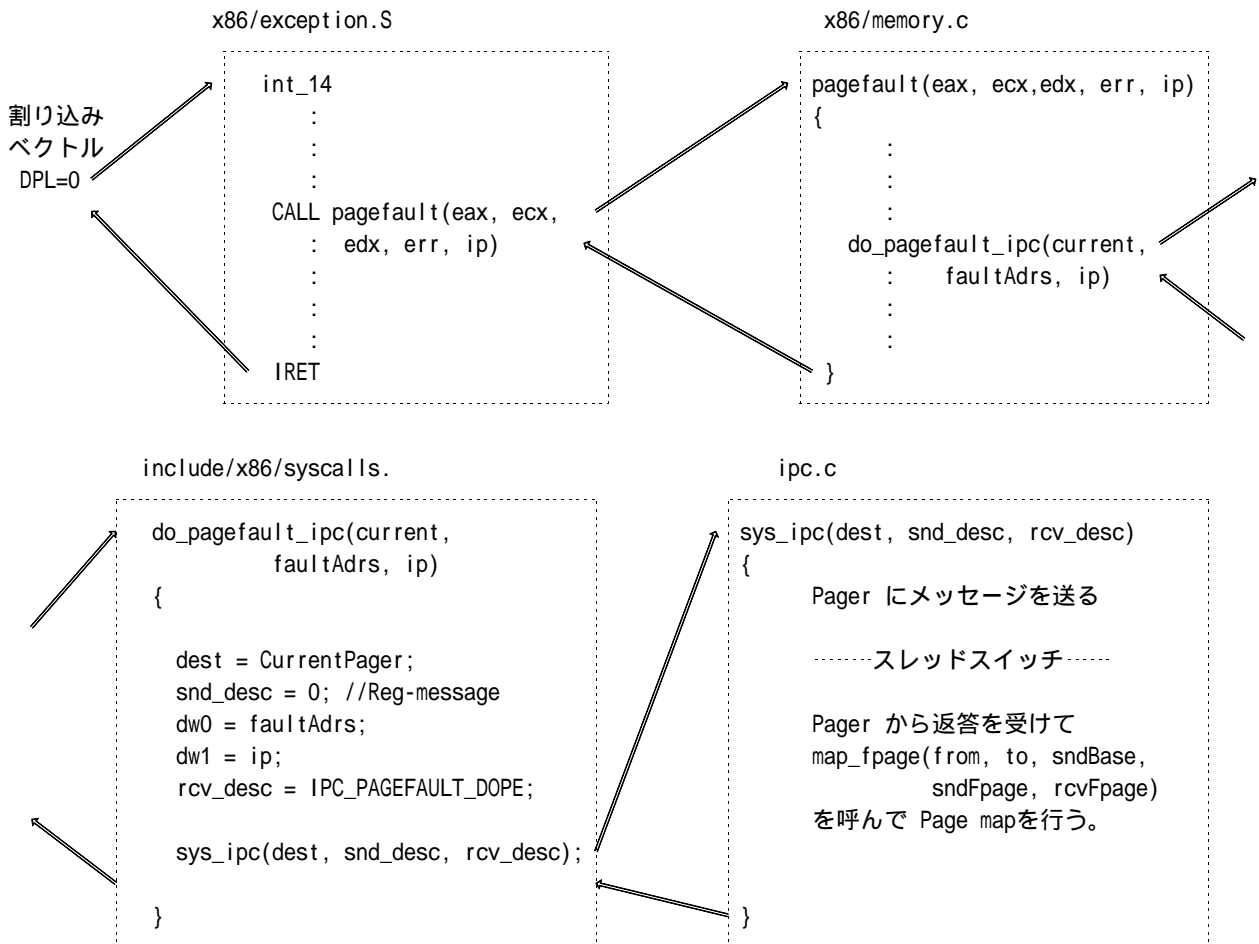


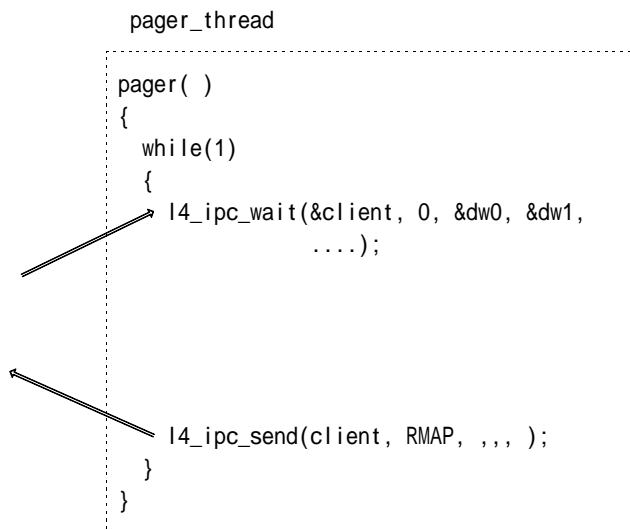
§ 1. L4 の Pagerの仕組み

【Pagefault割り込み発生時の制御フロー】



IPC_PAGEFAULT_DOPE = 0x82 = Fpage(0, 2**32-K)+RMAP

Pagerにメッセージを送るのに伴ってスレッドスイッチが起こり、pager_thread が wakeup される。 pager_threadが system call から戻る際に IRET が実行され、 pager_thread はユーザーモードで実行される。



RMAP = 0x02 = Fpage map を意味する

例えば page fault に対して Pager が 番地のページを map するには、
dw0 = ;
dw1 = | 12 << 2 | 0x2 ;
msg_desc = 0x02;
I4_ipc_send(client, msg_desc, dw0, dw1 ,,);
を実行する。

§ 3. Ix86 の割り込み 例外処理

IDT Gate Descriptor

Segment selector	Offset15..0			
Offset31..16	P	DPL	種別	

種別

- 00101 : Task Gate
- 01110 : Interrupt gate ⇒ EFLAGS.IFを clear
- 01111 : Trap gate

L4-Ka kernel の場合

割り込みベクトル	種別	DPL
0,1, 4~19, 32~47	Interrupt gate	0
20~31	Trap gate	0
3, 48~54	Interrupt gate	3

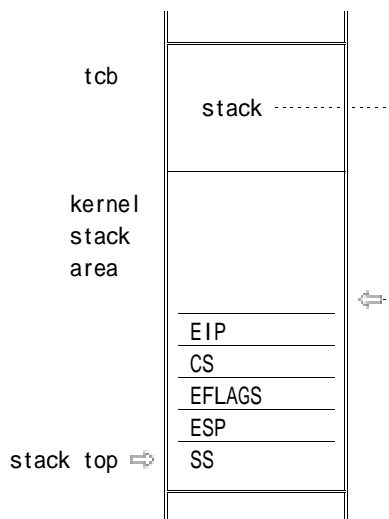
pagefaultは14

スレッド切り換え switch_to_thread (tcb_t * tcb, tcb_t current);

```

inline void switch_to_thread (tcb_t * tcb, tcb_t current)
{
    extern x86_tss_t __tss;
    __tss.esp0 = (dword_t) get_tcb_stack_top(tcb); ⇐ 鍵
    //-----
    dword_t dmy;
    アセンブラコード (
        "pushl %%ebp
        pushl $2f
        movl %%esp, current->stack
        movl tcb->stack, $$esp
        movl %%cr3, ebx
        cmpl tcb->page_dir, ebx
        je 1f
        movl tcb->page_dir, %%cr3
1: ret
        2: popl %%ebp"
    );
}

```



TSS の ESP0、つまり割り込みなどが起きた時にカーネルが使うカーネルスタックは、その時走っているスレッドのカーネルスタックになっている。

スレッド切り換え毎に、そのスレッドのカーネルスタックトップが TSS の ESP0 に設定される。

param
entry
uip
X86_UCS
X86_FLAGS
usp
X86_UDS

sigma0_main(kernel_info_page_t *kip) (I4-ka/apps/sigma0/sigma0.c:93)

```
変数 free_mem = kip -> main_mem_low;
kernel_mem = kip -> main_mem_hihg;
```

page_arrayを初期設定 (Kernel空間, sigma0空間等を予約済にする)

LOOP

```
I4_ipc_wait(&client, 0, &dw0, &dw1, .... );
msg = 2; // Fpage mapを意味する
```

```
IF client がカーネルスレッド
THEN
```

```
IF dw0== 0xFFFFFFFF
THEN // カーネルページの要求
page_arrayテーブルを上位アドレスから探す.
IF 空きページ有り
THEN dw0 = そのアドレス;
dw1 = dw0 | 1-page | Grant;
page_arrayテーブル更新;
//⇒空きページをカーネルページ(4K)としてGrant する。
ELSE msg = dw0 = dw1 = 0;
```

all '1'	00
---------	----

```
FI;
ELSIIF dw0== 0x00000001
THEN
IF dw1 & 0xFF == 0
THEN //
msg=0; dw0=0x100; //Num of pages for kernel use.
//⇒カーネルが使えるページ数を返す。
ELSE
msg = dw0 = dw1 = 0;
```

00 ----- 001	
	0

```
FI;
ELSE
不当メッセージ
FI;
```

```
ELSE // client はユーザスレッド
```

```
IF dw0== 0xFFFFFFFF
THEN // Map a 4K page writeable
page_arrayテーブルを下位アドレスから探す.
IF 空きページ有り
THEN dw0 =そのアドレス;
dw1 = dw0 | 1-page | Map ;
page_array テーブルに client を登録;
// ⇒空きページをカーネルページ(4K)としてMap する。
ELSE msg = dw0 = dw1 = 0;
```

11-----1100	
-------------	--

```
FI;
ELSIIF dw0== 0x00000001
THEN
IF dw1 & 0xFF == 1
THEN // カーネルinfoページ要求
msg = 0; dw0 = 0;
dw1=カーネルinfoページのアドレス | 4K-page | Map ;
//⇒カーネルinfoページ(4K)をMap する。
ELSE
msg = dw0 = dw1 = 0;
```

00 -----001	
	0-01

```
FI;
```

0G<= <=1G	1
-----------	---

```
ELSIIF dw0 == 0 OR 0x00000002 <= dw0 <= 0x40000000
```

	SP
--	----

```

THEN
    IF (dw0 & 0X01) && (dw1 & 0XFF)==SuperPageBit << 2)
    THEN // Map Superpage writable and uncacheble
        adr = dw0 & SUPERPAGE_MASK;
        page_arrayテーブルに adr からSuperpage サイズ分の空きがあるか?
        IF 空きページ有り
            THEN dw0 =adr;
                dw1 = dw0 | SuperPage | Map ;
                page_array テーブルに client を登録;
                //⇒要求された番地にスーパーページ(4M)をMap する。
            ELSE
                msg = dw0 = dw1 = 0;
        FI;
    ELSE
        IF 該当ページが割り付け済
        THEN msg = dw0 = dw1 = 0;
        ELSE page_arrayテーブルに client を登録;
            dw0 &= PAGE_MASK;
            dw1 = dw0 | 4K-page | Map ;
            dw2 = 0;
            // ⇒空きページを要求された番地に(4K)Map する。
        FI
    FI

ELSIF 0x40000004 <= dw0 <= 0xC0000000
THEN // Map 4M Superpage
    dw0 = (dw0 & SUPERPAGE_MASK) + 0X40000000;
    dw1 = dw0 | Superpage | Map;
    //⇒要求値 + 1G番地のスーパーページ(4M)をMap する。
ELSE
    FI;
FI;

I4_ipc_reply(client, msg, dw0, dw1, .... );
ENDLOOP;

```

1G<	<=3G
-----	------

§ 3. 独立した論理空間をつくるための Pager

L 4 マイクロカーネルの上で、独立な論理空間を持つタスクを実現するための Pager の作り方を: ChacmOSの pager を例題として説明する。

§ 3.1 カーネル info ページのマップ

○ Sigma-0 あるいは RMGR.pager から "Kernel info page" をマップする。

(1) I4_ipc_call(rootpager, 0, 0, 0, 0,
 (void *) (INFO_BASE + 0x32), &dw0, &dw1, &dw2,
 L4_IPC_NEVER, &result);

(2) Sigma-0 あるいは RMGR.pager へのメッセージ 『dw0 = 0, wd1 = 0』は、"Kernel info page" 要求を意味する。

(3) Sigma-0 あるいは RMGR.pager は、以下のメッセージを返答する。

dw0 = 0; dw1 = "Kernel info page"のアドレス | 4-Kpage | map;

(4) (1)の recv-descriptor は、以下を意味する。

Adrs=INFO_BASE		size=4KB	10
----------------	--	----------	----

Flex pageを受信する。
送られて来た page を INFO_BASE のページに割り当てる
サイズは 4KB

(5) こうして、INFO_BASE 番地に "Kernel info page" がマップされ、メモリ情報にアクセスできるようになる。

§ 3.2 フリーページリストの構築

○ Sigma-0 あるいは RMGR.pager から、使える全ページを受け取り、フリーリストに組み入れる。

(1) 使える全ページに対してくりかえす
for (i = pagecnt; i >=0; i++)
{

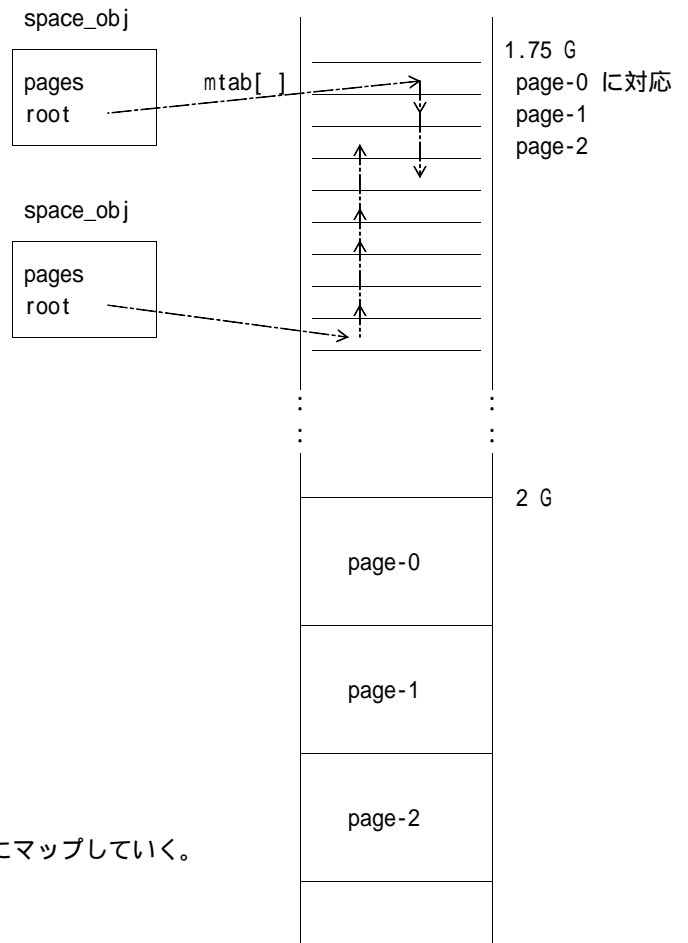
(2) ページを受け取り、マッピングする。

```
l4_ipc_call (rootpager,
             0, i*4096, 0, 0,
             (void*)(mapbase + 0x32),
             &dw0, &dw1, &dw2,
             L4_IPC_NEVER, &result);
:
:
mapbase += 4096;
:
:
```

○ rootpagerは、i*4096 番地のページを引き渡す。

○ IPC は、そのページをクライアントのmapbase 番地にマップしていく。

(3) 右頭のようにフリーリストに登録していく。



§ 3.3 個別論理空間用の Pager の仕組み

個別論理空間用の Pager の動作

```
void pager_thread( )
|
while(1)
{
    Page faultメッセージを受信する
    l4_ipc_receive (&client,
                   (void *) 0x01, &dw0, &dw1, &dw2,
                   L4_IPC_NEVER, &result);
```

recv-descriptor = (void *) 0x01 : 任意のスレッドから受信

注 | dw0: Page faultの生じた番地

カーネル info ページの要求ならば、kernel info page を返す。

```
if (dw0 == 1) { //カーネル info ページの要求
    dw1 = INFO_BASE + 0x30;
}
```

さもなければ、フリーリストから空きページを返す。

```
else {
    ○ フリーリストから、空きページ (ここでは とする) を見つける。

    ○ ページ をクライアントタスクのリストに移す

    ○ dw1 = 8 G + 4096 * + 0x32;
}
```

返答を返す。

```
l4_ipc_send (client, (void *) L4_IPC_SHORT_FPAGE,
             dw0, dw1, dw2,
             L4_IPC_NEVER, &result);
```

```
    ○ IPC は、ページ をクライアントの page faultを生じたページにマッピングする。
} //while
} //pager_thread
```

§ 4 Page fault 側

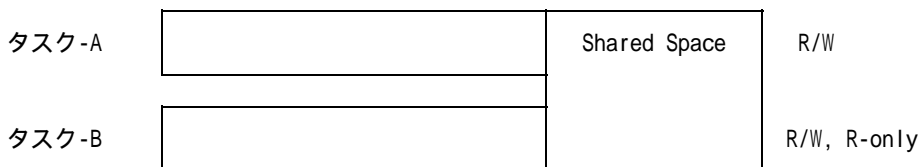
Page faultが発生すると、カーネルは do_pagefault_ipc() を実行して、Pagerにメッセージを送る。

```
send-desc = 0; -- Register Only
dw0 = fault address;
dw1 = ip;
dw2 = 0;
recv-desc = IPC_PAGEFAULT_DOPE = 0x82
```

Fpage(0, 2**32 - KernelSpace);

Fpage	adrs = 0		size=32	10
-------	----------	--	---------	----

§ 4. Shared 空間をサポートするための Pagerの仕組み



○例えば タスク-A と タスク-B が 2Gから 3G の空間を共用する場合、Pagerはその範囲に同一の物理メモリを割り当てればよい。

○アクセス権の制御には、page mapping tables のアクセス権ビットを操作する必要がある。
⇒ Kernel の助けが必要

§ 5. Copy on write の実現と Pager

§ 6 論理空間のクローンと Pager

- UNIX の FORK を実現するには
- ユーザ空間のコピー
- カーネルで管理されているスレッド状態のコピーは ?

○ 例えば Clone元タスクの空間を Share する形で、新しいタスクを生成し、Copy-on-write で必要なメモリを実際に割りつける。

§ 7 物理アドレスの求め方

- DMA では物理アドレスの指定が必要