LP49 能動オブジェクトライブラリ libl4thread （書きかけ）

H_2O

平成 21 年 12 月 10 日

1 目的

本ライブラリ libl4thread は、L4 マイクロカーネル上で以下の機能を提供するものである。

能動オブジェクト（並行オブジェクト）支援： 能動オブジェクト（自前のスレッドを持ち、他と並列実行できるオブジェクト) を容易に実現できる。

分散処理； リモートノードとのメッセージ交信を容易に行えるようにする。

非同期メッセージ： L4 は同期メッセージのみを提供しているが、分散処理のためには非同期メッセージも望まれる。

Future reply： 相手スレッドに非同期メッセージで要求を送り、非同期で返される返答メッセージを受信する仕組み。

メッセージの alternative 受信：

CSP モデルサポート： Hoare の CSP モデルは、もっとも重要な並列処理モデルである。Plan9 の libthread ライブラリも CSP モデルのサポートを意図しているが、必ずしも簡明とは言えない。本ライブラリ libl4thread は、CSP 的処理も簡明に記述できることを狙っている。

マルチスレッド処理記述の容易化： L4 は融通性と効率に優れるが、使いこなすには相応のスキルを要する。本ライブラリにより、L4 の機能をより簡便に使うことができる。

L4 は、プロセス間ページマップなどの高度な機能も有しているが、使いこなすには相応のスキルを有する。本ライブラリでは、このような高度機能を容易に使えるように追々機能追加をしていく予定である。

2 能動オブジェクト

オブジェクト指向では、オブジェクトに何かをやってもらうにはメッセージを送る。つまり、**メッセージパッシング**することにより、そのオブジェクトのメソッドが実行される。メッセージパッシングとはいうものの、Java や C++で普通に作ったオブジェクトのメッセージパッシングは関数呼び出しに過ぎず、せいぜい実行すべきメソッドが動的に決定されるだけである。つまり、オブジェクトのメソッドは呼び出した側のスレッドによって実行される。(特に区別する場合は、このようなオブジェクトを、**受動オブジェクト**と呼ぶ。)

(メッセージパッシングという言葉に釣られて、Java や C++のオブジェクトが並列動作すると誤解している人もいるようですが、それは違います。)

能動オブジェクトとは、自前のスレッドを有したオブジェクトであり、メッセージを受けると自前のスレッドで目的メソッドを実行する。メッセージパッシングは、関数呼び出しではなく、文字通りスレッドへのメッセージ転送であり、各能動オブジェクトは並行動作できる。

能動オブジェクトのメッセージ転送の形態には、一般に次の 3 種類がある。M をメッセージ、C を送り手オブジェクト、S を受け手オブジェクトとする。

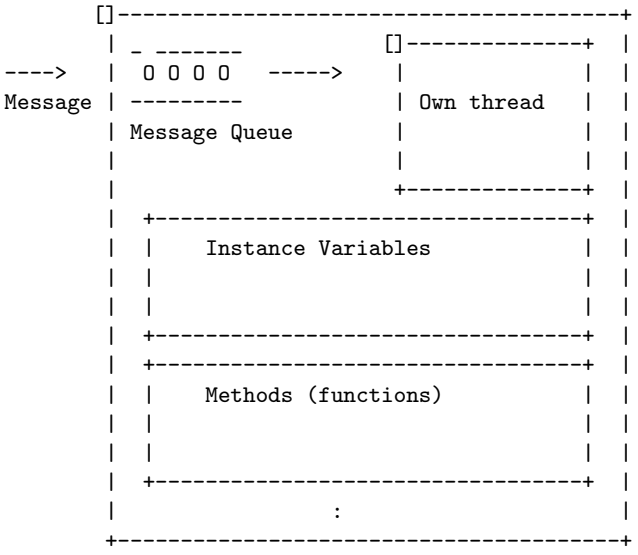
過去 (past) 型： C は S に M を送ったら、M が S によって実行されるのを待たずに、C は自分の処理を続行する。

現在型 (now)： C は S に M を送ったら、S から何らかの返答が戻ってくるまで待つ。

未来型 (future)： C は S に M を送ったら、返答が戻るのを待たずに、自分の処理を続行する。S はメッセージ M を処理して、応答を未来変数に入れる。C が S からの応答を使う場合には、未来変数をアクセスする。未来変数に応答が届いていなければ、届くまで自動的にまつ。

能動オブジェクトの仕組みを下図に示す。

【能動オブジェクトの仕組み】



能動オブジェクトの送られたメッセージは、メッセージ行列に挿入される。オブジェクトの自前スレッドは、メッセージ行列からメッセージを取り出しては、指定された処理を実行する。

3　本ライブラリの中核プレイヤー： L_thcb タイプ

(1) スレッド制御ブロック L_thcb

本ライブラリの中核プレイヤーは、**スレッド**（能動オブジェクトを含む）と**メッセージボックス**である。各スレッドは、非同期メッセージを受理するためのメッセージボックスと、非同期返答を受理するための利付等ーボックスを持っている。また、スレッドから独立したメッセージボックも可能であり、スレッド間の共通ボックスとして使える。

更に、自プロセス外 (リモートノードあるいは自ノードの別プロセス) のスレッドやメッセージボックスの場合は、それらにアクセスするために**代行者 (Proxy)** を用いる。

これらの中核プレイヤーを表現するのが、L_thcb テーブルである。各々は「L_thcb*ポインタ」によって識別される。

---【定義】--

```
typedef struct  L_thcb   L_thcb;
struct   L_thcb{
    L_thcb *        next;            //
```

```
    L4_ThreadId_t   tid;              // L4 スレッドの ID
    unsigned int    isthread: 1;      // スレッドの場合１
    unsigned int    ismbox: 1;        // メッセージボックスの場合１
    unsigned int    isrbox: 1;
    unsigned int    islocal: 1;       // ローカルの場合１
    unsigned int    islocalnode: 1;  // 同一ノード別プロセス内 --> Proxy
    unsigned int    isremotenode: 1; // リモートノード内 --> Proxy
    unsigned int    isactobj:1;       //能動オブジェクトの時１
    unsigned int          :1;
    unsigned int    state:8;
    unsigned int         :16;
    char  * name;
    int     lock;
    L_mbox  mbox;          // 非同期メッセージを入れるメッセージボックス
    L_replybox replybox;  //非同期返答メッセージを受けるリプライボックス
    union{
      struct{
        unsigned int    stkbase;
        unsigned int    stktop;
      } thd;
      struct{
         ....
      }proxy;
};
-------------------------------------------------------------
```

L_thcb テーブルは、以下を表す。

(isthread==1) の場合 スレッドである。スレッドは、非同期メッセージが挿入されるメッセージボックス L_thcb.mbox と非同期返答が挿入されるリプライボックス L_thcb.replybox を持っている。

スレッドの実行母体は L4 マイクロカーネルの L4 スレッドであり、フィールド L_thcb.tid で結ばれている。これはアドレスではなく、L4 スレッド ID L4_ThreadId_t である。L4 スレッドからは L4_UserDefinedHandle() を呼ぶことで、L_thcb テーブルが求まる。

(isthread==0 且つ ismbox==1) の場合 スレッド外の独立したメッセージボックスである。スレッドは自前のメッセージボックスからも、独立したメッセージボックスからもメッセージを受信できる。

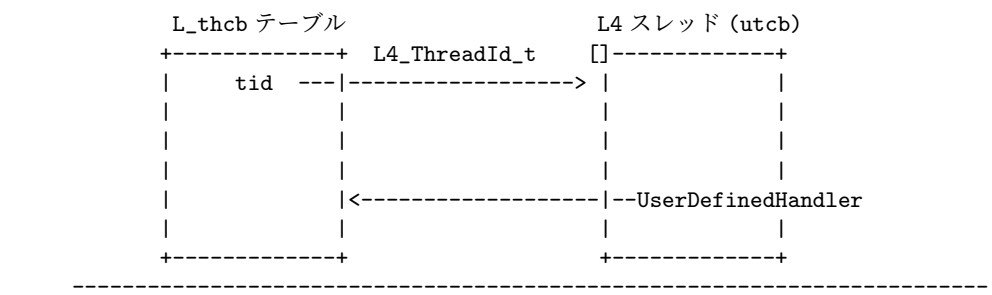
(islocal==0) の場合 別プロセスのスレッドあるいはメッセージボックスの Proxy である。メッセージは L4 の IPC で運ばれる。

(isremotenode==1) の場合 別ノードのスレッドあるいはメッセージボックスの Proxy である。メッセージは L4 の IPC で NW サービスに運ばれ、そこから目的ノードに転送される。(To be implemented)

スレッド実行の本体は、L4 マイクロカーネルが提供するスレッドである。下図に示すように、L_thcb テーブルから L4 スレッドへは、アドレスではなく L4_thcb.tid、つまり "L4_ThreadId_t" で結ばれている。L4 スレッドから L_thcb テーブルへは、UserDefinedHandler( ) で結ばれている。

スレッド外の独立したメッセージボックスは、L4 スレッドを持たない。

---【L_thcb と L4 スレッドの関係図】----------

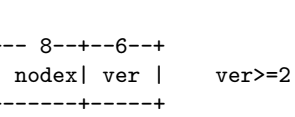


---

(2) Thread ID

L4 マイクロカーネルでは、スレッドは L4_ThreadId_t タイプで識別する。L4_ThreadId_t は、最上位 8 ビット procx をプロセス ID、次の 10 ビット tidx をプロセス内のスレッド番号、その次の 8 ビット nodex をノード番号（自分は 0)、最後の 6 ビット ver を version 番号として使用。

(tidx == 0) のスレッドは、プロセスの main スレッド、(tidx == 1) のスレッドは、非同期メッセージ処理を行うスレッドである。



(3) メッセージボックス L_mbox

---【定義】----------

```
typedef struct  L_mbox     L_mbox;
struct L_mbox{
    L_mbuf   *mqhead, *mqtail;
    L_thcb   *tqhead, *tqtail;
    int  lock;
};
```

メッセージボックスは非同期メッセージを貯めておく容器である。メッセージを運ぶメッセージバッファL_mbuf は, mqhead, mqtail が作る待ち行列に挿入される。

メッセージ行列が空きのときに読み出しを試みると、そのスレッドは、tqhead, tqtail が構成する待ち行列に挿入される。

(4) リプライボックス L_replybox

---【定義】----------

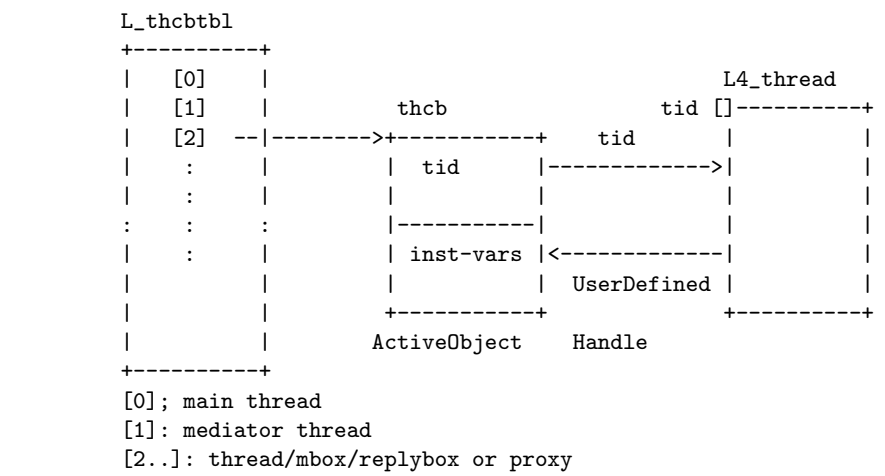
```
typedef struct  L_replybox L_replybox;
struct L_replybox{
    L_mbuf   *mqhead, *mqtail;
    L_ThreadId_t  receiver;
    int  lock;
};
```

TBD

(6) L_thcbtbl テーブル

---【構成図】--

```
extern L_thcb* L_thcbtbl[1024];
```



プロセス内の全L_thcb (スレッド、メッセージボックス等) は、L_thcbtbl テーブルに登録されている。

L_thcbtbl[0] は、プロセスの**メインスレッド**が登録されている。

L_thcbtbl[1] は、**mediator スレッド**が登録されている。プロセス外部からの非同期メッセージは、L4 IPC（同期型）を使って mediator スレッドに届けられる。Mediator スレッドは、非同期メッセージの宛先スレッドのメッセージボックスにメッセージを挿入する。

(7) 便宜定義

---【便宜定義】--

```
typedef L4_MsgTag_t     L_msgtag;

#define MLABEL(msgtag)   ((msgtag).raw>>18)

#define  TID2PROCX(tid)   (((tid.raw)>>24) & 0xff)
#define  TID2TIDX(tid)    (((tid.raw)>>14) & 0x3ff)
#define  TID2NODEX(tid)   (((tid.raw)>>6) & 0xff)
#define  TID2VER(tid)     ((tid.raw) & 0x3f)
#define  TID2THCB(tid)    (L_thcbtbl[TID2TIDX(tid)])
#define  A2TIDX(a)        TID2TIDX(((L_thcb*)a)->tid)

#define  INF   (-1)  // Infinite time in send/recv
```

---

4 L_mbuf: メッセージバッファ

---【定義】--

```
typedef union {
  L4_Word_t   raw;
  struct {
    L4_Word_t   u:6;
    L4_Word_t   t:6;
    L4_Word_t   flags:4;
    L4_Word_t   async:2 ;  //0: 動機メッセージ. 1: 非同期メッセージ, 2: 非同期返答
    L4_Word_t   mlabel:14; //メッセージ識別子, 非同期返答メッセージの割符
  } X;
} L_MR0;

typedef union {
  L4_Word_t   raw;
  struct {
    L4_Word_t   ver:6; // Version
    L4_Word_t   nodex_msgptn:8;  // msgptn = nump<<7 + nums<<4 + numi
    L4_Word_t   tidx:10;
    L4_Word_t   procx:8 ; // Num. of int-args
  } X;
} L_MR1;

typedef struct  L_mbuf  L_mbuf;
struct  L_mbuf {
    struct L_mbuf  * next;
    L4_ThreadId_t  sender;   // 非同期メッセージの場合、送り主のスレッド ID が入る
    union {
        L4_Msg_t       MRs;
        struct{
          L_MR0   mr0;
          L_MR1   mr1;
        }X;
    };
};
```

非同期メッセージは、メッセージバッファ L_mbuf に乗って運ばれる。

- next フィールドは、メッセージ行列を組むのに使われる。
- sender フィールドは、非同期メッセージの場合、送り主のスレッド ID が入る
- mr0 フィールドには、L4 のメッセージタグが載る。
- mr1 フィールドには、非同期メッセージのの宛先が載る。Mediator スレッドは、mr11 が指す宛先にメッセージを挿入する。

```
---【メッセージバッファ構成図】-----------------------------------------------
       MRs   Ex. "i2s2"
```

```
      +-------------------+
      |     next          |
      +-------------------+
      |     sender        |    Destination of asynchronous message
      +===================+
      |     mr0: L_MR0    | 0  L4 メッセージタグ
      +-------------------+
      |     mr1: L_MR1    | 1  非同期メッセージの宛先
      +-------------------+
      |      (e1)         | 2
      +-------------------+
      |      (e2)         | 3
      +-------------------+
      |     (size1)       | 4
      +-------------------+
      |     (str1)        |
      |                   |
      +-------------------+
      |     (size2)       |
      +-------------------+
      |     (str2)        |
      |                   |
      +-------------------+
-------------------------------------------------------------------------
```

5　スレッド/メッセージボックスの生成

(1) 生成

```
---【Signature】-------------------------------------------------------------------
L_thcb* l_thread_create(void (*func)(void*), int stksize, void *objarea);          //[1]

L_thcb* l_thread_create_args(void (*func)(void*), int stksize, void *objarea,
                             int argc, ...);                                      //[2]

L_thcb* l_mbox_create(char *name);                                                //[3]

void    l_thread_setname(char *name);                                             //[4]

void    l_yield(L_thcb *thcb);                                                    //[5]

// L_tidx l_proxy_connect(char *name, int nodeid, L4_ThreadId_t mate);            //[6]
-------------------------------------------------------------------------------------
```

[**1**] `l_thread_create(func, stksize, objarea)` は、ローカルスレッドを生成する。

- `func` は、スレッドが実行する関数
- `stksize` は、スタックサイズ (byte)
- `objarea` は、能動オブジェクト（後述）の場合はインスタンス変数域を、単なるスレッドの場合は`nil` を指定する。
  この値は、`func( )` の第 1 引数に引き継がれる。
  `objarea` を指定した場合は、メモリ域の開放 free(objarea) はユーザ責任である。

[**2**] スレッドの関数に objarea 以外の引数を引き継ぐ場合には、l_thread_create_args(...) を用いる。

- 引数はvoid*タイプに代入可能な値でなければならない。
- 第 1 引数は必ずobjarea である。
- argc で追加引数の数を指定し、その後ろに追加引数 (式) を並べる。

[**3**] l_mbox_create(name) は、独立したメッセージボックスを生成する。name はデバッグ用である。

[**4**] l_thread_setname(name) は、スレッドに名前 name をつける。name の長さは 15 bytes 以下。デバッグなどに使う。

[**5**] l_yield(thcb) は、実行を現スレッドから thcb スレッドに譲り渡す。(thcb == nil) の場合は、スケジューラが選ぶスレッドに譲り渡す。

(2) 消去など

```
---【Signature】------------------------------------------------------------------
int  l_thread_kill(L_thcb *thcb);           //[1]

void l_thread_exits(char *exitstr);         //[2]

void l_thread_killall();                    //[3]

void l_thread_exitsall(char *exitstr);      //[4]

int  l_proxy_clear(L_thcb  *thcb);          //[5]

void l_sleep_ms(unsigned  ms);              //[6]

int  l_stkmargin0(unsigned int stklimit);   //[7]

int  l_stkmargin() ;                        //[8]

int  l_myprocx();                           //[9]
-------------------------------------------------------------------------------------
```

... To Be Described ...

6 プロセス外部オブジェクトの登録

自プロセス外オブジェクトにアクセスするための proxy を設定する。

..... To Be Implemented .....

```
---【Signature】------------------------------------------------------------
L_thcb* l_thread_bind(char *name, L4_ThreadId_t mate, int attr);
int     l_thread_unbind(L_thcb* thcb);
-----------------------------------------------------------------------------
```

7　能動オブジェクトの定義法

(1) 基本

能動オブジェクトのインスタンス変数域は、L_thcb 域に連続して割り当てる。このために、第一要素を「L_thcb　_a;」とする構造体で定義する。

---【例】----------

```
typedef struct Alpha{                                   //[1]
    L_thcb  _a;       // L_thcb の継承に相当
    char    name[16];
    int     age;
    int     personalId;
} Alpha;

void alphafunc(Alpha *self)                             //[2]
{
   ..... thread body .....
   ....  self->age = 20; ....
}

Alpha     *a1;
L4_ThreadId_t  tid;

a1 = (Alpha*)malloc(sizeof(Alpha));                     //[3]
.... You can set value to instance vars like a1->age = 20; ...

l_create_thread(alphafunc, 4096, a1);                   //[4]
   // a1 の値は alpafunc() の第 1 引数に引き継がれる
   .....
tid = a1->_a.tid; //このようなアクセスももちろん可能
   .....
l_kill_thread(a1);                                      //[5]
free(a1);
```

[**1**] Alpha は能動オブジェクトのデータタイプを定義している。第一要素は「L_thcb　_a;」である。

[**2**] alphafunc() は Alpha オブジェクトのスレッドが実行する関数である。第 1 引数には、[4] で述べるように能動オブジェクトのデータ域のアドレスが引き継がれる。従って、self->name などとしてインスタンス変数にアクセスできる。
メッセージ受信とその処理の書き方は、後で説明する。

[**3**] 能動オブジェクトのメモリ域はユーザが割り当てを行う。

[**4**] 能動オブジェクトの生成は、第 3 引数に能動オブジェクト域を指定して、
l_create_thread(alphafunc, 4096, a1);
のように行う。生成が成功すれば第 3 引数の値が、失敗すればnil が反る。第 3 引数の値は、スレッドが実行する関数 [2] の第 1 引数として引き継がれる。

[**5**] 能動オブジェクトの削除はl_kill_thread() で行う。能動オブジェクト域の解放は、ユーザ責任である。

(2) 能動オブジェクト域と L4 スレッド ID 間の変換

1. 能動オブジェクトはL_thcb を継承 (拡張) したものである。従って、両者のアドレスは一致する。
2. 能動オブジェクトから、L4 オブジェクト ID を求めるには。
   ```
   L4_ThreadId_t  tid = self -> _a.tid;
   ```
3. L_MYOBJ マクロを使えば、実行中のスレッドから自分の能動オブジェクト域を求められる。

   ```
   ---【L_MYOBJ マクロ】---------------------------------------
     #define L_MYOBJ(type)   ((type)L4_UserDefinedHandle())
       Ex.  Alpha  *me;
            me = L_MYOBJ(Alpha*);
   ----------------------------------------------------------------
   ```

4. L_AOBJ() マクロを使えば、指定スレッドの能動オブジェクト域を求められる。

   ```
   ---【L_AOBJ() マクロ】---------------------------------------
   #define L_AOBJ(type, tid)  ((type)L4_UserDefinedHandleOf(tid))
      Ex. Alpha *you;
          you = L_AOBJ(Alpha*, tid);
   ----------------------------------------------------------------
   ```

8　メッセージバッファ及び L4 仮想レジスタ MRs の設定

同期メッセージは、L4 マイクロカーネルのメッセージ機能そのままである。L4 スレッドは 64 ワードの” 仮想” レジスタ MRs を有している。送信は、メッセージを MRs に設定してL4_send() を呼び出す。スレッドがメッセージを受信しようとする (Ex. L4_wait()、L4_receive()) と、メッセージが到着した時にその値は MRs に設定される。

非同期メッセージは、メッセージバッファL_mbuf に載せて送られ、宛先のメッセージボックス L_mbox に挿入される。

この説明から分かるように、メッセージの送受信には” 仮想” レジスタ MRs あるいは、メッセージバッファ L_mbuf のデータの設定や取り出しを行う。この操作を簡便に行うのがl_putarg(), l_getarg() である。

```
---【Signature】-----------------------------------------------------
○   mbuf はメッセージバッファ L_mbuf へのポインタであるが、値が nil の場合は、
  L4 スレッドの ”仮想"レジスタ MRs が使われる

L_mbuf* l_putarg(L_mbuf *mbuf, int mlabel, char *fmt, ...);     //[1]
  Ex.  L_mbuf  *mbuf;
       char buf1[...], buf2[...];
       mbuf = l_putarg(mbuf, 747, "i2s2", e1, e2, buf1, len1, buf2, len2)

int  l_getarg(L_mbuf *mbuf, char *fmt, ...);                    //[2]
  Ex.  int  mlabel
       int  x1, x2;
```

```
        char bufa[...],  bufb[...];
        int  lena,  lenb;
        mlabel = l_getarg(mbuf, "i2s2", &x1, &x2, bufa, &lena, bufb, &lenb)
-------------------------------------------------------------------------------
```

以下において、第 1 引数 mbuf が nil の場合は “仮想” レジスタ MRs , mbuf が否 nil の場合はメッセージバッファを意味する。

[**1**] `l_putarg(mbuf, mlabel, fmt, ...)` は, 送信データ設定を行う。mlabel はメッセージの先頭ワードの mlabel となる。fmt は引数のタイプと数を “iMsN” の形で指定する。M,N は整数値である。例えば、“i2s1” は sizeof(int) の値を 2 個、byte 列を 1 個持つことを意味する。

【例】`l_putarg(mbuf, 747, "i2s2", e1, e2, buf1, len1, buf2, len2)` は、mlabel に 747 を設定、2 個の “i” 型の引数（e1, e2) と、2 個の “s” 型の引数（開始アドレスと byte 長で与える）を設定する。

[**2**] `l_getarg(mbuf,  fmt, ...)` は, 受信データ取り出しを行う。

【例】 `mlabel = l_getarg(mbuf, "i2s2", &x1, &x2, bufa, &lena, bufb, &lenb);` の返り値はメッセージの mlabel である。引数は”i2s2” に従い第 1, 第 2 引数は変数 x1, x2 に代入され、第 3、第 4 引数はバッファ bufa （サイズは lena), bufb(サイズは lenb) lena, lenb にはバッファ長を入れておく。実行後には実際に代入されたサイズが設定されている。

9 Timeout

10 同期メッセージの送受信

```
---【Signature】-------------------------------------------------------------------
L_msgtag  l_send0(L_thcb *thcb, int msec, L_mbuf *mbuf);          //[1]
  Ex. L_mbuf *mbuf;
      mbuf = l_putarg(nil, mlabel, "i2s1", e1, e2, &buf, sizeof(buf));
      msgtag = l_send0(thcb, INF, mbuf);

L_msgtag  l_recv0(L4_ThreadId_t *client, int msec, L_mbuf **mbuf);  //[2]
     (第 1 引数のタイプは L4_ThreadId_t であり、l_reply0() の返答先として使われる。
      L_thcb* でないことに注意。)
  Ex. msgtag = l_recv(&client, INF, nil);
      n = l_getarg(nil, "i2s1", &x, &y, &buf2, &sz2);

L_msgtag  l_recvfrom0(L4_ThreadId_t client, int msec, L_mbuf **mbuf); //[3]
  Ex. msgtag = l_recvfrom(client, INF, nil);

L_msgtag  l_reply0(L4_ThreadId_t client, L_mbuf *mbuf);             //[4]
```

```
  Ex. msgtag = l_reply0(client, mbuf);

L_msgtag  l_call0(L_thcb *thcb, int smsec, int rmsec, L_mbuf *mbuf); //[5]
  Ex. msgtag = l_call(thcb, INF, INF, mbuf);
-------------------------------------------------------------------------------
```

[**1**] `l_send0(thcb, msec, mbuf)` は、宛先スレッドに同期メッセージを送る。

- 第 1 引数 thcb は、宛先スレッド。
- 第 2 引数 msec は最大待ち時間である。INF(= -1) を指定すると相手が受信するまで待つ。
- 第 3 引数 mbuf はメッセージバッファを示す、`nil` を設定すると L4 スレッドの “仮想” レジスター MRs が使われる。

【例】「`mbuf = l_putarg(nil, mlabel, "i2s1", e1, e2, &buf, sizeof(buf));`」
は、“仮想” レジスター MRs に引数などが設定される。返り値は第 1 引数の値、この例では nil、である。
「`msgtag = l_send0(thcb, INF, mbuf);`」
は、“仮想” レジスター MRs に設定されたメッセージ (mbuf == nil なので) を thcb に同期メッセージとして送る。

[**2**] `l_recv0(&client, msec, &mbufp);` は、動機メッセージを受信する。

- 同期メッセージを受信したら第 1 引数に送信者の`L4_ThreadId_t` 値 (`L_thcg*` ではない) が代入される。この値は `L_reply0()` で返答を返すときの宛先として使う。
- 第 2 引数は最大待ち時間である。INF(== -1) を指定すると、メッセージが到着するまで待つ。
- 第 3 引数は、受信メッセージの置き場所を示す。 第 3 引数が nil の場合は, メッセージは L4 スレッドの “仮想” レジスター MRs に載せられたままである。第 3 引数が nil でない場合にはメッセージバッファのアドレスが設定される。

【例】`msgtag = l_recv(&client, INF, nil);`
は、“仮想” レジスター MRs に同期メッセージを受け入れる。
`n = l_getarg(nil, "i2s1", &x, &y, &buf2, &sz2);`
は、 受信したメッセージの各引数を指定された変数に設定する。

[**3**] `l_recvfrom0(client, msec, &mbufp)` は、指定された client (`L4_ThreadId_t` タイプであることに注意) からメッセージを受信する。

[**4**] `l_reply0(client, mbuf);` は、指定された client (`L4_ThreadId_t` タイプであることに注意) に返答メッセージを返す。

[**5**] `l_call0(thcb, smsec, rmsec, mbuf)` は、`l_send()` と `l_recvreply()` の組み合わせである。

11 非同期メッセージの送受信

```
---【Signature】-----------------------------------------------------------------
L_msgtag  l_asend0(L_thcb *dest, int msec, L_mbuf * mbuf);     //[1]
     (第 2 引数 msec は、無効果である。l_send0() との対称性のため)
  Ex. mbuf = l_putarg(mbuf, mlabel, "i2s1 ", e1, e2, &buf, sizeof(buf));
      msgtag = l_asend0(dest, 0, mbuf);
```

```
L_msgtag  l_arecv0(L_thcb  *mbox , int msec, L_mbuf **mbuf);    //[2]
```
○ 第 1 引数 mbox が nil の時は、自スレッドが持つメッセージボックスから受信する。\\
非 nil の場合は、mbox が指定するメッセージボックスから受信する。

○ 引数 "msec " は、 INF (-1) または 0.
INF: メッセージが到着するまで待つ。
0: すでにメッセージが到着していなければ、即戻る。

○ 受信メッセージバッファの L_mbuf.sender に、送り主の L4_ThreadId_t が入っている。

```
Ex. L_mbuf *mbuf;
    L4_ThreadId_t  client;
    mbuf = l_arecv0(nil, INF, &mbuf); // 自メッセージボックスから受信
    mbuf = l_arecv0(mbox, INF, &mbuf); // mbox が指すメッセージボックスから受信
    n = l_getarg(mbuf, "i2s1 ", &x, &y, &buf2, &sz2);
    client = mbuf->X.mr1;

L_msgtag  l_areply0(L4_ThreadId_t client, L_mbuf * mbuf);       //[3]
```
（注意）第 1 引数は L4_ThreadId_t である。L_thcb* タイプではない。
```
Ex. mbuf = l_putarg(mbuf, mlabel, ``i1s1'', e1, &buf, sizeof(buf));
    n = l_asend0(client, mbuf);

int  l_arecvreply0(int rectime, L_mbuf **mbuf, int tnum, ...); //[4]
```
○ 非同期返答の受信は、次節で説明する。

```
L_msgtag  l_acall0(L_thcb  *thcb, int recmsec, L_mbuf *mbuf);   //[5]
Ex. rc = l_acall0(dest, INF, INF, mbuf);
```

[**1**] l_asend0(dest, msec, mbuf); は、dest スレッドに非同期メッセージを送る。msec は無視される。mbuf はメッセージバッファである。

【例】 mbuf = l_putarg(mbuf, mlabel, ``i2s1'', e1, e2, &buf, sizeof(buf)); は、mbuf に引数などが設定される。返り値は第 1 引数の値と同じ (この例では mbuf)。
msgtag = l_asend0(dest, 0, mbuf); は、mbuf に設定された内容を dest スレッド に非同期メッセージとして送る。

[**2**] l_arecv0(mbox, msec, &mbufp); は、非同期メッセージの受信である。

- 第 1 引数： スレッドの自前メッセージボックスから受信する場合には、第 1 引数を nil とする。独立メッセージボックスから受信する場合には、第 1 引数にその L_thcb * 値を指定する。
- 第 2 引数： 第 2 引数が (==0) であると、メッセージが到着済みでなければ即 nil を返す。(!=0) の場合は、メッセージが到着するまで待つ。
- 第 3 引数： 受信したメッセージバッファのアドレスが設定される。

- 非同期目セージの送り主の L4_ThreadId_t 値は、受信メッセージバッファの mbuf->sender フィールドに入っている。l_areply() で返答を返すときに使われる。

  【例】msgtag = l_recv(&client, INF, nil);
  は、 L4 スレッド “仮想” レジスター MRs に同期メッセージを受け入れる。
  n = l_getarg(nil, "i2s1", &x, &y, &buf2, &sz2);
  は、受信したメッセージの各引数を指定された変数に設定する。

[**3**] l_areply0(client, mbuf) は、指定された client (L4_ThreadId_t タイプであることに注意) に mbuf が指す返答メッセージを送る。

[**4**] l_arecvreply0(rectime, &mbuf, tnum, ...) は、非同期返答メッセージ (Future message) を受ける。要求メッセージと返答メッセージの対応をとる仕組みあり、.... 非同期返答メッセージ..... にて説明する。

[**5**] l_acall0(dest, recmsec, &mbuf) は、l_asend0() とa_arecv0() の組み合わせである。

12 非同期返答メッセージ

(1) 非同期返答メッセージの返送と受理

```
---【Signature】-----------------------------------------------------------------

L_msgtag  l_areply0(L4_ThreadId_t client, L_mbuf * mbuf);       //[3]
  ○  第 1 引数は、L4_ThreadId_t タイプである。 L_thcb* ではない。
  Ex. mbuf = l_putarg(mbuf, mlabel, ‘‘i1s1’’, e1, &buf, sizeof(buf));
      n = l_asend0(client, mbuf);

int  l_arecvreply0(int rectime, L_mbuf **mbuf, int tnum, ...); //[4]
  ○  第 1 引数：(== 0) 待たない。既に返答が到着していれがそれが返される。
              (!= 0) 返答が届くまで待つ。
  ○  第 2 引数: ここに返答の載ったメッセージのアドレスが代入される。

  ○  第 3 引数：Tally（割符）の数
  ○  第 4... 引数：  Tally のならび。
  ○  返り値：マッチした Tally の番号

L_msgtag  l_acall0(L_thcb  *thcb, int recmsec, L_mbuf *mbuf);   //[5]
  Ex. rc = l_acall0(dest, INF, INF, mbuf);

-------------------------------------------------------------------------------
```

[**1**] 要求メッセージを送る時 (l_asend0(dest, msec, mbuf)) に、要求メッセージの第 1 引数を要求メッセージと返答メッセージの対応をとる tally (割符) として使う。

[**3**] l_areply0(client, mbuf) は、指定された client (L4_ThreadId_t タイプであることに注意) に mbuf が指す返答メッセージを送る。

[**4**] l_arecvreply0(rectime, &mbuf, tnum, ...) は、非同期返答メッセージ (Future message) を受ける。要求メッセージと返答メッセージの対応をとる仕組みとして、tally (割符) が使われる。

- メッセージの mlabel フィールドは tally として使われる。
- Tally の個数は可変である。引数 tnum に tally の個数を設定する。
- (tnum == 0) の場合は、tally チェックは行わず、最初の返答メッセージが選ばれる。
- (tnum != 0) の場合、n 番目の tally がマッチしたら n (n = 1, 2, ...) を返す。

[**5**] `l_acall0(dest, recmsec, &mbuf)` は、`l_asend0()` と`a_arecv0()` の組み合わせである。

(2) 非同期返答の使い方

非同期返答メッセージは、要求メッセージに対応させる必要がある。また、複数のスレッドに要求メッセージを送って、対応をとって適切に返答を受ける必要がある。

「対応」をとるために要求メッセージに tally(割符) という整数データをのせ、非同期返答メッセージ受信自に tally のチェックを行えるようにした。要求メッセージでは、第 1 引数を tally として使う。

非同期返答メッセージを返すスレッドは、メッセージバッファ第 1 ワードの mlabel フィールドに tally 値を設定して、`l_areply()` を実行する。つまり、返答メッセージの mlabael フィールドは tally として使われる。

---【例】----------

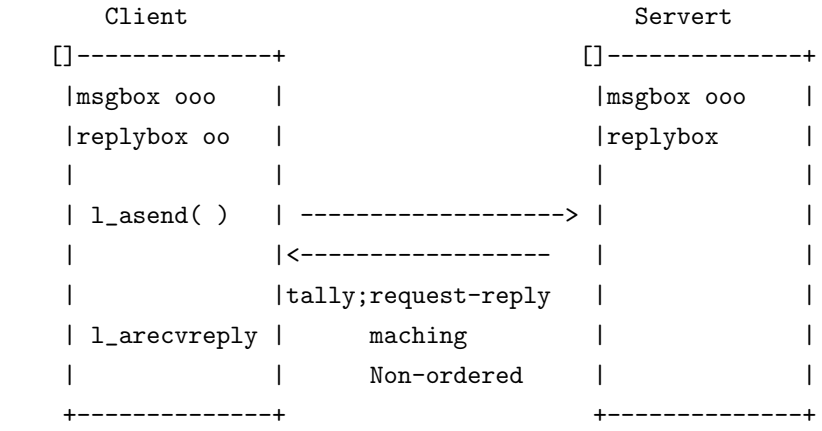
```
L_mbuf *mbuf1, *mbuf2, mbuf3;
l_putarg(mbuf1, label, "i2s2 ", tally1, e2, buf1, len1, buf2, len2);
                                    // tally1 が割符
l_asend(alpha, INF, mbuf1);  //alpha スレッドに非同期メッセージを送る

l_putarg(mbuf2, label, "i2s1 ", tally2, e3, buf3, len3);
                                  // tally2 が割符
l_asend(beta, INF, mbuf2);  //beta スレッドに非同期メッセージを送る
   .....
// 処理を続行し、返答が必要になったら  l_arecvreply() を呼び出す

k = l_arecvreply(INF, &mbuf3, 2, tally1, tally2);
    // tally1 もしくは tally2 を持つ返答を受理する。
    // 非同期メッセージが n 番目の tally を持っていたら、n を返す。
switch(k){
case 1: l_getarg(mbuf, "i2 ", &x, &y);  // tally1
        .....
        break;
case 2: l_getarg(mbuf, "i1s1 ", &z, &ss, &sz); // tally2
        ........
}
.............
```

----

(3) 非同期返答の実装

各スレッドは、非同期返答を受けるための Replybox (`L_thcb.replybox`) を有している。非同期変数は Replybox に挿入される。`l_arecvreply( )` は、tally が一致する返答メッセージが到着しているかをチェックし、有れば情報を返し、無ければ返答メッセージの到着を待つ。

---【実装の仕組み】----------



```
       Client                                Servert
   []--------------+                     []--------------+
    |msgbox ooo    |                      |msgbox ooo    |
    |replybox oo   |                      |replybox      |
    |              |                      |              |
    | l_asend( )   | -------------------> |              |
    |              |<------------------   |              |
    |              |tally;request-reply   |              |
    | l_arecvreply |      maching         |              |
    |              |      Non-ordered     |              |
    +--------------+                      +--------------+

-----------------------------------------------------------------------------
```

13　能動オブジェクトプログラム例

例として、マウスとキーボードからの入力を表示する Window プログラムを考える。Window はスクリーン上に任意個生成できるものとする。マウス、キーボード及び Window は並行動作するので、能動オブジェクト (ActObj) で作る。

以下のプログラム例で、なんとなくイメージが掴んでいただきたい。

```
---【例】----------------------------------------------------------------
enum{MOUSE, KEYBD, WINDOW,  ....};
typedef struct Window   Window;
Window  *w1, *w2, *currentwindow;

//---- マウス ActObj -------------------
typedef struct{
    L4_thcb  _a;
    int  x, y;
    .....
} Mouse;

void mousefunc(Mouse *self)
{
     ...............
      l_putarg(mbuf, MOUSE, "i5", typ, self->x, self->y, dx, dy);
      l_asend0(currentwindow, INF, mbuf);  //Current window にメッセージを送る
      .........
}

Mouse *mouseobj = (Mouse*)malloc(sizeof(Mouse));
l_create_thread(mousefunc, 4096, mouseobj); //マウス ActObj を生成

//--------キーボード  ActObj --------------------
typedef struct{
```

```
    L4_thcb  _a;
    char    line[128]
    .....
} Keyboard;

void keyboardfunc(Keyboard *self )
{
     ..............
      l_putarg(mbuf, KEYBD, "s1", len, self->line, len);
      l_asend0(currentwindow, INF, mbuf); //Current window にメッセージを送る
      ...........
}

Keyboard *keybdobj = (Keyboard*)malloc(sizeof(Keyboard));
l_create_thread(keyboardfunc, 4096, keybdobj); //キーボード ActObj を生成

//---- Window ActObj -------------------
struct Window{
    L4_thcb  _a;
    .....
};

void windowfunc(Window *self)
{
     ..............
    for(;;){
       tag = l_arecv(     mbuf);
       switch(MLABEL(tag)){
       KEYBD:    // キーボード ActObj からメッセージ受信の場合
         {
             char  line[128];
           l_getarg(mbuf, "s1", line, &len);  //
           ........
         }
       MOUSE: // マウス ActObj からメッセージ受信の場合
         { int  c1, x, y, dx, dy;
           l_getarg(mbuf, "i5", &c1, &x, &y, &dx, &dy);
           .......
         }
         .....
       }
    }
}


w1 = (Worker*)malloc(sizeof(Worker));
```

```
l_create_thread(workerfunc, 4096, workerobj); //Window ActObj を生成
```
