3-2 プログラムの実行

リンキングとローディング

プログラムを開発するには，いくつかのより小さな単位に分割して作成し，後でこれらを合わせて1つのプログラムにする方法が用いられる．この分割された個々のプログラムを**モジュール**(module)という．モジュールごとにコンパイルが行われ，**オブジェクトモジュール**が生成される．これらのオブジェクトモジュールは，一般に，それぞれ独立してロードし実行することはできず，すべてが結合されて，ローディング可能な**ロードモジュール**が生成される．このように，いくつかのオブジェクトモジュールを結合して1つのロードモジュールを生成することを**リンキング**(linking)あるいは**リンク**という．リンキングは，既存のプログラムおよびその一部を用いてプログラムを開発する場合にも有効である．また，リンクされるモジュールは，通常，異なるプログラミング言語で書かれたプログラムであってもよい．

では，リンキングはどのように行われるのであろうか．プログラミング言語で用いられるサブルーチンや変数の名前は，人間がプログラムを開発したり理解したりしやすくするためものであり，実行する際には，コンピュータの命令がメモリ上においてデータや他の命令の位置を識別するのに使用するアドレスに変換しなければならない．名前からアドレスに変換する操作は，コンパイラやアセンブラによって行われる．ただし，各モジュールをコンパイルした時点では，最終的なアドレスを得ることはできない．このため，モジュール内で参照するアドレスについては，そのモジュールの先頭を基点とする相対アドレスに変換し，モジュール外への参照は，名前のまま残しておく．このようなオブジェクトモジュールを複数結合して，相対アドレスや名前による参照を，ロードモジュールで定義されるアドレスに変換するのがリンキングである．リンキングを行うプログラムを**リンカ**(linker)または**リンケージエディタ**(linkage editor)という．

オブジェクトモジュール内の相対アドレスをロードモジュール内のアドレスに変換するには，各相対アドレスに，そのモジュールの先頭アドレスを加えればよい．ロードモジュールそのものの先頭アドレスは，オペレーティングシステムが決定するかユーザが指定する．ロードモジュールの先頭アドレスが決まれば，各モジュールの先頭アドレスはリンカがモジュールを順に並べていく過程で簡単に求められる．このようなアドレスの変更操作を**再配置**(relocation)といい，再配置が可能な形式のプログラムを**再配置可能プログラム**(relocatable program)という．

リンキングを行うには，オブジェクトモジュールごとに次のような情報が必要である．これらの情報は，コンパイラやアセンブラがオブジェクトモジュールといっしょに生成する．

(1) モジュール内で参照されているが，定義されていない名前　モジュール外で定義された名前の参照とみなされる．これを**外部参照**(external reference)という．高水準言語ではサブルーチン呼出しや大域変数の参照が外部参照として処理される．

(2) モジュール外部から参照される可能性のある名前　モジュール外部からの参照を許す名前(外部名)は，あらかじめプログラムで指定する．高水準言語ではサブルーチンや大域変数の宣言により，これらの名前が外部名として扱われる．

(3) オペランドに相対アドレスを含み，その書換えが必要な命令のアドレス

リンカは，ロードモジュールを補助記憶にファイルとして格納する．これを主記憶に読み込んで実行するのが**ローダ**である．ローダは，一般に，次のような手順でプログラムのローディングを行う．

[1] プログラムを格納しているファイルを見つけ出す．
[2] プログラムを実行するプロセスを生成する．
[3] 記憶領域を確保する．
[4] ファイルからプログラムを読み込む．
[5] パラメータを渡し，指定されたアドレスから実行を開始する．

コマンドインタプリタは，コマンドを処理するためのさまざまな操作と合わせて，上記の操作を実行する．

リンカとローダの機能を合わせもったプログラムを**リンキングローダ**という．リンキングローダは，実行可能なロードモジュールを直接主記憶上に作成するので，プログラムを試験的に実行するには手っとり早いが，そのつどロードモジュールを生成しなければならないので，同じプログラムを何回か実行する場合にはリンカを用いた方が効率的である．

ロードモジュールをファイルに格納する場合には，ローディングに必要なさまざまな情報もいっしょに格納される．こうした情報は，通常，ファイルの先頭のヘッダと呼ばれるデータ構造に置かれる．ローディングに必要な情報には，次のようなものがある．

- プログラムの大きさ　プログラムをロードする領域の確保に用いられる．
- プログラムの構成に関する情報　コード，データ，スタック(後述)の大きさなど．
- プログラムの実行開始アドレス　この値がプログラムカウンタにセットされて，実行が開始される．
- 再配置に関する情報　ローディング時に再配置が行われる場合に指定される．

ヘッダにどのような情報を置くか，ヘッダをどう構成するかは，オペレーティングシステムによって異なる．例えばUNIXでは，上記のほか，ロードのしかたやプログラムの構成に関する細かい情報がヘッダに記載される．

プログラムの構成とサブルーチン呼出し

プログラムは少なくとも，**コード**，**データ**，**スタック**の3つの領域を含んでいる．コード領域は，コンピュータの命令の列であり，プログラムを実行中に書き込まれることがない領域をいう．このため，1つのコード領域を複数のプロセスが意識せずに共用することができる．書込みがないので，実際に共用していても，各プロセスが個別にもっていても，プログラムから見れば同じである．データ領域には，プログラムを実行中に使用するデータが置かれる．デー

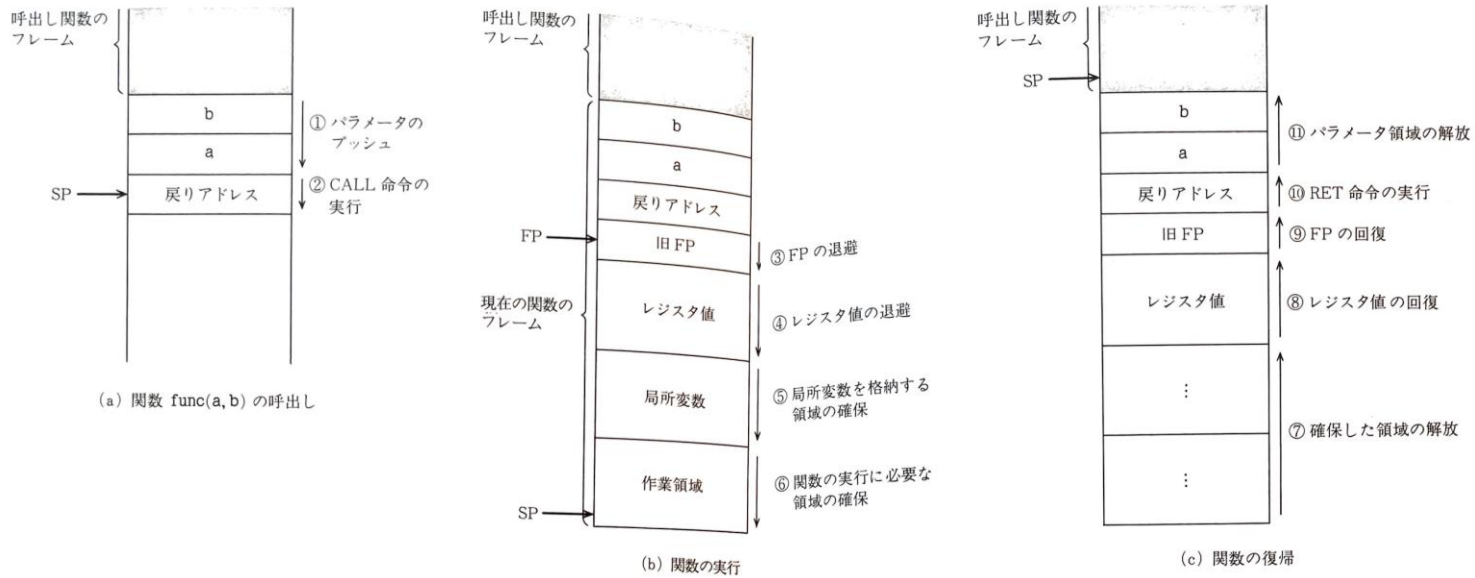

図 3.4 関数呼出し時におけるスタックのようす．FP はフレームポインタ，SP はスタックポインタ

タ領域は，読み書き可能な領域としてコード領域と区別される．高水準言語では，コンパイラが自動的に領域の振分けを行う．

UNIX では，データ領域は data と呼ばれる領域と bss と呼ばれる領域に分かれている．bss 領域は初期値が与えられていないデータを格納するための領域で，ロードモジュール内では，領域の大きさが記述されているだけである．オペレーティングシステムは，プログラムロード時に，記憶領域を割り当て，値をすべて 0 に初期設定する．

スタック領域は，サブルーチン(あるいは関数)の呼出しにおいて，戻りアドレスやサブルーチンのパラメータを格納するのに用いられる．スタックは，情報を後に入れたものほど先に取り出される仕組み(後入れ先出し)をもった記憶である．サブルーチン呼出しでは，サブルーチン A がサブルーチン B を呼び出したとき，後から呼び出した B が先に復帰し，次に A が復帰するので，後入れ先出しの性質をもったスタックが用いられる．スタックに情報を入れる操作を**プッシュ**，スタックからデータを取り出す操作を**ポップ**という．スタックに対してプッシュやポップの操作を実行するには，コンピュータが現在のスタックの先頭のアドレスを知っていなければならない．スタックの先頭アドレスを指すレジスタを**スタックポインタ**(stack pointer)という．

以下では，C 言語の関数呼出しを例にとり，スタックの利用について説明する．内容が多少細かいので，場合によっては読み飛ばしてもかまわない．

C の関数 func(a, b) を呼び出した場合を考えてみよう(図 3.4)．コンパイラは，パラメータ b，パラメータ a の順にスタックにプッシュする．次に，サブルーチン呼出し命令(CALL 命令)を実行する．この命令は，関数の戻りアドレスをスタックにプッシュし，呼び出す関数にジャンプする(プログラムカウンタに，呼び出す関数の先頭アドレスをセットする)というものである．呼び出された関数の先頭では，レジスタの退避を行う．これは，呼び出す側，呼び出される側それぞれ，変数の値や計算途中の結果を格納するのにレジスタを使用するからである．次に，関数の局所変数の領域をスタック上に確保し，指定があればその初期化を行う．

このように，1 回の関数呼出しに対応して確保される管理情報やパラメータ，局所変数などを格納するための連続領域を，**フレーム**(frame)または**活性レコード**(activation record)という．C や PASCAL などのプログラミング言語では，フレームがスタック上に確保され，復帰するときにスタックから取り除かれる．このようなフレームをとくに**スタックフレーム**(stack frame)と呼ぶ．フレーム内の情報を効率的に参照できるように，通常，フレーム内の特定の位置を指すベースレジスタが用いられる．これを**フレームポインタ**(frame pointer)という．フレームポインタはスタックフレームが確保されたときに設定し直され，これによって，それまで実行していた関数のスタックフレームの内容がスタックにプッシュされ，退避されることになる．これらの操作は，呼び出した関数の先頭で汎用レジスタの退避に先駆けて行われることが多い(このような操作を実現するコードは，コンパイラによって自動的に生成される)．

呼び出された関数の実行が終了すると，フレームポインタをもとに，スタックポインタを CALL 命令を実行した直後の位置にセットし，フレームポインタの値を回復する(退避しておいた値にセットする)．次にサブルーチン復帰命令(RET 命令)を実行する．これは，復帰アドレスをスタックからポップし，プログラムカウンタにセットするというものである．呼び出した側では，パラメータを一度にポップして，スタックポインタを関数呼出し前の位置に戻す．このように，関数呼出しの手順は，CALL/RET 命令の操作よりもはるかに複雑であり，特定の命令系列で実行される．関数呼出しのために実行される命令の系列を**呼出し系列**(calling sequence)という．呼出し系列は，コンピュータのアーキテクチャやプログラミング言語によって異なる．上に説明した手順はその一例である．