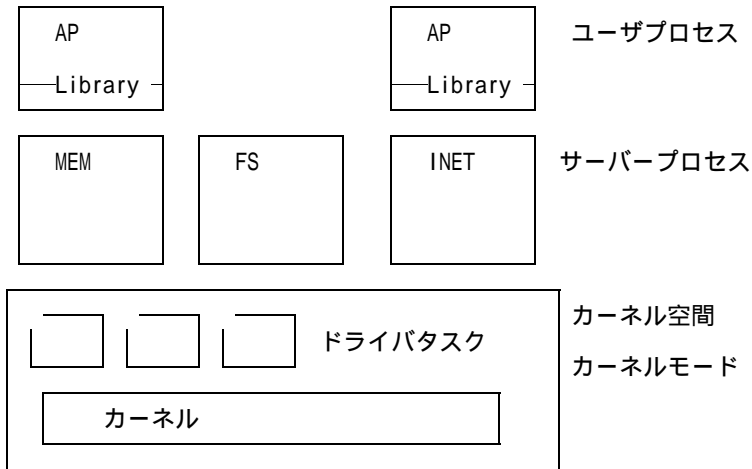


§ サービスタスク構成

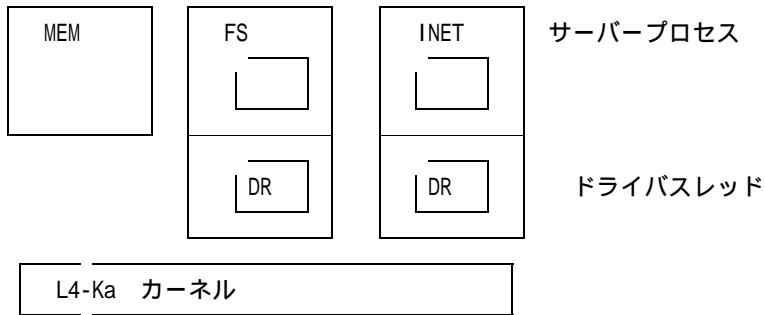
【Minix】



サーバープロセスは、ユーザモードで実行される。

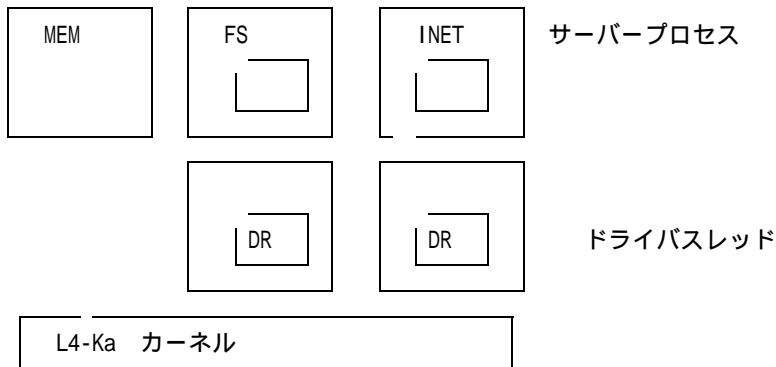
サーバープロセスには、I/O 空間のアクセス権を与えておく

【L4-Minix-1】



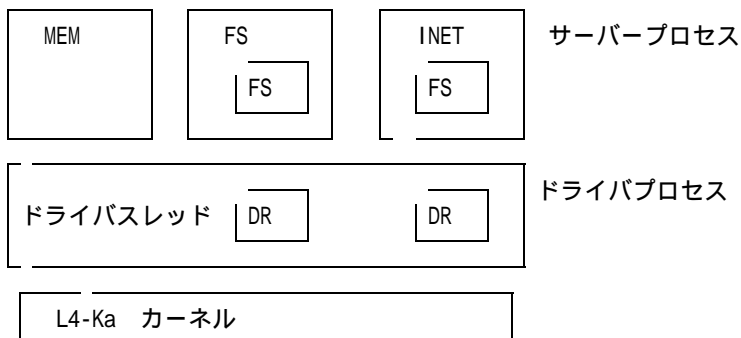
○割り込みハンドラのみは、割り込みモードで実行される

【L4-Minix-2】



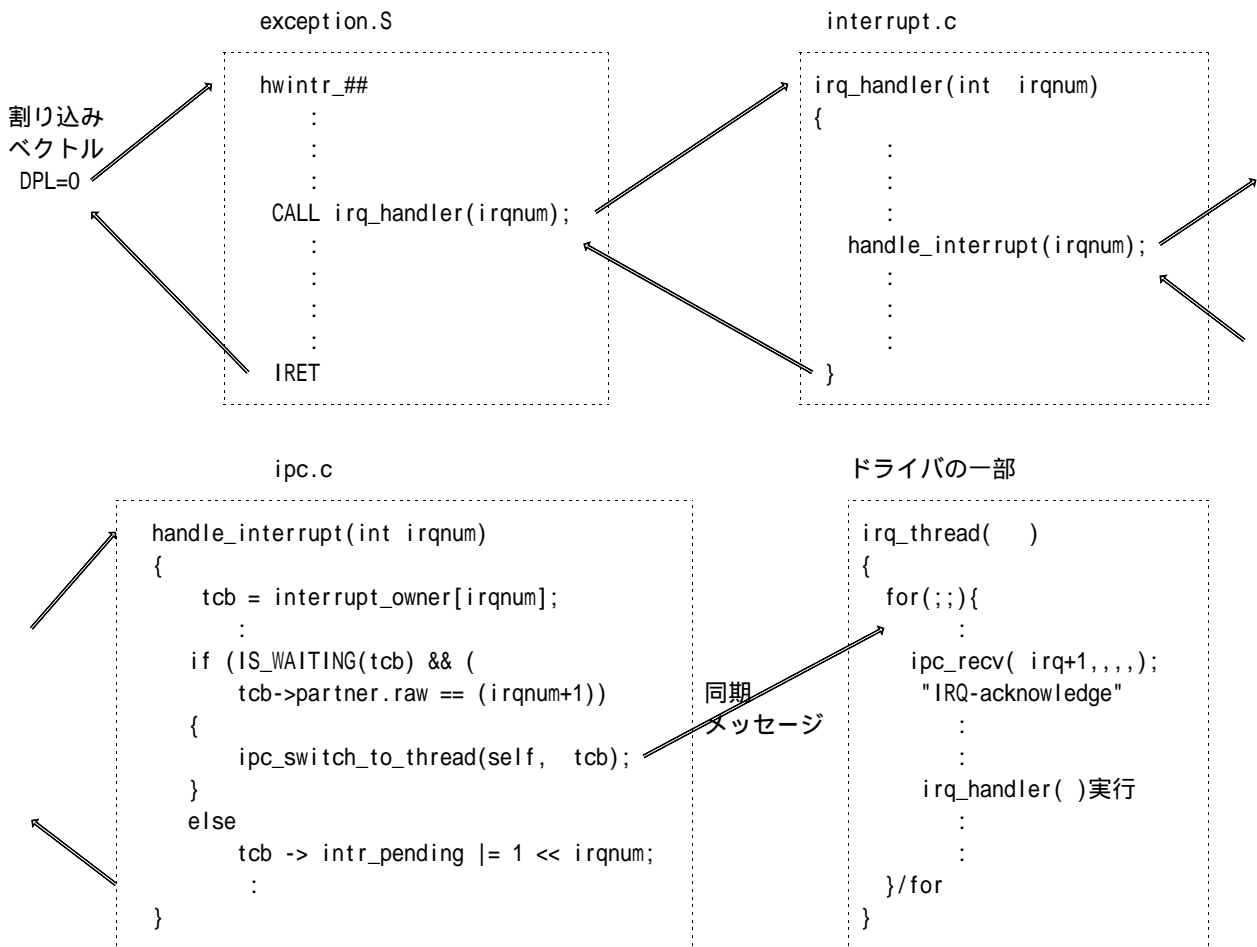
○割り込みハンドラのみは、割り込みモードで実行される

【L4-Minix-3】



§ 割り込みハンドリング

【割り込み発生時の制御フロー】



【IRQ SHARING】

- 同一の IRQ番号を複数のデバイスが共用する。
- PCIバス
- Linux の実現法

【IRQ_THREADと割り込みハンドラの置き場所】

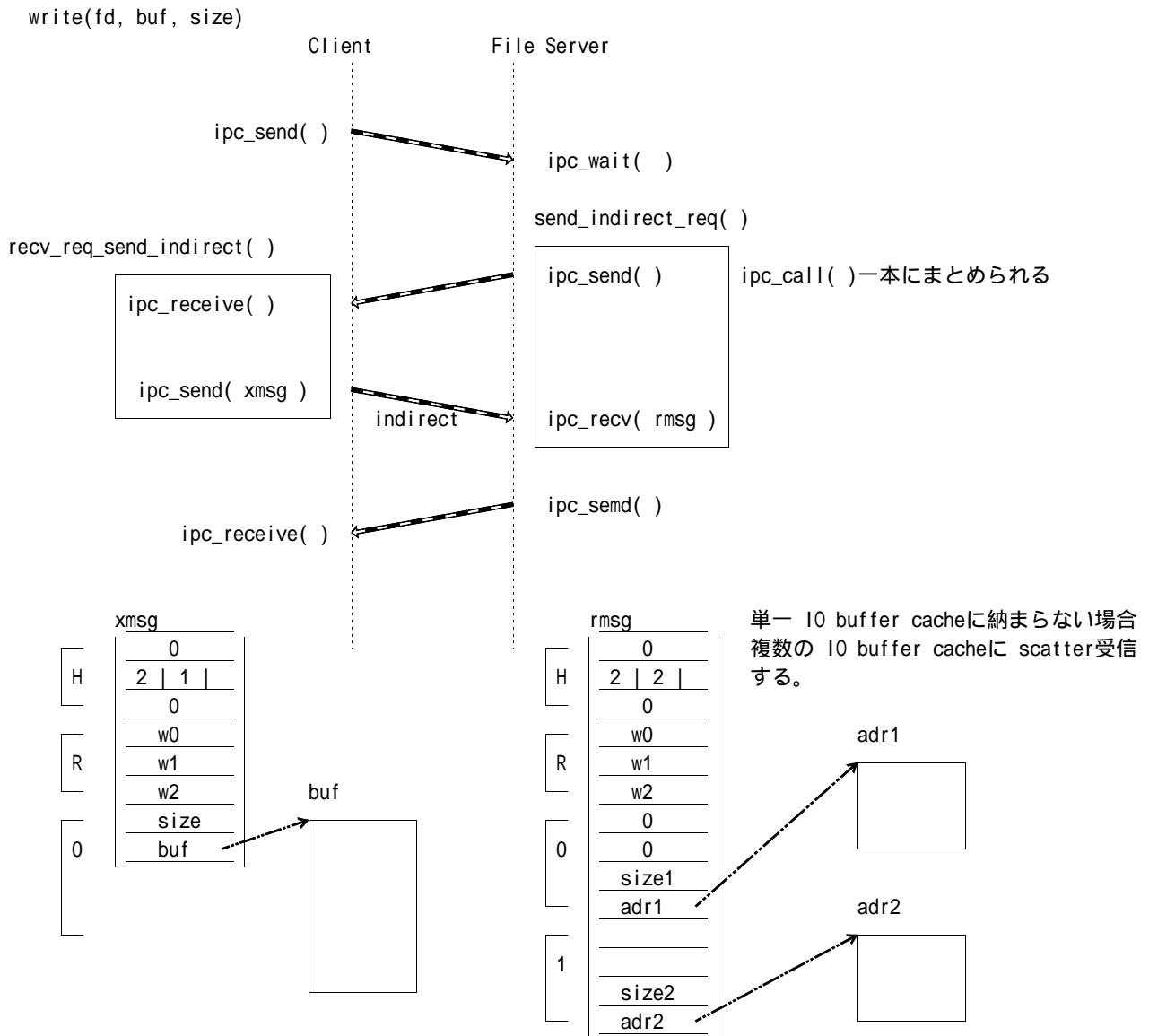
- (1) DRIVER タスク内
- (2) カーネル内

§2 WRITE()のしくみ: 別論理空間のバッファコピー

【MINIXの場合】

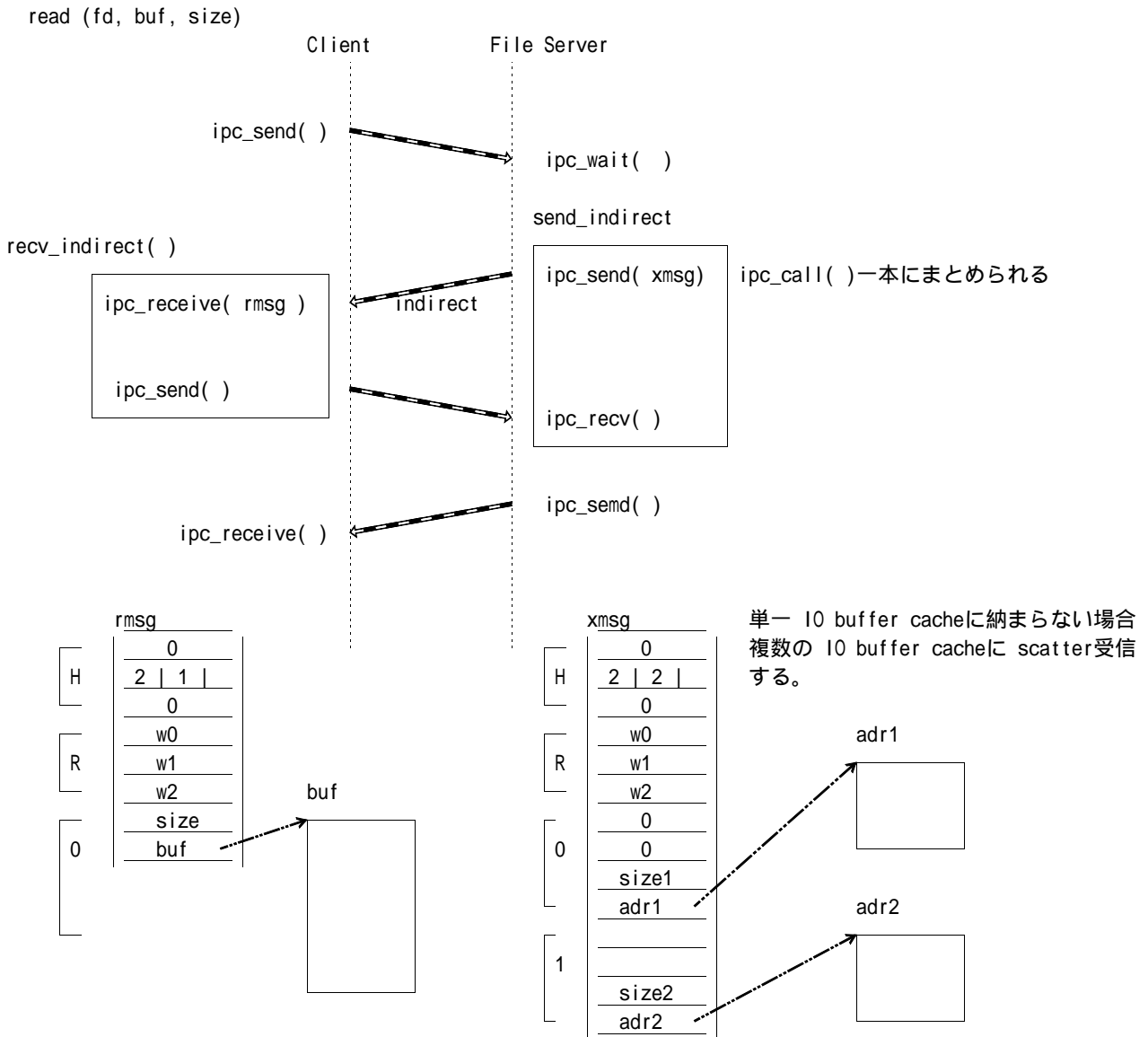
- MINIXでは、データコピーはカーネルプリミティブ `sys_copy()` を使う。
- MINIXのカーネルは、論理アドレスから物理アドレスに変換して、物理アドレス間でコピーを行う。
 ⇨ MINIX のメモリ管理は Paging も Segmentも使っていないので、物理アドレスコピーが簡単に実現できる。

【L4-Minix の場合】



- ユーザ空間と(カーネル外)ファイルサーバーの間のデータ転送は、重要な課題である。
- L4-IPC の Scatter-gather 型データコピーは、効果的である。
- 実装 ⇨ 評価データ ⇨ 論文

§3 READ() 別論理空間のバッファコピー



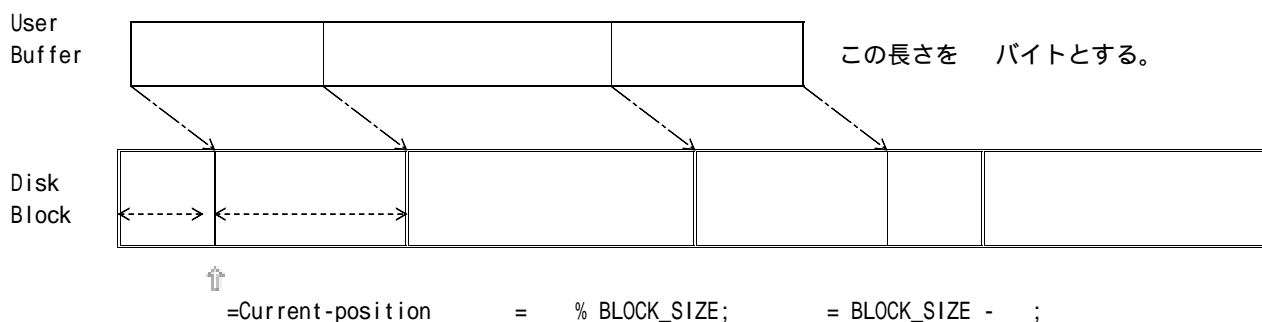
【ソースプログラム】

katsura:/home/maruyama/l4-ka/

katsura:/home/maruyama/l4-ka/minix/src/L4/L4-if.c
/L4/irq-if.c
/driver/*
/fs/*

§ 4 Scatter/gatherの具体化

```
r = read(fd, buffer, nbytes);
r = write(fd, buffer, nbytes);
```



○ send/recv-descriptor の アドレスとサイズ情報

```
= IF <
```

```
THEN
```

```
Current Block: [ CurrentBuf +      ]
```

```
ELSE 第一回目:      バイトをコバイトをコピーする。
```

```
Current Block [ CurrentBuf +      ]
```

```
Next block [ NextBuf + 0      BLOCK_SIZE ]
```

```
Next block [ NextBuf + 0      BLOCK_SIZE ]
```

```
Last block [ NextBuf + 0      残り ]
```

```
回 = (      -      -1)/BLOCK_SIZE + 1
```

```
残り =      -      -      *BLOCK_SIZE
```

```
FI;
```

Minix source program の関連箇所

○ minix/src/fs/read.c の中の

```
read_write(int rw_flag)
```

```
rw_chunk(rip, position, offset, left, rw_flag, buff, seg, usr);
```

§5 FS と INET 間のインタフェース

Cf. minix/src/fs/device.c

デバイス番号から該当 function を求める。

○ struct dmap { dmap_t dmap_open, dmap_rw, dmap_close; int dmap_task} dmap[] =

major
dev num
で index

| | | | |
|-------------|--------------|-------------|--------------|
| net_open() | call_task() | dev_opcl() | INET_PROC_NR |
| net_open() | call_task() | dev_opcl() | INET_PROC_NR |

ここに: void (*dmap_t)(int task, message *msgptr);

デバイスへの実際の書き込み 読み出し作業

```
int dev_io(int operationp, // DEV_READ, DEV_WRITE, DEV_IOCTL,,
           int non_block,
           dev_t device,
           off_t offset,
           off_t position,
           int bytes,
           int proc,
           char *buf)
{
    major = deviceのメジャー番号;
    task = dmap[major].dmap_task;
    メッセージを編集;
    (*dmap[major].dmap_rw)(task, &メッセージ);
    .....
}
```

CALL_TASK()

```
void call_task(int task_nr, //device taskの番号
              message * msgptr)
{
    task_nrで指定されたデバイスタスクにメッセージを送り、応えを受ける;
}
```

デバイスドライバタスク