≪解　説≫

B 6700 ワークフローマネージメントについて

鈴木　勲*　神谷是公*

1. はじめに

B 6700 システムではジョブ制御はワーク・フロ・マネジメント システム (Work Flow Management System, 以下 WFMS と略す) で行なわれる. バッチ, TSS, リモート・バッチ, リアルタイム処理を同一システムで行なうことを前提としているため, 各処理形態の特性をそこなわないこと, またはユーザの地理的条件, そのハードウェア, ソフトウェアの知識の程度によって提供されるサービスの量, 質において差別されないこと, 任意のサービス・レベルを用意してシステムのバランスをとれること, などを目的としてWFMS は開発されている.

2. WFMS の構成

WFMS は**図—1** のように構成されている. コントローラ (controller) はジョブ・レベルのスケデューリングとシステムの負荷を制御している. タスクの制御, 及びシステム資源の管理は MCP (Master Control Program) で行なわれる. コントローラはさらにオペレータに対する指示, システムの状況 (ハードウェア的なものをソフトウェア的に解釈したもの) の表示, オペレータの指示の実行などを行なう.

WFL コンパイラはジョブ制御文を記述しているワーク・フロ言語 (Work Flow Language, WFL と略す) をコンパイルし, 機械語を作成する. この機械語は通常のコンパイラ言語である ALGOL や COBOL で書かれたユーザプログラムをコンパイルして作成された機械語と同一のものである. したがってジョブ制御文は解釈実行されるわけではない. これは B 6700 システムがスタック・アーキテクチャのマシンであり, 本質的にブロック構造のマシンであるためシステム全体を1つの大きなブロックとみなし, (ソフトウェア的には MCP がこれに相当する) ジョブは動的に発生, 消滅するサブ・ブロックのようにハードウェア的, ソフトウェア的に扱われるためである.

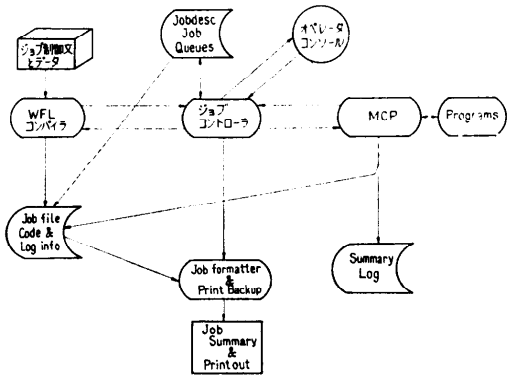


図-1 WFMS の構成

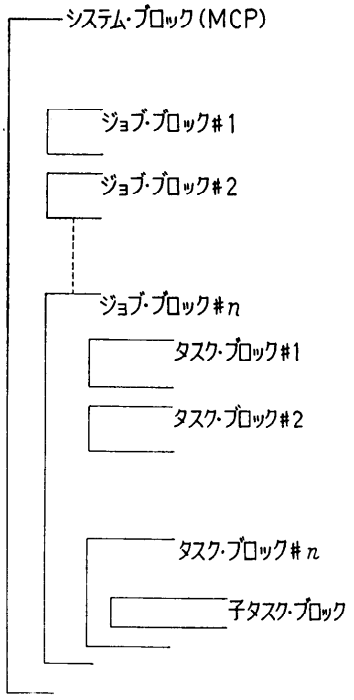


図-2 ジョブのブロック構造

3. ジョブの投入/起動方法

一つのシステムで, バッチ, TSS, リモート・バッ

* 高千穂バローズ (株) システム推進部

チ，リアルタイム処理を混在させて使用するとき，システムに投入されたジョブを，どのように制御するかを説明する．

3.1 システムの処理形態

システムの処理形態を簡単に図示すると図—3 のようになる．図—3 に示されているように，すべてのジョブは，WFMS に投入される．この WFMS はシステムの負荷を制御する手段を提供するものであり，それの中心的な，核となる機能がジョブ制御である．個々の処理形態のもつ内部的処理体系の相違は，MCP によって保障される．そしてジョブの終了と共に，結果のアウトプットやアカウンティング情報やジョブ・サマリ・リストは再び WFMS を通して，目的のユーザへ返される．

3.2 ジョブの投入

ジョブは，1つ，あるいは，複数のタスクから構成され，バッチ，リモート・バッチの場合，WFL というジョブ制御言語によって定義される．WFL で定義されたジョブは，そのコンパイラで機械語へ変換され，ジョブ・プロセスとしてジョブ・コントローラへ渡され，スケデュリングされる．したがってジョブ内で定義されているタスクは，ジョブ・プロセスによって発生させられ，ジョブ・プロセスの子タスク (Offspring task) として実行される．

更に，WFL 文を実行しているジョブ・プロセスにジョブ・ナンバが与えられ，このジョブ・ナンバによって結果のアウトプットや，アカウンティング情報はまとめられている．リモート・バッチ処理の場合も，ジョブの投入という点において基本的に変る問題はない．デマンド・ジョブの場合，その性質上ただちに処理過程に入らなければならない．(端末よりのアクセサビリティの保証) この問題はデマンド・ジョブを CANDE (Command AND Edit) というデマンド形のメッセージ・コントロール・プロセスのインターナル・プロセスとして CANDE 内で定義することによって，デマンド・ジョブの発生を直接 MCP に行なわせることによって解決される．

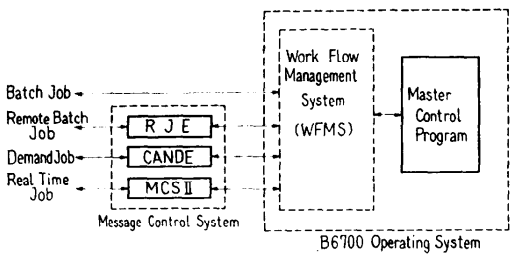


図-3 システムの処理形態

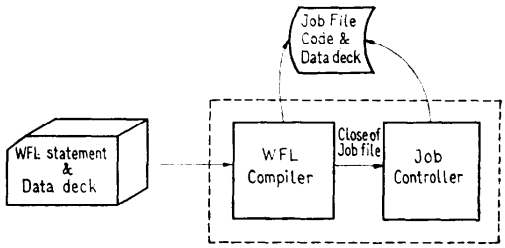


図-4 ジョブの投入例

もちろん，CANDE のインターナル・プロセスとしてのデマンド・ジョブは，CANDE と非同期に実行する．この場合，アカウンティング情報等のまとめなどの観点から，ある端末がログ・オン (log-on) することによって，ジョブ・ナンバを与え，ログ・オフ (log-off) と共に，1つのジョブが終了したとみなす．すなわちログ・オフまでの間に発生したデマンド・タスクはこのジョブ・ナンバの下で実行したものとして処理する．CANDE 自身は WFMS を通して発生させる．

3.3 ジョブ・スケジュリング

WFMS へ投入されたジョブは，あるジョブ・クラス内で優先順位に従ってスケデュールされる．

ジョブ・クラスは，CPU 時間，入出力量，優先順位等の上限，そのジョブ・クラス内での多重度，さらにそのジョブ・クラスで以前に実行されたジョブの開始時刻から次に実行すべきジョブの開始時刻までの間隔の上限などによって区分され，最大 1024 までのクラス分けが可能である．またある特定のジョブ・クラス内の優先順位は 0 から，そのジョブ・クラスで定義されている優先順付の上限までの範囲であり，優先順位の最大値は 99 である．**表—1** はジョブ・クラスの

表-1 ジョブクラスの例

Job class	Limit of CPU time	Limit of I/O time	Limit of Priority	Turn-Around Time
1	no limit	no limit	40	no limit
2	900秒	no limit	50	60分以内
3	900秒	300秒	55	30分以内
4	120秒	no limit	70	30分以内
5	50秒	no limit	85	2分以内
Default Queue	no limit	no limit	40	no limit
	※Job Class の指定のない Job の登録される Queue			

例である．ジョブがどのジョブ・クラスを選択するかは，ジョブ制御文によってなされるが，直接ジョブ・クラスを指定せず，CPU 時間や，入出力量や，優先順位等の上限を，ジョブ制御文で指定することによって，ジョブ・コントローラにその指定に一番適切なジョブ・クラスを選択させてもよいし，さらに，特定のインプット・デバイスとそのジョブクラスを結びつけておいて，そのインプット・デバイスから投入されたジョブを，そのデバイスと結びつけられているジョブ・クラスにスケデュールさせてもかまわない．ジョブに指定された CPU 時間，入出力量，優先順位等の制限は，そのジョブ・プロセスから発生するタスクに固有の制限が定義されていない限り，そのタスクにも適用される．なお優先順位は，CPU に対する要求度としての意味の他に，主記憶装置からのオーバレイの比率にも反映する．またこの優先順位は，ジョブ制御文や，ジョブ・クラスから整数部分の値として与えられ，プロセスの実行中，CPU の消費時間と入出力に消費した時間の比，(CPU/(CPU+IO)) を小数部分の値とし，全体を動的に変化する優先順位として扱うことも可能であり，小数部分を無視して，単純ラウンド・ロビン方式の採用も可能である．

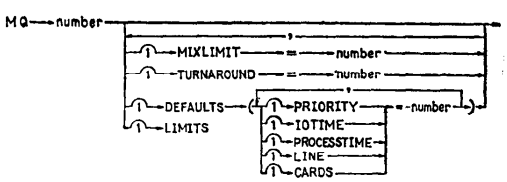
デマンド・ジョブに対する優先順位は，TSS 固有の方式を持っていて，多重レベル優先方式を採用している．1回の応答に対して，必要な CPU 時間がタイム・スライス（この値はユーザの業務内容によって自由に変更できる）をこえた時，そのデマンド・ジョブに与えられるタイム・スライスを延長し優先順位をさげるという操作を何段階か行ない，CPU 時間をあまり必要としないデマンド・ジョブを優先させる．何段階で制限するかは，ユーザによって決定することができる．

3.3 補足 ジョブ・キュウ（ジョブ・クラス）の作成と選択アルゴリズム

異なるサービスのレベルを提供するためジョブ・クラスを設定することができる．具体的にはジョブ・キュウを設定し，それぞれのキュウに制約を与え，異なるサービス・レベルをつくりだす方法を採用している．キュウ作成のためのコマンドは次の通りである．

MIXLIMIT はジョブの多重度を意味する．なおこのコマンドの記法は 5.1 文法記法の項を参照されたい．キュウの選択アルゴリズムは**図—5**（次頁参照）のようになる．

3.4 プログラム・ファイルの扱い方



プログラム・ファイルはコンパイルされると，ただちに実行可能なコード・ファイルとして磁気ディスク上にライブラリとして登録される．ただしプロセジャ単位，あるいはサブルーチン単位でコンパイルされたコード・ファイルは，他のコード・ファイルと結合して実行可能なコード・ファイルにしなければならない．いずれの場合も，このコード・ファイルは書込み (write 指令) から保護されている．コード・ファイルの構造，アクセス方式はどのような処理形態においても同一であり，したがって特定のプログラムを，バッチ処理で使用することも，デマンド処理で使用することも，セキュリティさえ満足していれば可能である．ジョブ制御言語でユーザ・コードというセキュリティを使用しないでコンパイルされたプログラムは，システム・ファイルとしてすべてのユーザによって利用可能であるけれども，コンパイルされたとき使用されたユーザ・コードと同じユーザ・コードを持たなければ使用が禁止されたり，他の特定のユーザに使用を許したりできる．更に特権的ユーザ・コードをもった人は，すべてのプログラムを使用できる．

4. ファイルおよび各種入出力機器，資源の使用方法

4.1 割り付け方法

各種入出力機器やファイルの割り付けは入力の場合はファイルの名前と，デバイスのタイプ指定があればそのタイプの一致するものを MCP がさがすことによって行なう．出力の場合はデバイスのタイプが指定されていればそのタイプの未使用装置，またはエリアを割り付ける．ジョブあるいはタスクの開始時点に使用可能な入出力装置を調べ，必要量が存在しない場合はジョブ/タスクの開始を中止する方式 (Resource limitation) と，このようなチェックを行なわずにジョブ/タスクを開始する方式 (Resource free) がある．(理由は後述)

入出力機器やファイルのタスクへの割り付けは必要になった時点で，すなわちファイルのオープン文を実

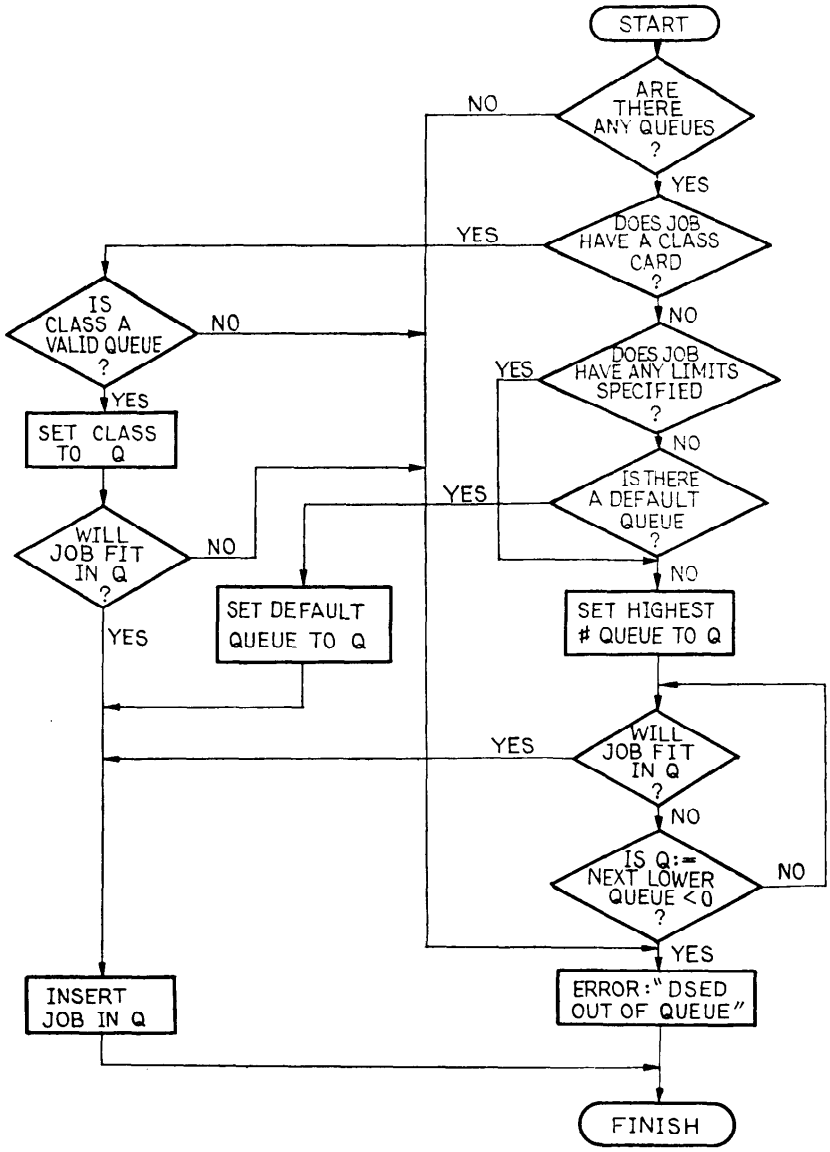


図-5 Job Enqweuing Alqorithm

行した時点，あるいは最初の入出力文を実行した時点で MCP が行う．磁気ディスクのスペースの割り当てはデマンド方式でロウ (ROW) 単位で，出力文の実行によって必要になった量を割り付ける．

メモリ・エリアの割り当ては，デマンド・ジョブの場合と，他の処理形態でのジョブに対する場合とは異なる．メモリ全体はデマンド・モードのスワップ・エリアと他のエリアとに分割される．スワップ・エリアはそのシステムでの全体のメモリ容量とデマンド・ジョブの処理量を考慮してインストレーションごとに決定する．

スワップ・エリアは，固定サイズのいくつかのスロットから構成され，特定のデマンド・ジョブにはそのジョブの平均コア使用量（ワーキング・セット・サイズ）にしたがっていくつかの連続したスロット（スワップ・スペース）が割り当てられる．もし割り当てら

れたスロットをこえてメモリを要求した時はスワップ・アウトされ，次のスワップ・インでスロット数が増加する．なお B 6700 のすべてのプログラムはコンパイル終了と共に，リ・エントラント可能であり，したがいコード部分とデータ部分は分離されている．しかしながらデマンド・ジョブの場合，コードの部分もスワップ・エリアにあると，リ・エントラントでなくなる．したがってデマンド・ジョブに対しても無条件にすべてをスワップ・エリアに割り当てるのではなく必要に応じてユーザがコマンドによって制御できるようになっている．すなわちすべてをスワップ・エリアに割り当てる方法，データ部分だけスワップ・エリアに割り当てる方法，すべてをスワップ・エリアに割り当てない方法を選択できる．

通常のエリアはダイナミック・ストレージ・アロケーションという考え方にもとづいて割り当てられ，未使用領域はリンクド・リスト (Linked List) で管理されている．この方法はデマンド・ジョブに対して，与えられた連続スロットからなるスワップ・スペースに対しても適用されている．

スワップ・エリアはメモリ全体からみて，使用領域となっている．なおバーチャル・メモリの実現法は，セグメーテンション方式を採用し，書き換え方式はデマンド方式をとりワーキングセット方式を採用している．

4.2 ファイルの使用方法/形態

次に複数のタスク間でのファイルの受け渡し，あるいは共用の問題，機密保護の問題を説明する．

ファイルのタスク間受け渡し方法は，ジョブ制御言語でグローバル・ファイルを宣言し，タスクを実行する時，そのタスク内で定義されているファイルと，このグローバル・ファイルを結びつけることによって行なわれる．制御の方法は，ジョブ制御文の実行としてのジョブ・プロセスと，ALGOL, COBOL 等の言語で書かれたプログラムの実行としてのタスクとの関係が，ALGOL の様なブロック構造を持つ言語の，アウタ・ブロック (outer block) と，インナ・ブロック (inner block) との関係と同じになっているため，グローバル・ファイルをジョブ・プロセス上に定義させることによってこのグローバル・ファイルは，タスクにとって，グローバルなファイルとなる．もし，ジョブ・プロセスとタスクの実行が非同期的であれば，タスク間でグローバル・ファイルを共有することができる．さらにプログラムのファイル宣言で，あるいはジョブ制御文でファイルの排他制御も可能である．

カード・ファイルについては，ローカル・デック，グローバル・デック，エクスターナル・デックの3種類の形態がある．ローカル・デックは，あるタスクにのみ使用可能なファイルであり，グローバル・デックは，ジョブを構成するすべてのタスクが使用可能であり，これらのカード・ファイルは，他のジョブからは使用できない．エクスターナル・デックはすべてのジョブあるのはタスクから使用できる．あるタスクがカード・ファイルを必要とした時，目的のカード・ファイルを探す順序は，ローカル・デック→グローバル・デック→エクスターナル・デックの順である．

次に異なる処理形態におけるファイルの統一的処理という問題であるが，ファイルの構造，アクセス方式のすべて共通の方式をとっている．したがってどのような処理形態をとっても，目的のファイルはセキュリティさえ満足していればアクセスできる．ファイルのセキュリティはいくつかのレベルに分けられる．そのファイルを作成した時ジョブ制御言語で指定したユーザ・コードと同じでなければアクセスできないファイル，特定のユーザ・コードと，特定のプログラムにのみインプットあるいはインプット/アウトプット，アウトプットでアクセスできるファイル，などである．

4.3 使用機器，資源の解放

割り当てられた使用機器，資源の解放は MCP によって行なわれる．解放されるきっかけは，クローズ文の実行の他，タスクやジョブの終了，ブロック (ALGOL 言語で定義されたブロックなど) を出る時である．タスクやジョブの終了も一種のブロックを出ることと同等なので，結局ブロックをぬけ出すときか，クローズ文を実行したときということができる．ブロックを出るとき，そのブロック内で定義されているファイルはクローズされ，そのブロック内で使用していたすべてのエリアはシステムに戻される．

5. ジョブ制御言語

広義のジョブ制御言語には次のような3種類のものがあるが，ここでは主として WFL について述べる．

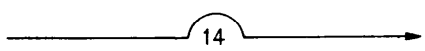
(1) WFL
(2) CANDE コマンド (TSS モードで，リモート・ターミナルで使用する)
(3) オペレータ コマンド

WFL は，もっとも一般的なジョブ制御言語でありアセンブラ言語形式ではなくコンパイラ言語形式を採

用している．WFL はセンタ・システムのカード読取装置，オペレータ用操作卓である映像表示鍵盤操作卓，RJE ターミナル，TSS ターミナルより入力することができる．以下に WFL の文法を説明するが，紙面の都合上最小限にとどめる．

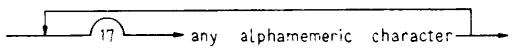
5.1 文法記法

文法は大文字，小文字，矢印，特殊文字（，・－／＝：＜＞＝；（）〔〕＋＊）で表現される．基本的な約束は矢印の方向に実線をたどって得られる文字列が文法的に正しい文となる．



これは，この実線を 14 回まで通過してもよいことを示している．たとえば

identifier

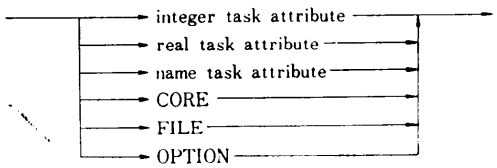


と表現されていれば，identifier は 17 字以内の英数字であるという意味になる．大文字はキイ・ワードであり，そのまま使用しなければならないことを示し，小文字の部分はユーザの選択部分，または任意記述部分である．

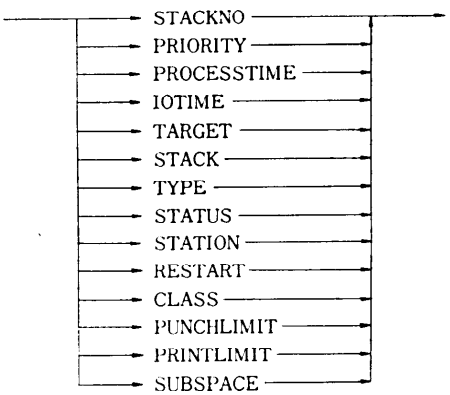
文法の具体的説明に入る前にジョブ，タスクを制御するために WFL で使用するタスクの属性，タスクの状態，ファイルの属性について述べる．

タスクの属性はタスクの実行を監視したり制御したりするために使用されコンパイルするときか，実行開始時に値が決定されるもの，実行中に動的に値が変化するもの（システムによって与えられる）がある．通常コンパイル時にタスクの属性はコンパイルによって値を与えられているため (defanlt value) 実行時にすべてのタスク属性について宣言したり，値を与えたりするわけではない．

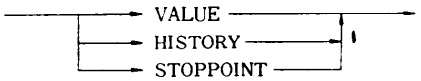
タスクの属性 (task attribute)



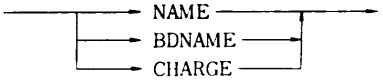
integer task attribute



real task attribute

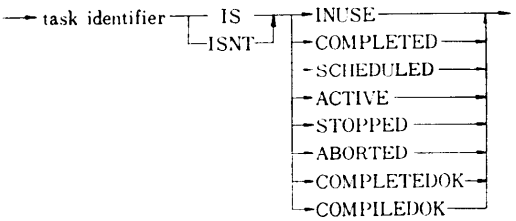


name task attribute



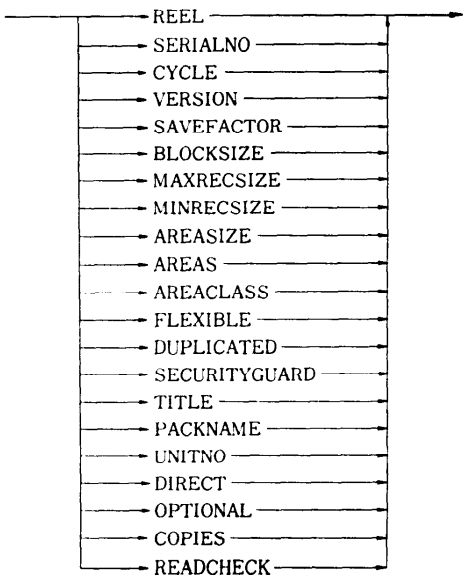
以上タスク属性の主要なものをあげたがこれらの意味については，紙面制約の都合上省略名から推察して頂くとして説明は省略する．詳細は参考文献 3)，4) を参照されたい．

WFL で参照できるタスクの状態はつぎの通りである．これらは条件文，代入文，ウェイト文などで参照できる．



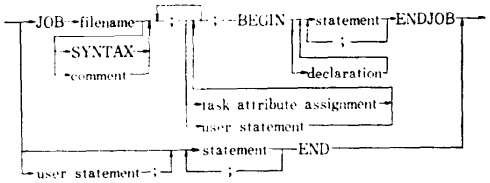
つぎにファイルの属性もコンパイル時にコンパイラが制御ブロックを作成するため実行時にファイルの属性をすべてを定義するわけではない．ファイルの属性は 100 以上定義されているが，代表的なものをあげると以下のようになる．詳しくは参考文献 5) を参照されたい．

ファイルの属性 (file attribute)



5.2 ジョブの記述

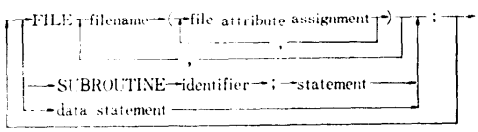
ジョブは WFL で静的に明確に記述される 1 つ以上のタスクのグループをいうだけでなく暗黙のうちに記述されている動的なものも含まれている.



ここで filename は, コンパイルしてできたコードファイル名という意味であるが, むしろ通常のいい方をすればジョブ名となる. むしろこの方がわかりやすい.

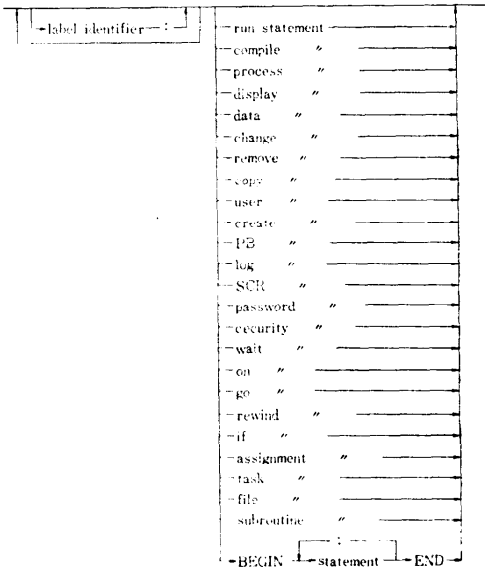
5.3 宣言

宣言が必要なのは, グローバル・ファイルとサブルーチンを使用する場合に限られている. サブルーチンはタスク開始ステートメントのグループに名前をつけて参照できるようにしてあるものをいう. 変数, ラベルなどは宣言する必要はなく文脈によって決定される.



5.4 制御文 (statement)

次にあげる20種類以上の制御文によって簡単なものから複雑なものまで記述できる. 複合文や代入文が使用できることが大きな特長である. この代入文は関数の使用をのぞくとほぼ ALGOL の代入文に相当する.



6. 例 題

例-1 FORTRAN のソースプログラムカードをコンパイルしてエラがなければ実行させる場合.

```
JOB SAMPLE 1; CLASS=1;
BEGIN
  COMPILE SAMPLEPROG FORTRAN;
  DATA
    ⋮
  END JOB.
```

例-2 FORTRAN のソースプログラムが磁気ディスク上にあり, それを一部分修正しながらコンパイルしてライブラリを作成する場合.

```
JOB SAMPLE 2; CLASS=1;
BEGIN
  COMPILE SAMPLEPROG FORTRAN LIB-
    RARY;
  FORTRAN FILE TAPE (TITLE=FORT-
    RNSOURCE);
  DATA
  $ SET MERGE
    ⋮                } 修正用プログラムカード
  END JOB
```

```
J
----- JOB STRUCTURE -----
  0230 JOB  55
W ..0231  55 *COPY/DECK
  0232 JOB  80 DCP/O
  0234 JOB  70 *SYSTEM/CANDE
  0243 JOB  55
W ..0244 55 ESPOL  JKL
S 0252 JOB  55
  0253 JOB  55
  ..0254 55 SEPCOMP   JHOST2/HOST
W ....0255 60 ESPOL   JHOST2/TEMP/SEP
  0248 JOB  55
S ..0249 55 *SYSTEM/DUMPANALYZER


M
----- JOB STRUCTURE -----
  0230 JOB  55
W ..0231 55 *COPY/DECK
      R: NO FILE PUNCH
      D:DISPLAY:NEXT DECK PLEASE..
  0232 JOB  80 DCP/O
    D:DCP 0 INITIALIZED*
  0234 JOB  70 *SYSTEM/CANDE
  0243 JOB  55
W ..0244 55 ESPOL   JKL
      R: OPERATOR STOPPED
S 0252 JOB  55
  0253 JOB  55
  ..0254 55 SEPCOMP  JHOST2/HOST
      D:JHOST2/TEMP/CDECK REMOVED.
W ....0255 60 ESPOL  JHOST2/TEMP/SEP
       R: NO FILE DINFO
  0248 JOB  55
S ..0249 55 *SYSTEM/DUMPANALYZER


C
----- COMPLETED ENTRIES -----
* 500/500 EOJ JOB
* 500/501 EOT COBOL DM6700MAIL/TRANS/PRO
* 502/502 EOJ JOB AUTOBACKUP
* 498/498 EOJ JOB
* 498/499 EOT LIBRARY/MAINTENANCE
  497/497 EOJ JOB AUTOBACKUP
  496/496 O-DS JOB
  495/495 EOJ JOB AUTOBACKUP
  491/491 EOJ JOB
  491/492 O-DS SYSTEM/DM6700ALL
  494/494 EOJ JOB AUTOBACKUP
  491/493 F-DS DM DM6700MAIL
  485/485 EOJ JOB AUTOBACKUP
  487/487 EOJ JOB
  487/488 R-DS DM6700MAIL/TRANS/PROG
  475/475 EOJ JOB AUTOBACKUP
  490/490 O-DS JOB
  489/489 EOJ JOB DM6700MAIL/TRANS/PROG
  481/481 EOJ JOB
```

図-6 Automatic Display Mode 表示例 (1)

例-3 FORTRAN のコンパイルとプログラム P1 の実行を同時に行ない，シンタックス・エラがなければプログラム P2 として実行させる場合.

```
JOB SAMPLE 3; CLASS=38;
BEGIN
```

```
W
----- WAITING ENTRIES -----
 0230/0231 55 *COPY/DECK
 NO FILE PUNCH
 0243/0244 55 ESPOL JKL OPERATOR STOPPED
 0253/0255 60 ESPOL  JHOST2/TEMP/SEP
 NO FILE DINFO

S
----- SCHEDULED ENTRIES -----
 0252 JOB  55
 0248/0249 55 *SYSTEM/DUMPANALYZER

A
----- ACTIVE ENTRIES -----
 0232 JOB  80 DCP/O
 0234 JOB  70 *SYSTEM/CANDE
 0243/0244 55 ESPOL  JKL
 0253/0254 55 SEPCOMP  JHOST2/HOST


MSG
----- MESSAGES -----
0135 DISPLAY:FILE HEADR.
0137 NO FILE FILES
0134 INV OPERATOR @ 005:000F:1*
0110 DISPLAY:CHECK OUTPUT.
0110 DISPLAY:FTR 113-0224.
0133 PSUB NOT BOUND @ 002:0000:3
0110 DISPLAY:FTR 113-0224.
0132 CR010 READ CHECK .
0130 NO FILE SYMBOL/UDSTRUCT
0107 DISPLAY:DISPLAY-0281.
0110 DISPLAY:FTR 113-0224.
0090 ASA/TEMP REMOVED.
0093 OPERATOR DSED @ 075:0070:1*
0120 INV INDEX @ 053:00D2:1*
0106 DISPLAY:DISPLAY-0281. *
0106 DISPLAY:DISPLAY-0281. *
0106 DISPLAY:DISPLAYADR. .
0109 NO FILE EARLY/BIND *
```

図-7 Automatic Display Mode 表示例 (2)

```
PROCESS COMPILE P2 FORTRAN (T1)
  LIBRARY;
DATA
  ⋮
RUN P1;
IF T1 ISNT COMPILEDOK THEN
BEGIN DISPLAY "SYNTAX ERR"; GO
  TO ERR;
END;
RUN P2;
ERR:
END JOB.
```

例-4 読者は次の例を解読してほしい．WFL の文法についてのわずかばかりの知識と，通常のコンパイラ言語の知識があれば容易に理解できるはずである.

```
JOB TEST 1; CLASS=2;
BEGIN
ON RESTART RUN MY/AUDIT/PROG;
FILE TAPE (TITLE=MASTER/FILE, KIND
```

```
  =PETAPE);
FILE PACK (TITLE=RUN/AGAIN, KIND
  =DISKPACK);
STARTPOINT:
IF TAPE ISNT PRESENT THEN
  BEGIN
  DISPLAY "MOUNT MASTER/FILE ON
    ANY UNIT";
  WAIT (TAPE);
  END;
PROCESS B;
RUN A;
IF A IS ABORTED THEN
  BEGIN
  DISPLAY "PROBLEMS-CALL DP MANA-
    GER";
  WAIT (PROGRAM B IS COMPLETED);
  GO TO DONE;
  END;
PROCESS C;
RUN D;
WAIT (PROGRAM C IS COMPLETED);
RUN E;
WAIT (PROGRAM B IS COMPLETED);
RUN F;
IF PACK IS PRESENT THEN
  BEGIN
  COMPILE SETUP/PROGRAM ALGOL LI-
    BRARY;
  IF SETUP/PROGRAM IS COMPILEDOK
    THEN
    BEGIN
    RUN SETUP/PROGRAM;
    GO TO STARTPOINT;
    END;
  END;
DONE:
END JOB;
```

7. ジョブの監視

通常規模のシステムで数十個のタスクが常時マルティ・プログラミング，マルティ・プロセシングされるB 6700 システムでは，ジョブの実行状況，システムの状態をオペレータが通常の方法で監視することは不可能に近い．そのため自動的にシステムの状態，入出力装置の割り当て状態などをオペレータにかわりやすい方法で表示する機能が用意されている．これをAutomatic Display Mode とよんでいる．

B 6700 システムではオペレータ用監視/制御操作卓として複数の映像表示鍵盤操作卓（以下ディスプレイ装置と略す）を使用する．どのディスプレイ装置にどのような情報をどのように表示するかを指示するためのコマンドがある．**図—6**，**図—7** はディスプレイ装置への表示例である．ユーザの指定により，一定時間間隔で表示情報を更新させることができる．

参考文献

1) E. I. Organick: Computer System Organization B 5700/B 6700 Series, Academic Press, 1973.
2) Burroughs Corp: ALGOL language information manual, Burroughs Corp, 1972.
3) Burroughs Corp: B 6700 System miscellanea, Burroughs Corp, 1973.
4) Burroughs Corp: B 6700 System software handbook, 1973.
5) Burroughs Corp: Input-output subsystem, 1974.
6) Burroughs Corp: B 6700 Work flow management reference manual, Burroughs Corp, 1973.
7) Burroughs Corp: B 6700 Work flow management user's guide, Burroughs Corp, 1973.

（昭和 49 年 5 月 24 日受付）