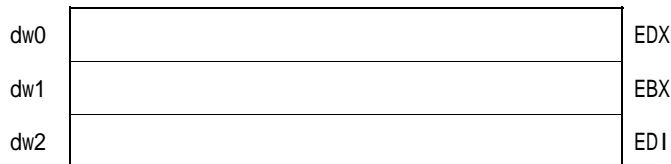
#define L4_IPC_TIMEOUT_NULL ((l4_timeout_t) { timeout: {15, 15, 15, 15, 0, 0}})

#define L4_IPC_TIMEOUT(snd_man, snd_exp, rcv_man, rcv_exp, snd_pflt, rcv_pflt) \
    ((l4_timeout_t) { \
        timeout: { rcv_exp, snd_exp, rcv_pflt, snd_pflt, snd_man, rcv_man } })
```

## § 3. メッセージのデータタイプ

### § 3.1 レジスタメッセージ

○ Send descriptor / Recv descriptor が "reg"を指定した場合。



【例】

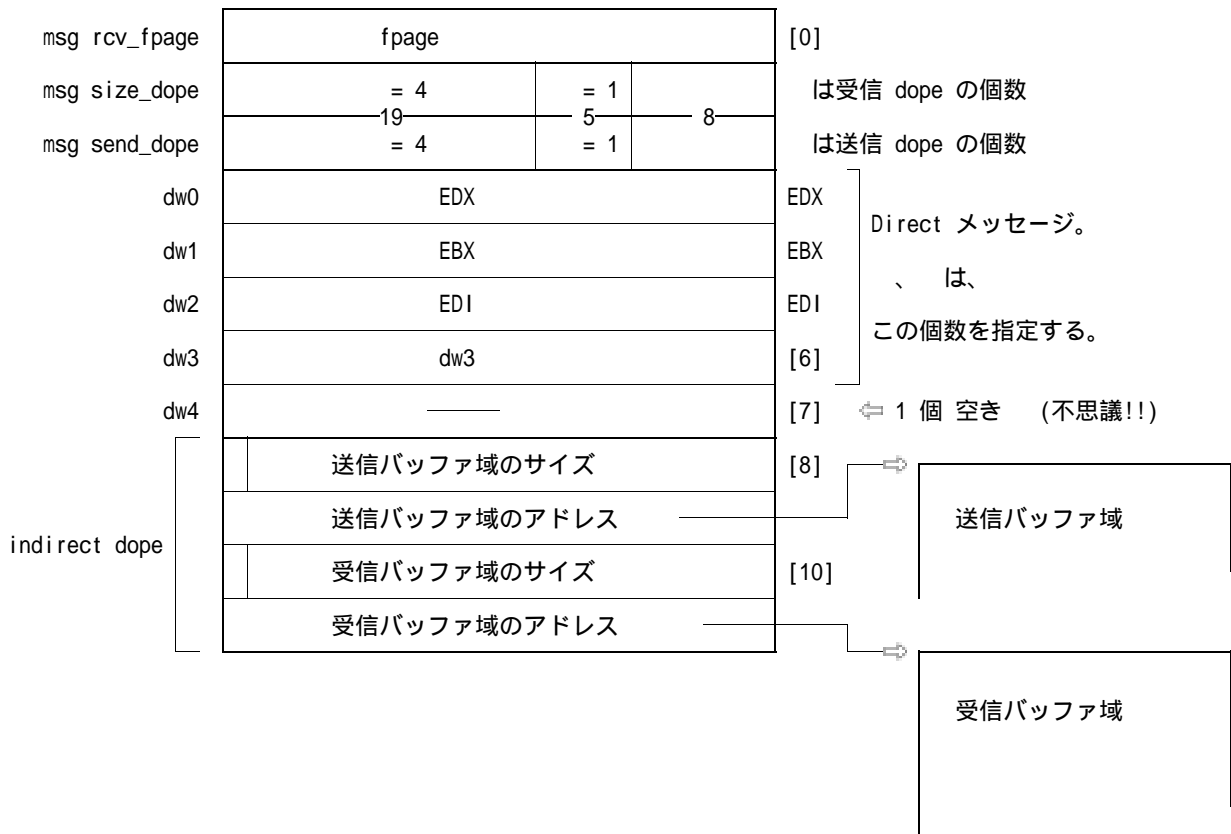
```
l4_threadid_t toThread;
dword_t out0, out1, out2;
l4_msgdope_t rcode;

l4_ipc_send (toThread, NULL, out0, out1, out2, L4_IPC_NEVER, &rcode);

l4_threadid_t fromThread;
dword_t in0, in1, in2;
```



### §3.3 メモリメッセージ+ Indirect メッセージ (バッファ間メッセージ)



【例】

```

l4_threadid_t  toThread;
dword_t  out0, out1, out2;
l4_msgdope_t  rcode;
dword_t  sendmsg[12];
char  sendbuf[1024];

sendmsg[0] = sendmsg[1] = 0;
sendmsg[2] = 8 << 13 | 1 << 8;
sendmsg[6] = 第3 パラメータ;
sendmsg[7] = 第4 パラメータ;
sendmsg[8] = sizeof (sendbuf);
sendmsg[9] = sendbuf;
sendmsg[10] = sendmsg[11] = 0;

l4_ipc_send (toThread, sendmsg, out0, out1, out2, L4_IPC_NEVER, &rcode);

l4_threadid_t  fromThread;
dword_t  in0, in1, in2;
l4_msgdope_t  rcode;
dword_t  rcvmsg[8];
char  rcvbuf[2048];

rcvmsg[0] = rcvmsg[2] = 0;
rcvmsg[1] = 8 << 13 | 1 << 8;
rcvmsg[8] = rcvmsg[9] = 0;
rcvmsg[10] = sizeof (rcvbuf);
rcvmsg[11] = rcvbuf;

l4_ipc_wait( &fromThread, rcvmsg, &in0, &in2, &in3, L4_IPC_NEVER, &rcode);

```

## § 4. Fpageを使ったページ転送

【例】 pager\_thread( ) の例

```
void pager_thread( )
{
    l4_threadid_t src;
    dword_t dw0, dw1, dw2;
    l4_msgdope_t result;
    int r;

    r = l4_ipc_wait(&src, L4_IPC_OPEN_I*C, &dw0, &dw1, &dw2,
                  L4_IPC_NEVER, &result);
    while(1)
    {
        if (dw0 == 1)
            dw1 = kernel-info-tableのアドレス;
        else {
            if (マップ可能な空きページがある){
                dw1 = マップするページのアドレス | 12 << 2 | 0x02;
                // Fpage = [ pagenum | 12 |1|0]
                // size writable
            }else {
                dw0 = dw1 = 0;
            }
        }
        r = l4_ipc_reply_and_wait(src, L4_IPC_SHORT_FPAGE /* 0x2 */,
                                dw0, dw1, dw2,
                                &src, L4_IPC_OPEN_I*C, &dw0, &dw1, &dw2,
                                L4_IPC_NEVER, &result);
    }
}
}
```

## § 5. IPC call

○ Cf. l4-ka/apps/include/l4/x86/ipc.h

○送受信

```
int l4_ipc_call(
    l4_threadid_t dest, //宛て先スレッドID
    void * send_descriptor, //送信記述子
    dword_t snd0,
    dword_t snd1,
    dword_t snd2,
    void * rcv_descriptor, //受信記述子
    dword_t * rcv0,
    dword_t * rcv1,
    dword_t * rcv2,
    l4_timeout_t timeout,
    l4_msgdope_t * result);
```

送信レジスタパラメータ

受信レジスタパラメータ

### ○返答 & 受信

```
int l4_ipc_reply_and_wait(
    l4_threadid_t dest, //宛て先スレッドID
    void * send_descriptor, //送信記述子
    dword_t snd0,
    dword_t snd1,
    dword_t snd2,
    l4_threadid_t * src, //送り主スレッドID
    void * recv_descriptor, //受信記述子
    dword_t * rcv0,
    dword_t * rcv1,
    dword_t * rcv2,
    l4_timeout_t timeout,
    l4_msgdope_t * result);
```

送信レジスタパラメータ

受信レジスタパラメータ

### ○送信

```
int l4_ipc_send(
    l4_threadid_t dest, //宛て先スレッドID
    void * send_descriptor, //送信記述子
    dword_t snd0,
    dword_t snd1,
    dword_t snd2,
    l4_timeout_t timeout,
    l4_msgdope_t * result);
```

送信レジスタパラメータ

### ○受信 (Open wait)

```
int l4_ipc_wait(
    l4_threadid_t * src, //送り主のスレッドID
    void * recv_descriptor, //受信記述子
    dword_t * rcv0,
    dword_t * rcv1,
    dword_t * rcv2,
    l4_timeout_t timeout,
    l4_msgdope_t * result);
```

受信レジスタパラメータ

### ○受信 (Close wait)

```
int l4_ipc_receive(
    l4_threadid_t src, //送り主のスレッドID
    void * recv_descriptor, //受信記述子
    dword_t * rcv0,
    dword_t * rcv1,
    dword_t * rcv2,
    l4_timeout_t timeout,
    l4_msgdope_t * result);
```

受信レジスタパラメータ

### ○割り込み処理スレッドの登録

あるスレッドの中で、下記の `associate_interrupt( )` を実行すると、パラメータで指定した IRQ-割り込みが発生すると、そのスレッドに割り込みを知らせるメッセージが配送される。

```
int associate_interrupt(int irq)
{
    dword_t dummy;
    l4_msgdope_t result;
```

```

        return l4_ipc_receive (L4_INTERRUPT(irq), 0, &dummy, &dummy, &dummy,
                               L4_IPC_TIMEOUT_NULL, &result);
    }

```

○割り込みメッセージの受信

○スリープ

## § 5. Sys call

○ Cf. l4-ka/apps/include/l4/x86/syscalls.h

○ Unmap

```

void l4_fpage_unmap(
    l4_fpage_t  fpage,
    dword_t    map_mask);

```

○ 自己のスレッドID

```

l4_threadid_t l4_myself( );

```

○スレッドの生成

```

void l4_thread_ex_regs(
    l4_threadid_t  destID,
    dword_t        eip,
    dword_t        esp,
    l4_threadid_t  * preempter,
    l4_threadid_t  * pager,
    dword_t        * old_eflags,
    dword_t        * old_eip,
    dword_t        * old_esp);

```

○

```

void l4_thread_switch(
    l4_threadid_t  destID);

```

○

```

cpu_time_t l4_thread_schedule ( ....)

```

○

```
l4_taskid_t l4_task_new (  
    l4_taskid_t  destID,  
    dword_t     mcp_or_new_chief,  
    dword_t     esp,  
    dword_t     eip,  
    l4_threadid_t pager);
```

○

```
void l4_yield( );
```