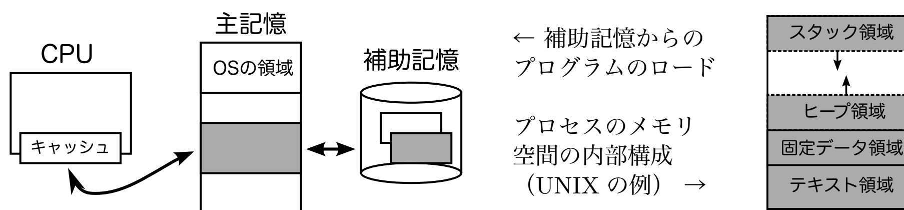
目次

- メモリの管理
  - プロセスのメモリ領域
  - メモリ領域の割り付け
  - 相対アドレス，絶対アドレス
- 仮想記憶
  - 仮想記憶とは
  - ページングとアドレス変換
  - ページの置き換え

★1　メモリの管理

プログラムは，**補助記憶装置** (☆ 1) から**主記憶装置（メインメモリ）**にロードされてから実行される．プログラムをロードするタイミングや，メモリ上にどのように割り付けるか，等を決める**メモリ管理**も OS の仕事の一つである．

☆ 1) 補助記憶装置: 二次記憶装置ともいう．一般に HDD 等のディスク装置が用いられる．
ロード: load



★1.1　プロセスのメモリ領域

個々のプロセスに割り付けられるメモリ領域は，以下のような部分から成る（右上図参照）．ヒープとスタックの領域は，プロセスの実行中に動的に拡大できるようになっている．

- テキスト領域: プログラムの命令部分
- 固定データ領域: プログラムのデータ部分
- ヒープ領域: プログラム中での動的なメモリ割り当て要求 (☆ 2) に応じて使用
- スタック領域: ローカル変数の格納や，サブルーチン呼び出し時のレジスタ値の退避などに使用

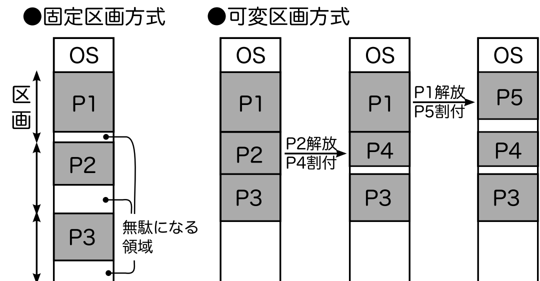
☆ 2) 例えば C 言語では `malloc()` 等の関数を利用してメモリを動的に確保することができる (コンパイル時ではなくプログラムの実行時に記憶領域の大きさを決められる)．

★1.2　メモリ領域の割り付け

マルチプログラミングシステムではプロセスを順次切り替えながら実行するため，主記憶上に複数のプロセスのメモリ領域を割り付ける作業が必要となる．この割り付けの方式は，(1) 一つのプロセス用に連続した領域を割り付けるかそれとも分割された不連続な領域を割り付けるか，(2) メモリを割り付ける**区画**のサイズを固定とするかそれともプロセス毎に可変とするか，によって分類される．

区画: partition

区画サイズを固定とする**固定区画方式**は管理が容易であるが，右図（いずれも連続な割り付けの場合を示している）に示すように，無駄な領域が発生するので，メモリ使用効率が低くなる．一方，**可変区画方式**では，必要サイズのみ割り付けるので使用効率は高いが，解放と割り付けを繰り返していくうちに小さな空き領域がそこらじゅうにできてしまう**断片化**（**フラグメンテーション**）という現象が発生しやすい．



可変区画方式では断片化は避けられないため，定期的に各区画を再配置して空き領域をまとめる**コンパクション** (☆ 3) と呼ばれる操作を行う必要がある．

☆ 3) コンパクション: compaction. デフラグメンテーション (defragmentation) とも．ディスク装置でも同様の問題は発生する．

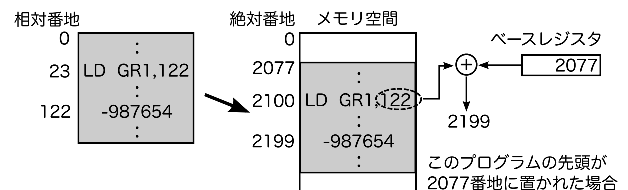
マルチプログラミングで並行動作するプロセスが多いと，全てのプロセスを主記憶上に置いておけなくなる．そのような場合，レディ状態のプロセスのメモリ領域を補助記憶上に一時的に追い出して，かわりに別のプロセスを主記憶上にロードして実行するようにすればよい．このように，補助記憶を用いて主記憶に置くプロセスを交換できるようにすることを，**スワッピング** (☆ 4) という．

☆ 4) スワッピング: swapping. 主記憶の内容を補助記憶に退避させることを**スワップアウト**，補助記憶の内容を主記憶に移すことを**スワップイン**という．

★1.3　相対アドレス，絶対アドレス

あるプログラムが主記憶上に置かれる位置は，実行の度に異なる．さらに，コンパクションやスワッピングのために，実行開始から終了までの間に位置が変化することもある．これらの理由から，プログラムをコンパイル／アセンブルしてオブジェクトプログラムを作成する時点では，ロード／ストアや分岐のように番地指定の必要のある命令が指定すべき番地を決定しておくことができない．

この問題を解決する方法の一つは，右図に示すように，オブジェクトプログラムの段階ではプログラム自身の先頭を 0 番地とする**相対アドレス**で番地を指定しておき，実行の際には，このプログラムの先頭が主記憶上で実際に置かれた位置を表す特別なレジスタ（**ベースレジスタ**（**再配置レジスタ**ともいう））を用いて実際の主記憶上の番地（**絶対アドレス**）を計算する，というものである．現代のコンピュータではこのような番地計算はハードウェアで自動的に行われる．



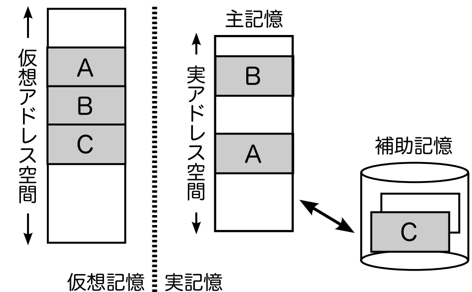
このような方式を採ることで，プログラムを主記憶上の任意の場所に置くことができるようになる．このようなプログラムは**再配置可能**であるという (☆ 5)．

☆ 5) プログラムを再配置可能にする方式はここに示したもの以外にもあるが，この授業では省略する．

★2　仮想記憶

★2.1　仮想記憶とは

仮想記憶 (virtual memory) とは，実在する主記憶のメモリ空間（アドレス空間）と切り離された仮想的・論理的なメモリ空間のことである．仮想記憶につけた番地を**仮想アドレス**といい，仮想アドレスで指定されるメモリ空間を**仮想アドレス空間**という．また，実在の主記憶の番地とメモリ空間を**実アドレス**, **実アドレス空間**という．現代の OS では，プロセスを仮想記憶上で動作させる方式が広く用いられている．この方式には，次のような利点がある．



1. **主記憶の容量を超える大きさのアドレス空間を扱える**
   プロセス毎のアドレス空間のサイズの合計，あるいは一つのプロセスのアドレス空間のサイズが，実際にコンピュータに搭載されている主記憶の容量を超えることができる (☆6)．これは，主記憶に入りきらない分を補助記憶装置上に置くことで実現される．あるプロセスのアドレス空間の全体または一部を主記憶／補助記憶のどこに置くかは OS が自動的に決定して管理してくれる．
2. **プログラミングの手間が省ける**
   仮想記憶方式では，プログラムは仮想アドレスを用いてメモリアクセスする．仮想アドレスから実アドレスへの変換や，アクセス先が主記憶上になかった時にその領域を主記憶上に持ってくる操作は，ハードウェアや OS によって自動的に行われるので，プログラムの側はそのような操作に煩わされずに済む．主記憶の容量がいくつであるかも気にしないで済む．
3. **記憶保護が容易になる**
   例えば，仮想アドレス空間をプロセス毎に独立に持たせるようにしておけば，あるプロセスが間違えて他のプロセスのアドレス空間にアクセスしてしまうようなことを防げる．このように，仮想記憶を用いると**記憶保護**がきちんとできる (☆7)．

☆6)［発展］仮想記憶の実用化の前にも，プログラムのメモリ領域をいくつかの部分に分割して一部を補助記憶に追い出す**オーバレイ**という方式でこのようなことは実現されていた．スワッピングも目的は同様である．

☆7) きちんと記憶保護をしていないと，応用プログラムが間違えて OS の領域を書き換え，システム全体がクラッシュしてしまうこともある．

★2.2　ページングとアドレス変換

仮想記憶では，アドレス空間を一定サイズの**ページ** (☆8) に分割してこれを単位として扱う方式が代表的である．これを**ページング**という．ページング方式の場合，以下に示すように，仮想アドレスも実アドレスも「**ページ番号**」と「**ページ内オフセット**」（ページ内での位置を表す）から構成され，**ページテーブル**と呼ばれる表を用いて仮想アドレスから実アドレスへの変換が行われる (☆9)．ページテーブルの管理は OS が行う．

☆8) ページ，ページング: page, paging. 1 ページのサイズは 4KB のことが多い．

☆9) このような変換はハードウェアで高速に実行できるようになっていることが多い．

仮想記憶における アドレス変換の例

仮想アドレス: 8bit
実アドレス: 5bit
ページサイズ: 4 語

アドレスの下位 2bit が ページ内オフセット，上位 bit がページ番号

仮想アドレス
000000 00
000000 01
000000 10
000000 11
000001 00
000001 01
000001 10
000001 11
:
111111 10
111111 11

ページテーブル

仮想ページ番号	有効 bit	実ページ番号
000000	1	101
000001	1	000
000010	0	
:	:	:
111111	0	

● 仮想アドレス 00000110 へのアクセス
→ 実アドレス 00010 に対応付けられる
● 仮想アドレス 00001001 へのアクセス
→ ページフォルト → そのページを主記憶へ
→ ページテーブル更新 → 実アドレスを…

実アドレス	内容
000 00	ADDA GR1,GR2
000 01	:
000 10	123
000 11	’H’
001 00	
001 01	
001 10	
001 11	
:	:
111 10	
111 11	111

一般に，全ての仮想ページを主記憶上に置いておくことはできず，一部のページは補助記憶上に置かれることになる．アクセス先のページが主記憶上にない（対応する実ページ番号が存在しない）ことを**ページフォルト** (page fault) という．そのようなページはページテーブルの「有効 bit」で区別される (☆ 10)．

☆ 10) この授業では省略したが，キャッシュメモリでもこの例と類似の仕組みが用いられる．調べてみよう．

UNIX ではプロセス毎に独立した仮想アドレス空間を持つようになっている．すなわち，2 つのプロセスの同じ仮想アドレスがそれぞれ異なる実アドレスに対応づけられる．

★ 2.3 ページの置き換え

主記憶がいっぱい（全ての実ページが仮想ページに対応づけられた状態）のときにページフォルトが発生したら，いずれかのページを補助記憶に追い出して，必要なページを主記憶上にコピーしなければならない．補助記憶へのアクセスは主記憶に比べて桁違いに遅いのが普通なので，このようなページの置き換えはなるべく少ない方がよい (☆ 11)．そのため仮想記憶では，主記憶から追い出すページの選択法を工夫することが重要となる．

このページ選択のアルゴリズムとしては，**LRU 法** (☆ 12) がよく用いられる．これは，「最近いちばん使われていないページを置き換える」というものである．

☆ 11) 主記憶容量が少ないと，ページ置き換えが頻発して通常の処理がままならなくなる，**スラッシング**という状態に陥ることがある．

☆ 12) LRU: Least Recently Used. LRU 法は，キャッシュメモリの解説時に登場した「メモリアクセスの時間的局所性」を活かすアルゴリズムといえる．

Q1. 主記憶上に A,B,C,D,E という 5 ページが存在しており，最近のアクセスを古い方から並べると C,E,B,A,E,D,C という順だった．この状況でページフォルトが発生した場合，LRU 法ではどれが新しいページに置き換えられるか．また，続けて再度ページフォルトが発生した場合，今度はどれが置き換えられるか．