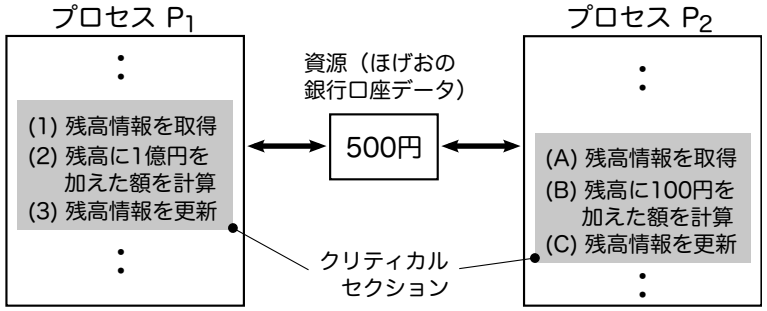
目次

★11　プロセスの管理とスケジューリング（前回のつづき）

★11.5　プロセス間の同期と通信

複数のプロセスが資源を共有しながら並行して動作する場合，それらがタイミングをあわせて（同期をとって）処理を行うようにしないと，矛盾が生ずることがある．例えば，右図のように銀行口座データのファイルを扱うプロセス P_1 と P_2 が並行動作していたとする．このとき，(1) → (A) → (2) → (3) → (B) → (C) という順に処理が行われると，大変なことになってしまう．



Q1. どう大変なことになるのか説明しなさい．

この問題を避けるためには，一方のプロセスが図のグレイの部分の処理を行う際には資源を占有して，その部分の処理が終わるまでは他方のプロセスで同じ資源にアクセスする処理を開始できないようにすればよい．このようにプロセスの実行を制御することを**排他制御**という．また，図のグレイの部分のように資源を占有してひと続きに実行する必要のある箇所を**クリティカルセクション**という．

クリティカルセクション: critical section

排他制御によく用いられるのは，**ロック**という方式である．これは，“lock”（鍵をかける）という名が表す通り，資源に鍵がかかっている／いないを表す 2 値変数を用いる方式である (☆ 1)．クリティカルセクションを実行する時は，次のようにする；(1) 対象資源のロック変数の値を確認し，ロックされていれば待機．さもなくばロックしてクリティカルセクションの実行に入る．(2) クリティカルセクションを抜けたらロックを解除．

☆1) プロセスの同期と排他制御の機構としては，資源のロック状態を 2 値ではなく整数で表す**セマフォ**もよく用いられるが，この授業では説明を省略する．

ロックのような排他制御の機構では，複数の資源をロックしたい場合などに，下手をすると 2 つのプロセスが互いに相手のロック解除を待って処理が先に進まなくなる**デッドロック** (☆ 2) 状態に陥ることがある．排他制御の仕組みは，デッドロックが起こらないよう注意深く設計しなければならない．

☆2) デッドロック: deadlock. 3 つ以上のプロセスの間でも起こり得る．

また，複数のプロセスが互いにメッセージをやりとりできるように，OS は一般に**プロセス間通信**の機構を備えている．例えば，UNIX 系 OS では，パイプ，ソケット，共有メモリなど様々なプロセス間通信の手段が提供されている．

プロセス間通信: InterProcess Communication, IPC

★11.6 ［発展］ スレッド

上述のように，OS は自分が管理する資源をプロセス毎に割り当てる．UNIX などの OS では，アドレス空間 (☆ 3) もプロセス毎に独立に生成する．そのため，

- プロセスの生成やコンテクストスイッチングのオーバヘッドが大きい
- アドレス空間を共有していないので，プロセス同士がメモリ経由でデータをやりとりするのが簡単ではない．プロセス間通信を行う必要がある

といった問題を抱えている．最近の OS では，このような問題点を解決するため，1 つのプロセスの中に処理の流れを複数持たせることができるようにしている．この処理単位のことを**スレッド** (☆ 4) という．同じプロセス内のスレッドは資源を共有していて切り替えも「軽い」．複数の処理を切り替えながら実行するネットワークサーバや GUI を備えたプログラムなどの処理効率の改善に効果がある．SMP 等による並列処理にも向いている．

☆ 3) アドレス空間: メモリの空間．詳しくは次回以降．

☆ 4) スレッド: thread.「軽い」ので，軽量プロセスと呼ぶこともある．

★12 メモリの管理と仮想記憶

★12.0 メモリとアドレスに関する復習 $+\alpha$

Q2. 以下の問に答えなさい．

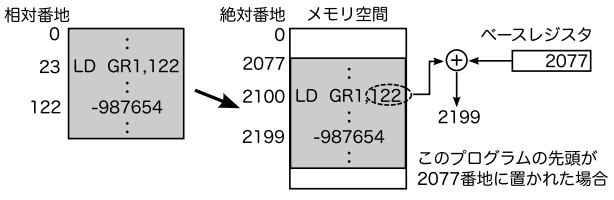
(1) 1 語 16bit で 4096 語分のメモリの容量は何 kB か．
(2) 1 語 16bit で最大 4096 語分のメモリを搭載可能なコンピュータを考える．ワードアドレッシングの場合，メモリの番地を表現するには何 bit 必要か．
(3) 1 語 16bit で最大 4096 語分のメモリを搭載可能なコンピュータを考える．バイトアドレッシングの場合，メモリの番地を表現するには何 bit 必要か．
(4) あるコンピュータでは，C 言語の int 型のデータがひとつ 32bit で表されるとする (☆ 5)．このコンピュータの 1MB 分のメモリには，int 型のデータを最大いくつ格納できるか．

☆ 5) C 言語では，int 型などのいくつかのデータ型の大きさは言語規格で定まっていないので，計算機環境によって異なる場合がある．

★12.0.1 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 (☆ 6) 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．

☆ 6) コンパクション，スワッピングは，メモリ管理のための働きの一種です．後で登場します．



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．

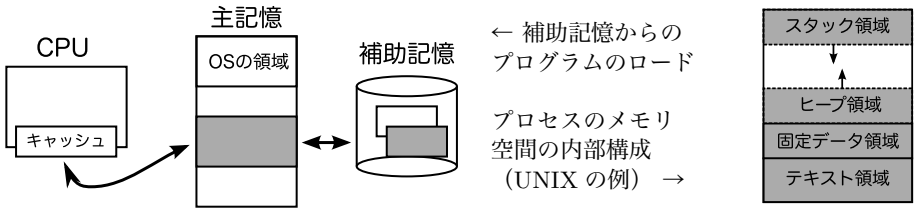
このような方式を採ることで，プログラムを主記憶上の任意の場所に置くことができるようになる．このようなプログラムは**再配置可能**であるという (☆ 7)．

☆ 7) 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．

★ 12.1 メモリの管理

プログラムは，**補助記憶装置** (☆ 8) 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．

☆ 8) 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



★ 12.1.1 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 (☆ 9) 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

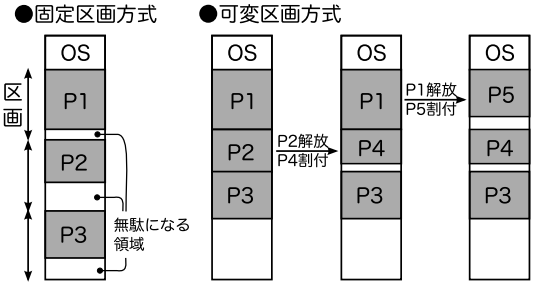
☆ 9) 例えば C 言語では `malloc()` 等の関数を利用してメモリを動的に確保することができる (コンパイル時ではなくプログラムの実行時に記憶領域の大きさを決められる)．

★ 12.1.2 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** (☆ 10) と呼ばれる操作を行う必要がある．

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** (☆ 11) という．

☆ 10) 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．

☆ 11) 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．