

オペレーティングシステム

第10章 メモリ管理の実装例

メモリ管理の実装例

TacOS のメモリ管理プログラムを実装例とする.

- 可変区画方式
- ファーストフィット
- OS がユーザプロセス領域の割当てに使用
- プロセスがヒープ領域を管理するプログラムも同じアルゴリズム

データ構造の初期化

```
#define MBSIZE sizeof(MemBlk) // MemBlk のバイト数
#define MAGIC (memPool) // 番兵のアドレスを使用する

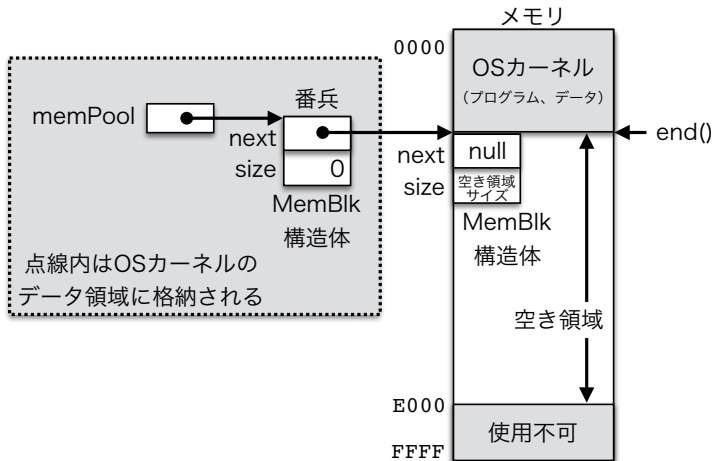
// 空き領域はリストにして管理される
struct MemBlk { // 空き領域管理用の構造体
    MemBlk next; // 次の空き領域アドレス
    int size; // 空き領域サイズ
};

// メモリ管理の初期化
MemBlk memPool = {null, 0}; // 空き領域リストの番兵
public int _end(); // カーネルの BBS 領域の最後

void mmInit() { // プログラム起動前の初期化
    memPool.next = _ItoA(addrrof(_end)); // 空き領域
    memPool.next.size = 0xe000 - addrrof(_end); // 空きメモリサイズ
    memPool.next.next = null;
}
```

- `memInit()` はカーネル起動時に一度だけ実行される。
- 番兵付きのリストで管理する。

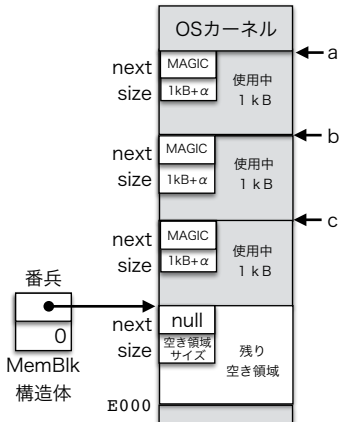
初期化直後のデータ構造



- `_end()` はカーネルサイズにより決まる.
- **E000** より後ろはビデオメモリや IPL ROM がある.

メモリの割付け

右のプログラムで a, b, c を割付けたときのデータ構造



```
a = mmAlloc( 1024 ); // 1KiB の領域を割り付ける  
b = mmAlloc( 1024 ); // 1KiB の領域を割り付ける  
c = mmAlloc( 1024 ); // 1KiB の領域を割り付ける
```

メモリの割付けプログラム

```
// メモ리를割り付ける
int mmAlloc(int siz) {
    int s = (siz + MBSIZE + 1) & ~1;
    MemBlk p = memPool;
    MemBlk m = p.next;

    while (_uCmp(m.size,s)<0) {
        p = m;
        m = m.next;
        if (m==null) return 0;
    }

    if (_uCmp(m.size ,s+MBSIZE+2)<=0) {
        if (memPool.next==m && m.next==null)
            return 0;
        p.next = m.next;
    } else {
        MemBlk n = _addrAdd(m, s);
        n.next = m.next;
        n.size = m.size - s;
        p.next = n;
        m.size = s;
    }
    m.next = MAGIC;
    return _AtoI(_addrAdd(m, MBSIZE));
}
```

```
// 메모리割り当て
// 制御データ分大きい偶数に
// 直前の領域
// 対象となる領域

// 領域が小さい間
// リストを手繰る

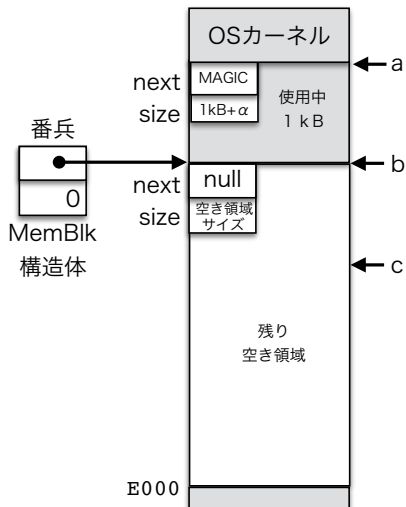
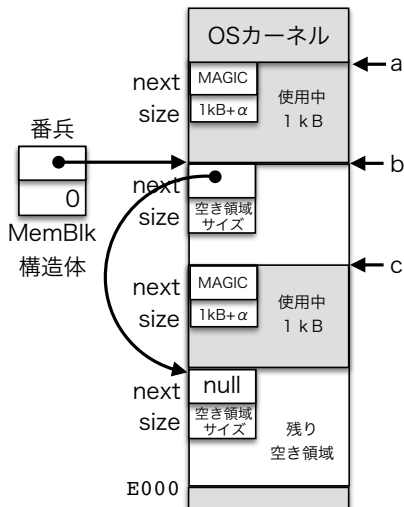
// 메모리가不足する場合は
// 에러を表す null 포인터

// 分割する値がない領域サイズ
// 리스트の長さがゼロにならない
//   ようにする
// リストから外す
// 領域を分割する値がある
// 残り領域

//マジックナンバー格納
// 管理領域を除いて返す
```

領域の解放

b, cを開放したときのデータ構造



メモリの解放プログラム（前半）

```
// メモリを解放する
int mmFree(void[] mem) {
    MemBlk q  = _addrAdd(mem, -MBSIZE);
    MemBlk p = memPool;
    MemBlk m = p.next;

    if (q.next!=MAGIC)
        badaddr();

    while (_aCmp(m, q)<0) {
        p = m;
        m = m.next;
        if (m==null) break;
    }
}
```

// 領域解放
// 解放する領域
// 直前の空き領域
// 直後の空き領域

// 領域マジックナンバー確認

// 解放する領域の位置を探る

- 領域の本当の先頭アドレスを計算する.
- MAGICを確認する.
- 空き領域リストを辿り，挿入位置を決める.

メモリの解放プログラム

```
void[] ql = _addrAdd(q, q.size); // 解放する領域の最後
void[] pl = _addrAdd(p, p.size); // 直前の領域の最後

if (_aCmp(q,pl)<0 || m!=null&&_aCmp(m,ql)<0) // 未割り当て領域では？
    badaddr();

if (pl==q) { // 直前の領域に隣接している
    p.size = p.size + q.size;
    if (ql==m) { // 直後の領域とも隣接してる
        p.size = p.size + m.size;
        p.next = m.next;
    }
} else if (ql==m) { // 直後の領域に隣接している
    q.size = q.size + m.size;
    q.next = m.next;
    p.next = q;
} else {
    p.next = q;
    q.next = m;
}
return 0;
}
```