

システムプログラム 第6回

創域理工学部 情報計算科学科

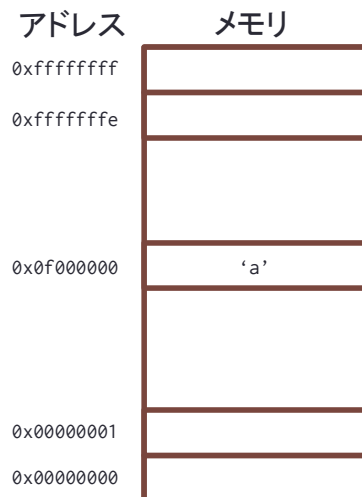
松澤 智史

本日の内容

- システムプログラムの一部であるOSについて学ぶ
 - OSの機能の一つである「メモリ管理」を知ることによってプロセスからメモリが抽象化されていることを学ぶ

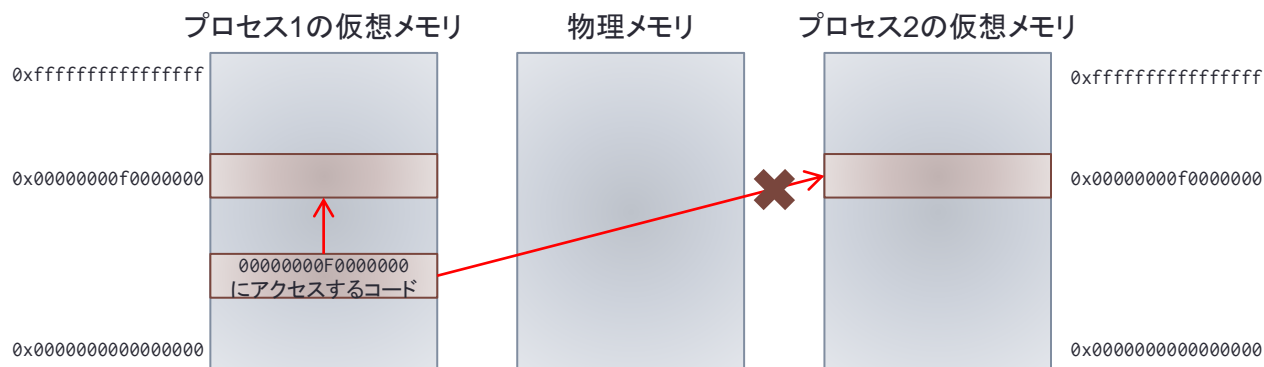
計算機のメモリ

- 1バイトごとにアドレスが割り当てられている
- 扱える上限がある
 - 32ビットのOSの場合は, $2^{32} \div \text{約40億バイト} \div 4\text{GB}$
 - 64ビットのOS(とCPU)の場合は 2^{64} バイト



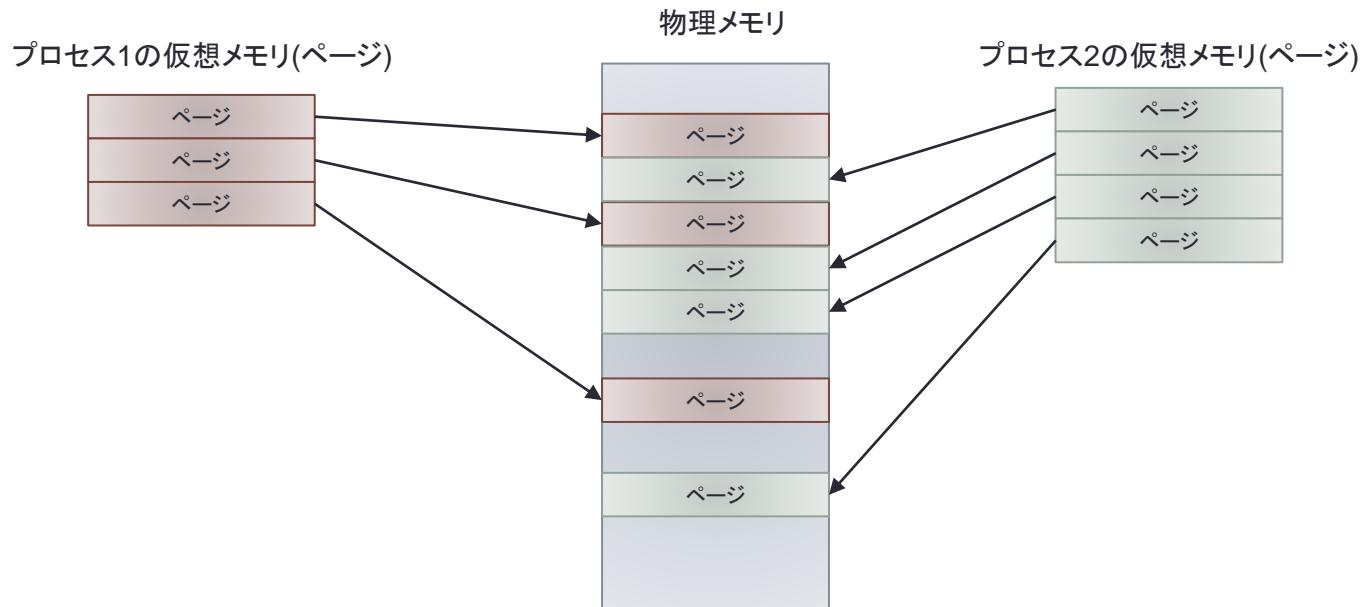
仮想メモリと仮想アドレス空間

- カーネルは複数のプロセスを稼働する
- カーネルは各プロセスに独立したメモリ空間を提供する
 - 独立したメモリ空間は**仮想メモリ**と呼ばれる
 - 仮想メモリには**仮想アドレス空間**が割り当てられる
 - 仮想アドレスは物理メモリのアドレスとは関係ない



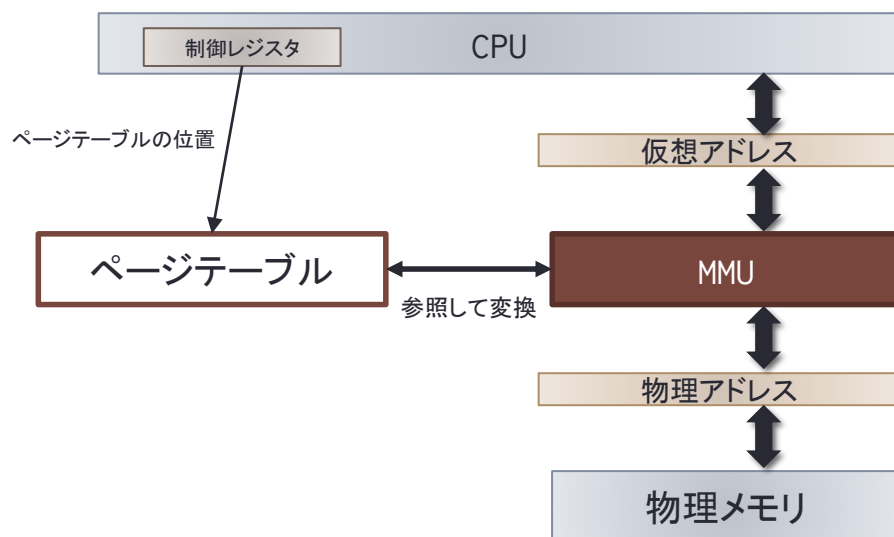
ページング

- 仮想メモリと物理メモリのアドレスのマッピング方式の一つ
 - Linuxで採用されている
- メモリ領域を小さな固定長のページの集合とし、ページ単位で管理



Memory Management Unit(MMU)

- 仮想アドレスを物理アドレスに変換する
- CPUにハードウェアとして内蔵されている
- メインメモリ上に配置するページテーブルに仮想⇔物理の対応表を保持する
 - 高速化のためのTLB(Translation Lookaside Buffer)キャッシュもある



TLB

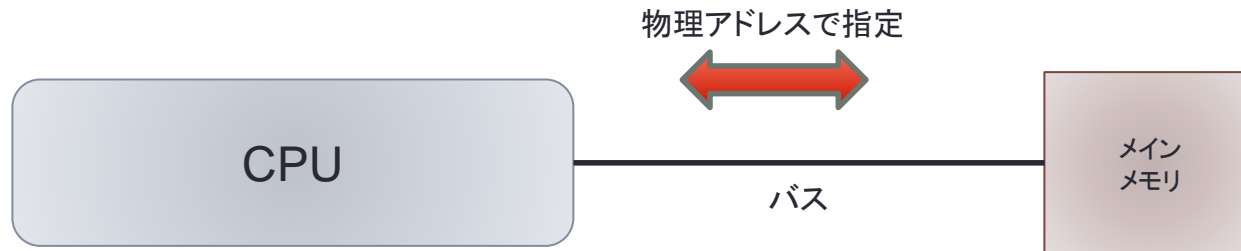
仮想アドレスページ	物理アドレスページ
0x03FF00	0x3FEE00

仮想アドレスをキーとして物理アドレスを得るハッシュテーブルのようなもの

TLBの場所はCPUによって異なる

仮想メモリの有効範囲

- 仮想メモリ(仮想アドレス)が有効なのはCPU内のみ
- バスに流れるメモリアドレスは物理アドレス
- **ユーザープログラムは仮想アドレスのみ**
 - カーネルモードのプログラムのみ物理アドレスを直接扱える
 - CPUはカーネルモードとユーザモードの2種類をサポート
 - カーネルモードのプログラム
 - カーネル本体
 - デバイスドライバ



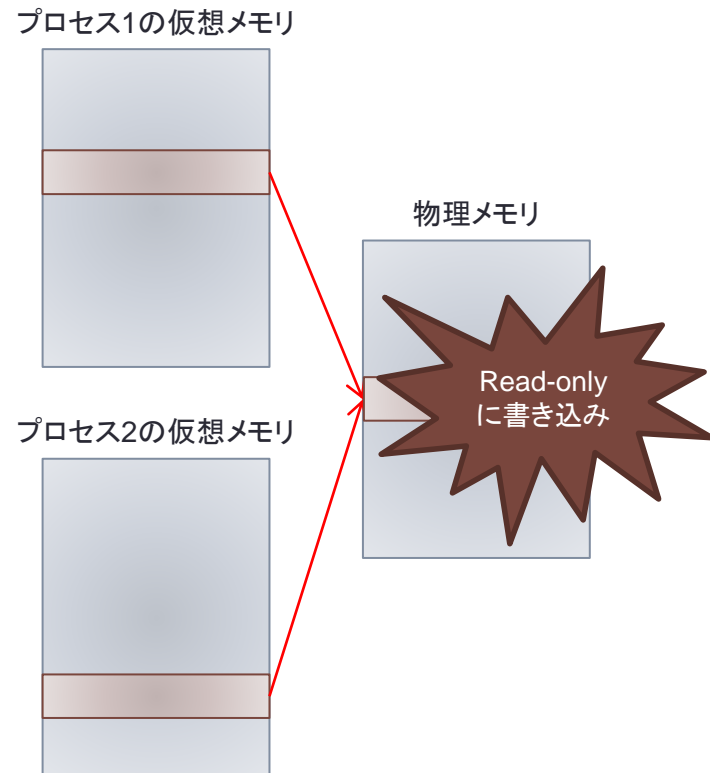
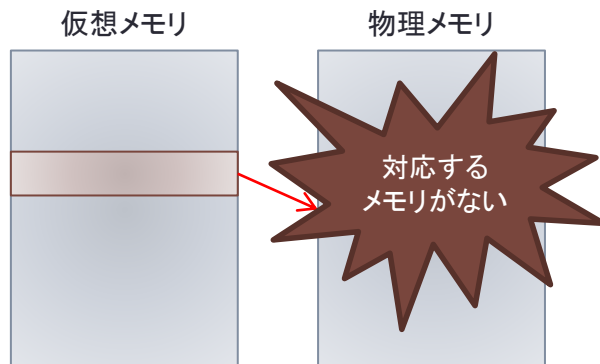
物理メモリの割り当て

- 仮想メモリは使用される際に物理メモリに割り当てられる
- デマンドページング方式と呼ばれる
- 二段階割り当て
 - 仮想メモリの割り当て(mmapシステムコール等) ※管理表に登録するのみ
 - 実際にアクセスがあった際に物理メモリが割り当てられる
- アクセスのない仮想メモリは物理メモリが割り当てられない

仮想メモリサイズ \geq 必要な物理メモリサイズ

余談: ページフォルト

- 物理メモリが割り当てられていない場所をプロセスがアクセスした時に発生する
 - 正しいアドレスを最初にアクセスした場合
 - ページテーブルがなければ作る
 - 正しいアドレスを2回目以降にアクセスした場合
 - ページインの処理を行う(後述)
 - 不正なアドレスをアクセスした場合
 - エラーでプロセス終了



スワッピング (swap)

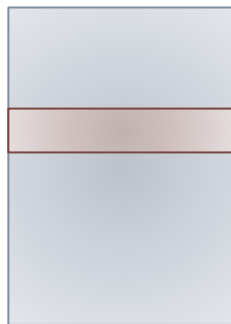
- スワップアウト

- 物理メモリが足りなくなった場合にメモリの内容をHDDやSSDに書き出し(スワップアウト)してメモリ不足を補う
- スワップアウトするディスクの領域を仮想メモリと呼んだりする(紛らわしい)

- ページアウト

- 仮想記憶のメモリ管理単位であるページがディスクに退避される
- スワップアウトの一種
- ディスクから戻す処理をページインと呼ぶ

プロセス1の仮想メモリ



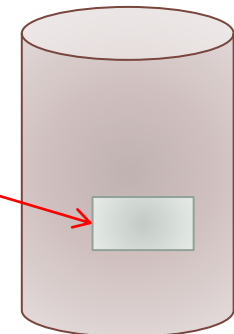
プロセス2の仮想メモリ



物理メモリ



HDD/SSD

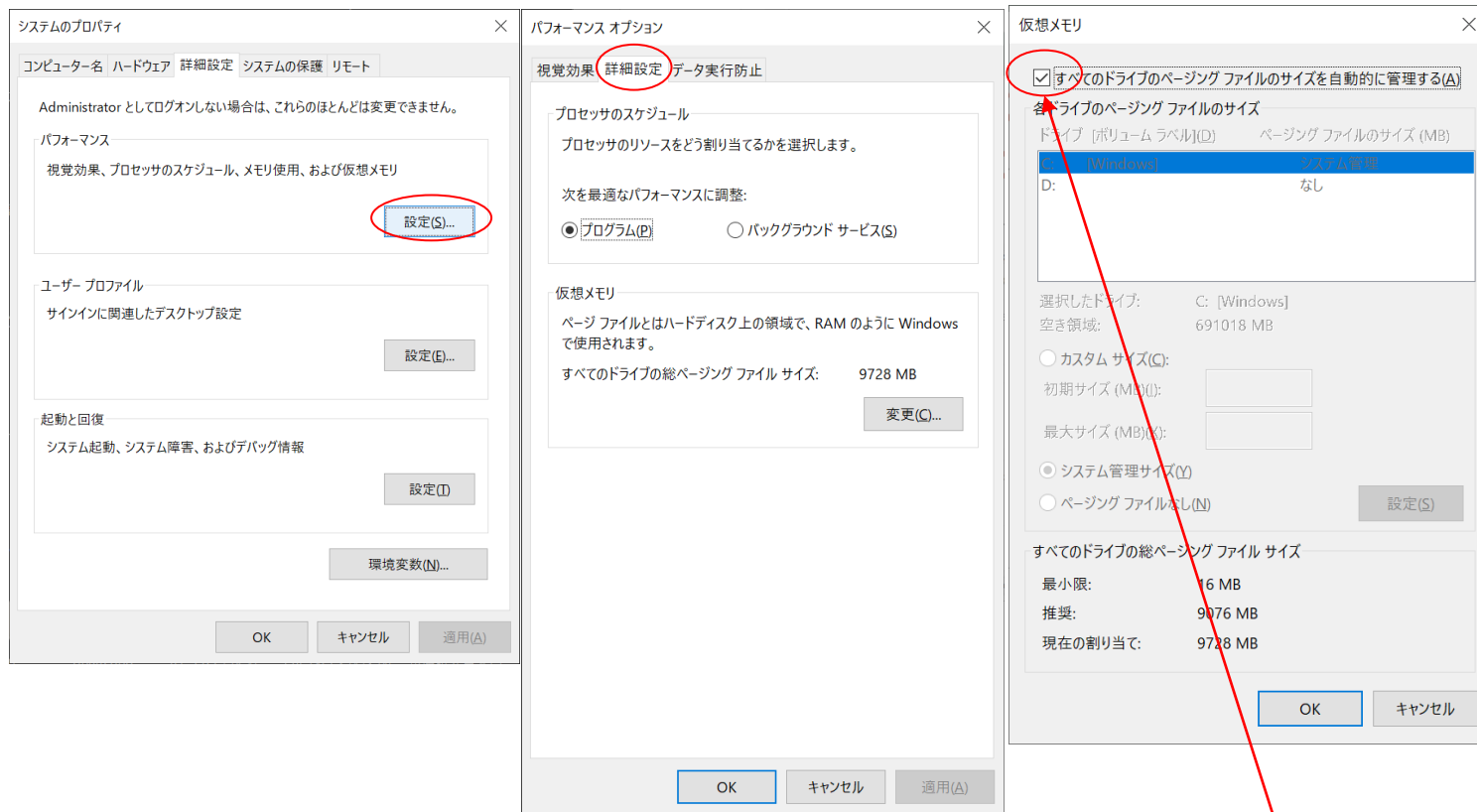


退避

HDD/SSD上にメモリの内容を退避させる領域をswapと呼ぶ

```
tusedls01$ free
              total        used        free      shared  buff/cache   available
Mem:           20559388      425028     19647628       12216       486732     19780668
Swap:           3354620           0       3354620
```

余談: スワップ領域の設定(windows)



このチェックを外すとカスタマイズ可能

仮想アドレス空間の内訳

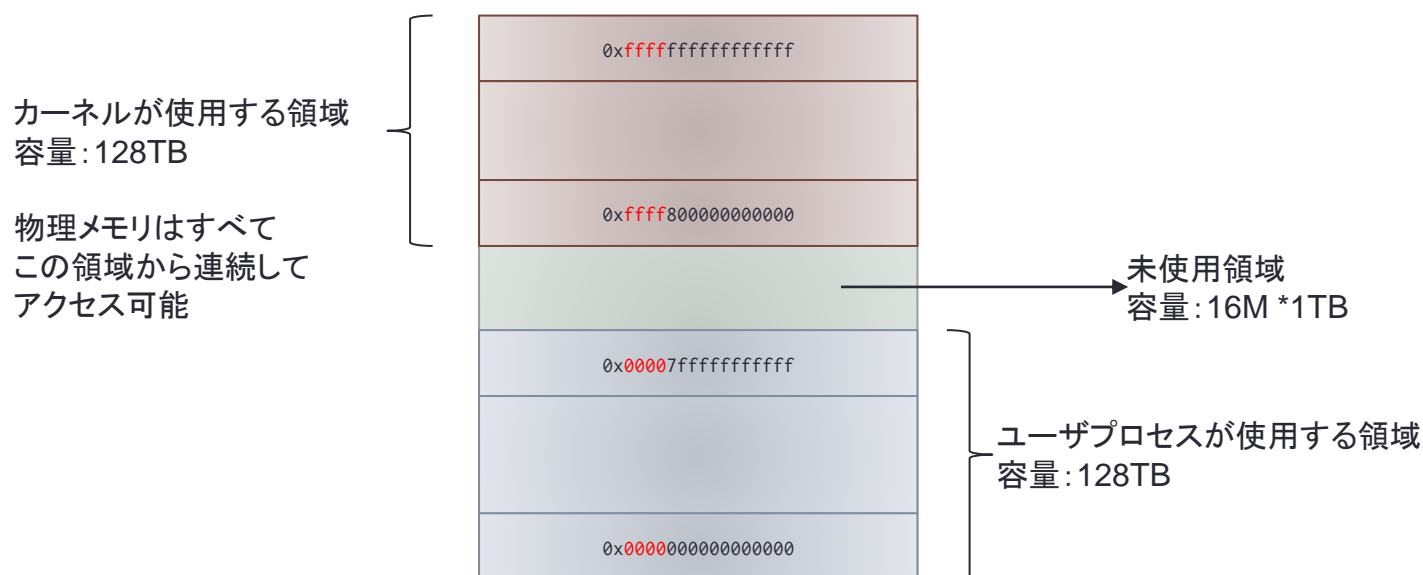
32ビット OSの場合

- 仮想メモリはプロセスごとに異なるが
カーネルが使用する領域は全仮想メモリで共通となる



仮想アドレス空間の内訳

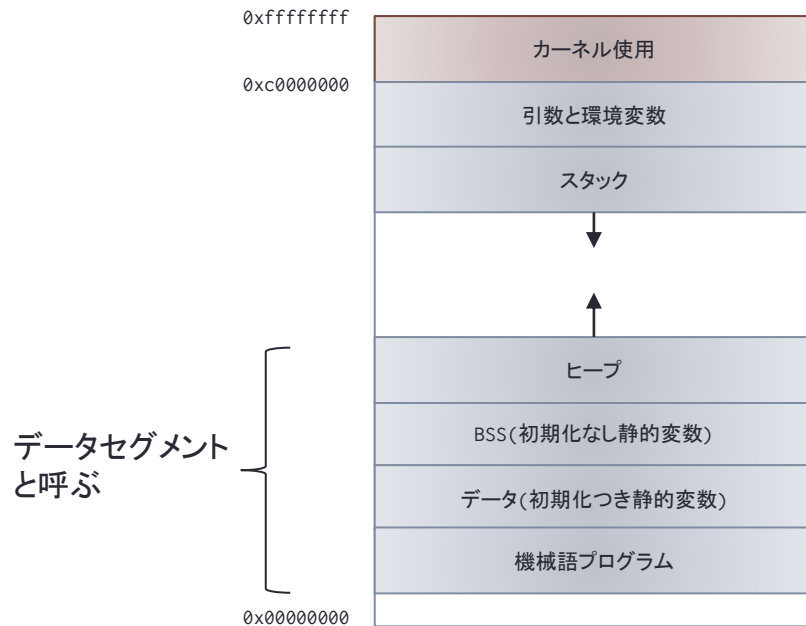
64ビット OSの場合



赤字の部分省くと連続したアドレス空間になる
(実質48ビットの仮想アドレス空間) ※56ビットのケースもある

参考: https://www.kernel.org/doc/html/latest/x86/x86_64/mm.html

仮想アドレス空間の基本的構造



```
tusedls01$ cat address.c
#include <stdio.h>
#include <stdlib.h>

void func(int i) {
    printf("func:&i = %p\n", &i);
}

int global = 5;

int main(){
    int c;
    printf("main:&c = %p\n",&c);
    printf("&global = %p\n",&global);
    int *p;
    p = malloc(sizeof(int));
    printf("main:p = %p\n",p);

    func(c);
    printf("&main = %p\n",&main);
    printf("&func = %p\n",&func);
}

tusedls01$ gcc address.c -o address
tusedls01$ ./address
main:&c = 0x7ffe0abd5634
&global = 0x601034
main:p = 0x148b010
func:&i = 0x7ffe0abd561c
&main = 0x400580
&func = 0x40055d
tusedls01$
```

大きく分けてプログラム、データ、スタックの3つに分割
プログラムは機械語命令を入れる(読み込み専用)

データ:大域変数, 静的変数, malloc() 等で確保したデータを置く さらに3つに分割

- データ:初期値付きの変数
- BSS:初期値なしの変数 OSが自動的に0に初期化
- ヒープ:malloc() で確保

スタックには、関数の局所変数(auto変数、static が付かないもの), 関数の引数 が置かれる
スタックの底には引数と環境変数が置かれる (高い番地から低い番地へ伸びる)

ヒープは、低い番地から高い番地へ伸びる

スタック領域とヒープ領域

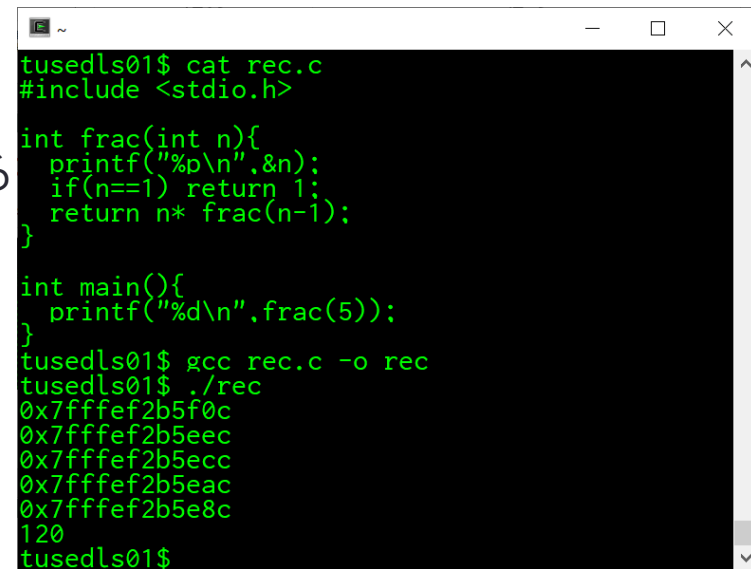
- どちらも動的に割り当てと解放が可能なメモリ領域

- スタック領域

- スタック領域は、確保した順番と逆の順番で解放する
- 管理は易しいが、任意に確保と解放を決められない
- プログラムには「入れ子構造」が多く使われ、入れ子の内側ほどデータ保持の期間が短くなるためスタック領域との相性が良い

- ヒープ領域

- ヒープ(山積み)領域には順序がない
- どのような順序で確保・解放するかは、ソフトウェア側で自由に決められる
- 全体の管理は難しいため、性能的に遅くなることがある



```
tusedls01$ cat rec.c
#include <stdio.h>

int frac(int n){
    printf("%p\n",&n);
    if(n==1) return 1;
    return n* frac(n-1);
}

int main(){
    printf("%d\n",frac(5));
}
tusedls01$ gcc rec.c -o rec
tusedls01$ ./rec
0x7ffffef2b5f0c
0x7ffffef2b5eec
0x7ffffef2b5ecc
0x7ffffef2b5eac
0x7ffffef2b5e8c
120
tusedls01$
```

余談: 実際のカーネルソース

`/usr/src/kernels/3.10.0-957.1.3.el7.x86_64/include/linux/sched.h`

```
struct task_struct {  
    volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */  
    void *stack;  
    atomic_t usage;  
    unsigned int flags;    /* per process flags, defined below */
```

省略

```
    struct mm_struct *mm, *active_mm; ← メモリの構造体
```

`/usr/src/kernels/3.10.0-957.1.3.el7.x86_64/include/linux/mm_types.h`

```
struct mm_struct {  
    struct vm_area_struct * mmap;    /* list of VMAs */  
    struct rb_root mm_rb;  
    struct vm_area_struct * mmap_cache;    /* last find_vma result */
```

省略

- start_code, end_code: テキスト・セグメントの開始番地と終了番地
- start_brk, brk: ヒープの開始番地と終了番地
- start_stack: スタックの開始番地
- arg_start, arg_end: 引数の開始番地と終了番地
- env_start, env_end: 環境変数の開始番地と終了番地

プロセスごとの仮想アドレス空間

/proc/プロセスID/maps

```
tusedls17$ ps
  PID TTY          TIME CMD
 21395 pts/1    00:00:00 bash
 25057 pts/1    00:00:00 ps
tusedls17$ more /proc/21395/maps
00400000-004dd000 r-xp 00000000 fd:00 50336697      /usr/bin/bash
006dd000-006de000 r--p 000dd000 fd:00 50336697      /usr/bin/bash
006de000-006e7000 rw-p 000de000 fd:00 50336697      /usr/bin/bash
006e7000-006ed000 rw-p 00000000 00:00 0
02520000-026e3000 rw-p 00000000 00:00 0
7f8db9c0d000-7f8dc0137000 r--p 00000000 fd:00 33720336      [heap]
7f8dc0137000-7f8dc02f9000 r-xp 00000000 fd:00 79389      /usr/lib/locale/locale-archive
7f8dc02f9000-7f8dc04f9000 ---p 001c2000 fd:00 79389      /usr/lib64/libc-2.17.so
7f8dc04f9000-7f8dc04fd000 r--p 001c2000 fd:00 79389      /usr/lib64/libc-2.17.so
7f8dc04fd000-7f8dc04ff000 rw-p 001c6000 fd:00 79389      /usr/lib64/libc-2.17.so
7f8dc04ff000-7f8dc0504000 rw-p 00000000 00:00 0
7f8dc0504000-7f8dc0506000 r-xp 00000000 fd:00 79395      /usr/lib64/libdl-2.17.so
7f8dc0506000-7f8dc0706000 ---p 00002000 fd:00 79395      /usr/lib64/libdl-2.17.so
7f8dc0706000-7f8dc0707000 r--p 00002000 fd:00 79395      /usr/lib64/libdl-2.17.so
7f8dc0707000-7f8dc0708000 rw-p 00003000 fd:00 79395      /usr/lib64/libdl-2.17.so
7f8dc0708000-7f8dc072d000 r-xp 00000000 fd:00 79471      /usr/lib64/libtinfo.so.5.9
7f8dc072d000-7f8dc092d000 ---p 00025000 fd:00 79471      /usr/lib64/libtinfo.so.5.9
7f8dc092d000-7f8dc0931000 r--p 00025000 fd:00 79471      /usr/lib64/libtinfo.so.5.9
7f8dc0931000-7f8dc0932000 rw-p 00029000 fd:00 79471      /usr/lib64/libtinfo.so.5.9
7f8dc0932000-7f8dc0954000 r-xp 00000000 fd:00 79056      /usr/lib64/ld-2.17.so
```

仮想メモリの利用範囲

パーミッション

r - 読み
w - 書き
x - 実行
p - private

利用状況

strace

プログラム実行時の呼び出しシステムコールをみる

```
tusedls01$ strace ./address
execve("./address", ["/address"], [/* 35 vars */]) = 0
brk(NULL)
  = 0x1775000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9bbc68000
access("/etc/ld.so.preload", R_OK)
  = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=157557, ...}) = 0
mmap(NULL, 157557, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff9bbc41000
close(3)
  = 0
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0\0\1\0\0\0\340\2\0\0\0\0"... 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2151672, ...}) = 0
mmap(NULL, 3981792, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff9bb67b000
mprotect(0x7ff9bb67b000, 2097152, PROT_NONE) = 0
mmap(0x7ff9bb67b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c2000) = 0x7ff9bb67b000
mmap(0x7ff9bb67b000, 16864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ff9bb67b000
close(3)
  = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9bbc40000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9bbc3e000
arch_prctl(ARCH_SET_FS, 0x7ff9bbc3e740) = 0
mprotect(0x7ff9bb67b000, 16384, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ) = 0
mprotect(0x7ff9bbc69000, 4096, PROT_READ) = 0
munmap(0x7ff9bbc41000, 157557)
  = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ff9bbc67000
write(1, "main:&c = 0x7ffdd4171ae4\n", 25main:&c = 0x7ffdd4171ae4
) = 25
write(1, "&global = 0x601034\n", 19&global = 0x601034
) = 19
brk(NULL)
  = 0x1775000
brk(0x1796000)
  = 0x1796000
brk(NULL)
  = 0x1796000
write(1, "main:p = 0x1775010\n", 19main:p = 0x1775010
) = 19
write(1, "func:&i = 0x7ffdd4171acc\n", 25func:&i = 0x7ffdd4171acc
) = 25
write(1, "&main = 0x400580\n", 17&main = 0x400580
) = 17
write(1, "&func = 0x40055d\n", 17&func = 0x40055d
) = 17
exit_group(17)
+++ exited with 17 +++
tusedls01$
```

```
tusedls01$ strace -e mmap ./address
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fade1028000
mmap(NULL, 157557, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fade1001000
mmap(NULL, 3981792, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fade0a3b000
mmap(0x7fade0dfd000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c2000) = 0x7fade0dfd000
mmap(0x7fade0e03000, 16864, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fade0e03000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fade1000000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fade0ffe000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fade1027000
main:&c = 0x7fff40bbad94
&global = 0x601034
main:p = 0x18f7010
func:&i = 0x7fff40bbad7c
&main = 0x400580
&func = 0x40055d
+++ exited with 17 +++
tusedls01$
```

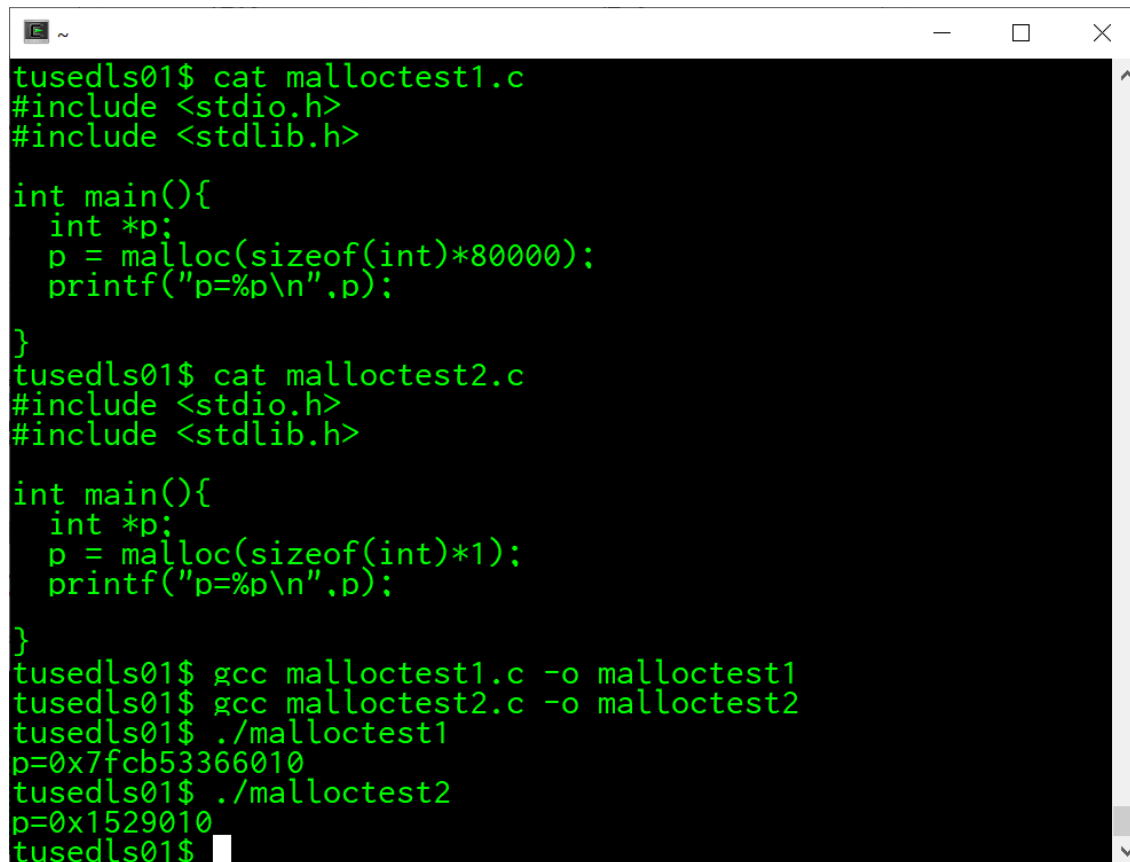
-e オプションをつけると
特定のシステムコールのみ表示

登場するシステムコール

- mmap(スタック用)
 - OS上のリソースの一部または全部を連続した仮想アドレス空間にマッピングする関数
- brk(ヒープ用)
 - プログラムブレーク (program break) の場所を変更する
 - プログラムブレークはデータセグメントの末尾を示す
 - プログラムブレークを増やす→プロセスへのメモリを割り当てる
 - プログラムブレークを減らす→メモリを解放する
- mprotect
 - メモリ領域の保護設定をする
 - read, write, execなどの権限を付与

余談:mallocとmmapとヒープ

- mallocは基本的にヒープ領域を使って割り当てる
- 一定サイズを超えるmallocが呼ばれた場合, mmapを使う



```
tusedls01$ cat malloctest1.c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int *p;
    p = malloc(sizeof(int)*80000);
    printf("p=%p\n",p);
}
tusedls01$ cat malloctest2.c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int *p;
    p = malloc(sizeof(int)*1);
    printf("p=%p\n",p);
}
tusedls01$ gcc malloctest1.c -o malloctest1
tusedls01$ gcc malloctest2.c -o malloctest2
tusedls01$ ./malloctest1
p=0x7fcb53366010
tusedls01$ ./malloctest2
p=0x1529010
tusedls01$
```

まとめ

- メモリ管理もOSの重要な機能の一つである
- 各プロセスは仮想メモリと仮想アドレス空間が独立に割り当てられる
- 仮想メモリと物理メモリの対応はMMUが管理する
- ユーザプログラムは仮想アドレス空間しか見えない
- 仮想メモリにはカーネル領域とユーザ領域がある
 - カーネル領域は全プロセスで共通である
 - ユーザ領域はプロセスごとの仮想のメモリ空間
- 仮想メモリのアドレス空間は、プログラム、データ、スタックの3領域がある

質問あればどうぞ