システム設計言語 DEAPLAN について†

林 達 也††

Abstract

A system design language, DEAPLAN, is described. This language is designed so as to be suitable for use in structured top-down design of the whole operating systems.

Some major features that seem to be necessary in such design languages are as follows:

(1) ability to define and use any module and/or module type whose logical level is higher than that of a procedure appearing in conventional programming languages,

(2) ability to define and use any module whose internal structure is undefined,

(3) ability to define and use any data and/or data type whose internal structure is undefined,

(4) ability to provide appropriate means for highly efficient structured design,

(5) ability to describe storage allocation and interface between modules.

Using such a language, it is hoped that we can take a step toward the computer aided design of operating systems.

1. まえがき

最近 Dijkstra の structured programming と呼ばれる手法が注目されている[1)~5)]. これはプログラムを全体から部分へと，段階的に確定していき，設計上必要な意志決定の全てを一度にまとめて行なわずに，それらが必要な段階までできるだけ延期し，各段階では絶対必要な意志決定のみを行なう手法である.

すなわち，structured programming は，このような必要最小限の意志決定の連鎖からなり，得られるプログラムは階層的なモジュール構造を持つ.

したがって，論理の正当性を直観的にとらえやすく，大規模であっても，最初から，誤りのないシステムを作成する上で極めて有効であると考えられる. また structured programming では，設計者は単に一つのプログラムを作るのではなく，実は潜在的には，一つのプログラムクラスを作成しているということができる. そこで，与えられた前提条件や目的が変更されても，プログラムの改良や修正が容易に行える.

関係するモジュールのみを他のものと取り替えればよいからである.

ここに述べるシステム設計言語（以下 DEAPLAN と呼ぶ）は，この手法を個々のプログラムから OS 全体の設計にまで拡張して，structured top-down design を可能にする目的で設計された.

ところで，OS をその設計段階から製造段階を通じて一貫して記述しようと試みたものには，他にも ESDL[6)]がある. しかし，これは DEAPLAN と異なり，設計段階では性能評価のために OS の制御の流れを記述することが主目的である. したがって，設計段階と製造段階では記述上の観点に相違があり，単一言語での記述は困難である. 実際，ESDL は 6 水準の言語群からなる複合言語である. また，文献 7）も性能評価に主眼が置かれている. 文献 8）ではシステムの構造を単純化してとらえ，オートマトン的な立場から OS の論理を記述する試みがなされている. DEAPLAN の場合はあくまでも単一言語であり，性能評価以前に，まず高品質な OS の作成を援助することが，主たる設計の狙いである. また特定の OS 構造を前提としてはいない.

以下においては，まず最初に設計言語において必要な機能を明らかにするために，**2.** で OS の論理構造を定式化する. それから **3.** で DEAPLAN の概要を説明し，**4.** でその記述例を示し，最後に **5.** で用途について簡単に触れる.

2. OS の論理構造

この論文では，

† DEAPLAN-a Design and Implementation Language for Operating Systems, by Tatsuya Hayashi (Software Planning Department, Fujitsu Limited)

†† 富士通(株)ソフトウェア計画部

(1) スペース,
(2) データ,
(3) モジュール

をOSを構成する本質的な実体と見なす.

スペースはデータやモジュールなどをその上に割付けるための記憶媒体で,主記憶,2次記憶,レジスタなどの種類がある.OSでは効率などの点で,割付けスペースや割付け方法を明示する必要があろう.

データは種々の型を持ち,原則として関係する処理対象物を抽象化して表現したものと考えられる†.

データは,モジュールによって値を付与されたり参照されたりする.

次にモジュールは,定められた論理機能を遂行する実体で,その複雑度にはさまざまのレベルがある,手続き (procedure) もそのようなレベルの一つである.

またここでは,OS全体も(最大の複雑度を持った)一つのモジュールと見なされる.

モジュールは一般に内部構造を持ち,内部データと成分モジュールとから構成される.したがって,OS全体は階層構造を成しているともいえる.

一方,OSにはスペース,データおよびモジュールが存在することから,それらの間には次のような関係が生じる.

(1) スペース～データまたはモジュール　割付関係,
(2) データ～モジュール　アクセス関係,
(3) モジュール～モジュール　呼出し関係.

以上の点から,ここではモジュールM(の論理構造)を次のような 11-tuple としてとらえる.

M=(E, S, D_x, D, P, V, A_x, A, C_x, C, L).

E: Mの入口の集合,
S: Mが割り付けられるスペースの集合,
D_x: 外部に存在するデータでMがアクセスするものの集合,
D: Mを構成する内部データの集合,
P: Mを構成する成分モジュールの集合,
V: D∪P の要素の割付け関係の集合,
A_x: D_x の要素に関するアクセス関係の集合,
A: Dの要素に関するアクセス関係の集合,
C_x: 外部のモジュールと自身またはPの要素との間の呼出し関係の集合,
C: Pの要素間の呼出し関係の集合,
L: コマンドの順序集合.

A_x, A および C_x, C のようなインタフェイス情報,あるいは S, V のような割付情報は,OSのような複雑な論理構造を表現する上で必要ではないかと思われる.

一方,モジュールMの論理そのものはLによって示される.Lはコマンドの順序集合であり,コマンドとはMの成分モジュールあるいはMの外部に存在するモジュールへの呼出し指令のことである.

各コマンドの形式は,それによって呼び出される相手モジュールのEにおいて定義されているものとする.Eの要素は入口と呼ばれそれぞれ入口名と0個以上の仮パラメータとから成り,対応するコマンドはその入口の仮パラメータを実パラメータで置き換えて得られる.

一つのコマンドによって,一つの論理単位が遂行される.そこで,もしLがn個のコマンドから成っていれば,モジュールMの論理は,それよりも複雑度の小さいn個の論理単位から形成されるわけである.

その場合,あるコマンドから次のコマンドへ制御が移る時点は,前のコマンドによって呼び出されたモジュールがその論理動作を完全に終了した場合とそうでない場合の二通りある.前者は直列処理の場合であり後者は並列処理の場合である.

コマンドによって呼び出されたモジュールの論理は,再びいくつかの小さい論理単位に分解することができる.このようにして,モジュールMの論理は究極的にそれ以上分解できない基本的なモジュール群によって表現される.それらとしては,PL/I[9]などのプログラミング言語に現われる IF～, DO～, ～+～, ABS(～)など (のコマンドに対応するモジュール) が適当であろう.この論文では,IF～, DO～などに対応するモジュールを ST (ステートメント)型, ～+～などに対応するものを OP(演算子)型, ABS(～)などに対応するものを PROC(手続き,関数)型と呼び,いずれもモジュールの一種と見なしている (**3.2** 参照).このような基本モジュールはいうまでもなくOSの外部にあらかじめ存在し,その論理構造は自明でなければならない.

3. 言語仕様

ここでは主として,DEAPLAN の基本的な特色と思われるものについて述べよう.詳細な仕様について

† 例えば OS/360 において,TCB と呼ばれるデータはタスクを,また UCB と呼ばれるデータは入出力装置を抽象化して表現したものである[13].

は別の機会にゆずる.

3.1 記述形式

DEAPLAN の記述単位はモジュールである. その場合, 各成分モジュールは, ステートメントや内部手続きあるいは演算子を除いて, それぞれ親モジュールとは独立した記述単位を構成する. それは,

(1) 成分モジュールの内部構造は, 原則として親モジュールの構造とは論理的に独立している. いいかえれば, 前者は後者にとって必要以上に詳細すぎる情報であり, 後者の直観的な把握をかえって困難にする.

(2) 全体を一つの記述単位にまとめると, 量的に極めて膨大なものとなり, 取扱いが不便である.

(3) 成分モジュールの追加/修正は, 常に全体の更新を必要とする.

などの理由による.

さてモジュールは8種類の宣言を用いて以下のように記述される (これを外部 MODULE 宣言と呼ぼう). キーワードは下線を付して示す.

MODULE モジュール名 モジュール型 [(親モジュール名)]
ENTRY 宣言 [ENTRY 宣言]…
DATA 宣言 [DATA 宣言]…
MODULE 宣言 [MODULE 宣言]…
LOGIC 宣言
SPACE 宣言 [SPACE 宣言]…
MAP 宣言 [MAP 宣言]…
ACCESS 宣言 [ACCESS 宣言]…
CALL 宣言 [CALL 宣言]…
END モジュール名

2. で述べた 11-tuple は各宣言と次のように対応する.

E～ENTRY 宣言の列, S～SPACE 宣言の列
D_x, D～DATA 宣言の列, V～MAP 宣言の列
P～MODULE 宣言の列, A_x, A～ACCESS 宣言の列
L～LOGIC 宣言, C_x, C～CALL 宣言の列

外部 MODULE 宣言に含まれる MODULE 宣言 (以下内部 MODULE 宣言と呼ぶ) は通常 ENTRY 宣言のみから成る. ただし, ステートメントや内部手続きまたは演算子の場合は設計の進行にともない, いずれ LOGIC 宣言 (必要なら MAP 宣言) まで含まなければならない. ACCESS, CALL 両宣言は不要である. また, あるモジュールが内部データや成分モジュールの単なる集合体 (例えば OS/360 のロードモジュールなど[12]) の場合は, LOGIC 宣言は不要である.

MODULE 宣言は, いわゆるブロックとしての効果を持つ. その場合記述単位としては独立しているが, 成分モジュールの外部 MODULE 宣言 (ブロック) をその親の外部 MODULE 宣言 (ブロック) の直接的な子ブロックと見なすのが自然であろう.

3.2 モジュールの種類

現在モジュールの標準的なレベルあるいは型として確立されているのは, PL/I などのプログラミング言語に現われるものに限られると考えられる. すなわち, ステートメント, 手続きおよび演算子に対応するもので, DEAPLAN ではそれぞれ ST 型, PROC 型および OP 型と呼ばれ, 標準型として扱われる. これに対して, 例えば OS/360 ではシステム全体(OS/360), コンポーネント, タスク, ロードモジュール, セグメントなどといったより高いレベルのモジュールが存在する[12]. そして, これらは名称こそ違え今日のどの OS にも存在しているはずである. そこで, システム設計言語としてはまず第一に, 高いレベルのモジュール型が自由に定義でき, その型の任意のモジュールが導入できなければならないであろう.

DEAPLAN の場合は, MODULE 宣言のところでたとえば,

MODULE TYPE TASK;

と型宣言を行なって TASK (タスク) 型を定義すれば, これを含む親 MODULE 宣言のブロック内で, 自由にこの型の成分モジュールを導入することができる.

なお, DEAPLAN の標準型には上記3種以外は存在しない. これらについて次に説明しよう.

(1) ST 型

マクロと BEGIN ブロックの概念を統合した効果を持つ. すなわち, ST 型の MODULE 宣言の本体は, まず最初に仮パラメータ (もしあれば) が実パラメータによってそっくり置換される (ALGOL 60 の call by name に等しい). ついでその本体は, そのモジュールを呼びだしたコマンドの位置に展開され, そのコマンドは消去される. そしてその展開された本体は BEGIN ブロックとしての効果を持つのである. ただし上記のことはあくまでも実効的な動作形式であって, DEAPLAN の上で実際に書き換えが生じるわけではない. また DEAPLAN には BEGIN ブロッ

クそのものは存在しない．BEGIN ブロックは必要以上に詳細な情報を加えて親の論理の直観的把握を困難にする．ST 型のように間接的に導入すれば，その問題は解決される，というのが著者の意見である．また，いわゆるテキストマクロそのものは，原理的にホスト言語の構文とは全く無関係に記述できる．したがって，テキストマクロを用いたドキュメントはかならずしも見やすいとは言えない．この問題も ST 型によって解決できるであろう．ST 型はモジュールの一種で論理的な意味を有しており，マクロのような単なる記号列ではないからである．

さらに ST 型は次の PROC 型と異なり実行時の動的なリンケージの必要がない．また後述するように，その内部データの動的な割付処理も不要にできる．

したがって，ST 型は効率を全然損なわずに structured design を可能にする上で有効と思われる．

(2) PROC 型

いわゆる手続きあるいは関数に相当する型で，パラメータは動的なリンケージの際に ALGOL 60, 68 や BLISS[10] の call by value 方式で渡される．PROC 型の場合これが最も自然だと思われる．

(3) OP 型

動作形式は ST 型と同じであるが，

a．レベルが最低である．

b．特殊記号をモジュール名に用いることができる．

c．パラメータは1個あるいは2個に限られる．

d．コマンドの形式は，パラメータの数によってそれぞれ prefix あるいは infix で，式の構成要素となる．

e．式の評価順序を定めるために，プライオリティを持つ．

などの点で ST 型とは異なる．

DEAPLAN にも ST 型，PROC 型，OP 型の基本モジュールが用意されており，それらは PL/I や BLISS とほとんど同じである．また，設計者がこれらの型のモジュールを自由に導入できるという点で，DEAPLAN は拡張可能言語でもある．

3.3 データの種類

システム設計言語においても，PL/I などのプログラミング言語に用意されているデータ型はやはり必要である．しかしそれ以外に，内部構造の未定義なデータやデータ型を導入できることが絶対に必要と考えられる．初期のステップでは，内部構造を決定することは困難だし，またその必要もないからである．

そこで DEAPLAN では，PL/I とほとんど同じ標準型の他に，そのような機能を持っている．たとえば DATA 宣言で，

```
DATA NAMETAB ORG;
```

とすれば，内部構造の未定義な NAMETAB なるデータが定義される．また，

```
DATA TYPE LINE ORG;
```

とすれば LINE なる内部構造の未定義な型が定義される．その場合，後のステップで決定される内部構造は，

```
DATA TYPE LINE ORG;
    ELM 1 P ORG,
          2 X BIN(15),
          2 Y BIN(15),
        1 Q ORG,
          2 X BIN(15),
          2 Y BIN(15);
```

のように元の定義文の直後に ELM 文を追加して明らかにすることもできる．ここで，P, Q は線分の端点を表わし，P.X および P.Y (Q.X および Q.Y) は点 P(Q) の2次元座標値（2進 15 桁の整数型）を示す．しかしその内部構造は，LINE が定義されている MODULE 宣言においては余分な情報で，このためにその論理が直観的にとらえにくくなる恐れがある．また，LINE が定義されている MODULE 宣言が別の外部 MODULE 宣言になっている場合は，ELM 文を追加するのはわずらわしい．そこで DEAPLAN では，内部構造を真に必要とする成分モジュールの MODULE 宣言の中で，

```
DATA TYPE EXT REFINED LINE ORG;
        ELM ——;
```

とすればよいようになっている．これは，LINE が最初は外部で定義されたデータ型であるが，その内部構造はここで ELM 文によって定義されることを示す．同様にデータに対しても，

```
DATA EXT REFINED NAMETAB ORG;
        ELM ——;
```

などとすればよい．

3.4 生存期間

一般に OS では，データだけでなくモジュールもシステムの動作期間の最初から最後まで継続して存在しているとは限らない．効率上または論理的な理由により，必要な時点で生成され，必要がなくなれば消去さ

れるのである.

したがってシステム設計言語では，データやモジュールの各々に対してその生存期間を指定できるのが望ましい．DEAPLAN の場合は，TSPAN 文を用いて次の3通りの指定方法がある．(ただし ST 型，内部 PROC 型，OP 型モジュールは除かれる.)†

(1) TSPAN STATIC;

(2) TSPAN AUTOMATIC;

(3) TSPAN CONTROLLED [(モジュール名, …)]

(1)が標準とされその場合は省略してもよい．これは，親モジュールの生存期間と同一であることを示す．あるモジュールMが STATIC であるとは，M自身およびその内部データ，成分モジュールの全てがMの親と同一の生存期間を持つことを意味する．ただしMの内部データや成分モジュールは再指定によって(2)あるいは(3)に変更することができる.

なお，ST 型，OP 型モジュールの場合は，それに対するコマンド（定義ではなく）を直接含む親の各々に対して，常に STATIC と見なされる．また，内部 PROC 型の場合は，その定義を直接含む親モジュールに対して常に STATIC と見なされる.

(2)は親モジュールがコマンドによって起動された時点で生成され，その論理動作が完了した時点で消去されることを意味する.

(3)はかっこ内のモジュール（もしあれば）によって，動的に生成／消去されることを意味する.

例

```
DATA NAMETAB ORG;
    TSPAN STATIC;

MODULE STEPTASK TASK;
 TSPAN CONTROLLED (STEPCONTROL);
```

3.5 コマンドとパラメータ

プログラミング言語で通常実行文あるいはステートメントなどと呼ばれているものは，DEAPLAN ではコマンドすなわちモジュールに対する呼出し指令と見なされる．いい換えれば，あるモジュールの論理は，レベルの高低にかかわりなく全て，他の1個以上のモジュール（再帰型の場合は自身も含まれる）を必要に応じて次々に呼び出すことによって実現されると考えるのである．これによって，単なるプログラムの世界をより広い OS の世界に無理なく解析接続できる，というのが著者の意見である.

さてコマンドの形式はモジュールのレベルによって異なる．それぞれ次に説明しよう.

(1) ST 型

例えば，IF A=B THEN X=Y ELSE X=Z のように，入口名 (IF THEN ELSE) とパラメータ (A=B, X=Y, X=Z) の混在が許される.

パラメータの種類は，

(a) 式　　(d) ラベル名

(b) スペース名　　(e) コマンド列†

(c) 入口名

で，全て call by name 方式である.

(2) PROC 型

たとえば，ABS(A*B) のように，入口名とパラメータとは分離され，入口名の後にパラメータ（もしあれば）がつづく．そしてパラメータ部はかっこでくくられる.

パラメータの種類は式のみに限られ，入口名やスペースの場合はポインタを介して間接的に渡される.

パラメータもデータと同様に，MAP 宣言や ACCESS 宣言で割付方法，アクセス方法が明らかにされる．その場合，パラメータの受け渡しは call by value なので，ポインタ型のパラメータで間接参照されるデータのみが ACCESS 宣言の対象となる.

(3) OP 型

A*B, −C などのように2項演算子の場合は infix, 単項演算子の場合は prefix である．この型のコマンドは式の中でのみ用いることができる.

パラメータに関しては，ST 型と同じである.

(4) 新モジュール型

PROC 型と同じである．例えば，WAIT (1, P) や POST (Q, 20) など．ここで，WAIT コマンドはP (ポインタ型)で示される同期信号群の中からどれか1個の信号が発生するまで動作を停止するためのもの．POST は逆にQ (ポインタ型) で示される同期信号の発生を通知するコマンドで，第2パラメータの 20 は相手方モジュールに渡す補足情報である[12].

次に基本コマンドの種類であるが，これは PL/I や BLISS と大差ない．ただし入出力コマンドは用意されない．また BLISS と異なり GOTO 文は含まれるが，使用しなくても済むように考慮されている.

† 実は非基底スペースに対しても指定する必要がある．これは PL/I の領域データに相当する.

† コマンド列は1個以上のコマンドから成り，コマンドの区切りは ";" である．但しコマンド列の最後は原則として "$" で示されるが，文脈で明らかな場合はなくてもよい.

```
MODULE TYPE SYSTEM % WHOLE OPERATING SYSTEM %;
MODULE MOS SYTEM;
   ENTRY EOS;
   DATA JOB_STREAM ORG; TSPAN CONTROLLED (IN_CTRL);
   DATA SYSIN ORG; TSPAN CONTROLLED (IN_CTRL, EXEC_CTRL);
   DATA (INPUT_QUEUE, OUTPUT_QUEUE) ORG; TSPAN STATIC;
   DATA SYSOUT ORG; TSPAN CONTROLLED (EXEC_CTRL, OUT_CTRL);
   DATA LIST_STREAM ORG; TSPAN CONTROLLED (OUT CTRL);
   DATA (SYSCATLG, SYSACCT, SYSSVCLIB, SYSEXECLIB, SYSJCLLIB) ORG; TSPAN STATIC;
   DATA CSCB ORG; TSPAN CONTROLLED (CONSOLE_COMMUNICATION, MASTER_CTRL, COMMAND_PROCESSOR);
   DATA CSCB_QUEUE ORG; ELM 1 (FIRST_CSCB, LAST_CSCB) PTR (CSCB); TSPAN CONTROLLED (MASTER_CTRL);
   DATA USER_FILE ORG; TSPAN CONTROLLED (EXEC_CTRL);

   MODULE TYPE NON_TASK % SUPERVISOR %;
   MODULE TYPE TASK;
   MODULE TASK_CTRL NON_TASK; TSPAN STATIC;
          ENTRY ATTACH, DETACH, LINK, XCTL, RETURN, WAIT, POST, PGM_INTERRUPT; END TASK_CTRL;
   MODULE IO_CTRL NON_TASK; TSPAN STATIC;
          ENTRY EXCP, IO_INTERRUPT; END IO_CTRL;
   MODULE MASTER_CTRL TASK; TSPAN STATIC;
          ENTRY EMC EQU EOS; END MASTER_CTRL;
   MODULE CONSOLE_COMMUNICATION TASK;
          TSPAN CONTROLLED (MASTER_CTRL);
          ENTRY ECC; END CONSOLE_COMMUNICATION;
   MODULE COMMAND_PROCESSOR TASK;
          TSPAN CONTROLLED (MASTER_CTRL);
          ENTRY ECP; END COMMAND_PROCESSOR;
   MODULE IN_CTRL TASK;
          TSPAN CONTROLLED (COMMAND_PROCESSOR);
          ENTRY EIC; END IN_CTRL;
   MODULE EXEC_CTRL TASK;
          TSPAN CONTROLLED (COMMAND_PROCESSOR);
          ENTRY ECC; END EXEC_CTRL;
   MODULE OUT_CTRL TASK;
          TSPAN CONTROLLED (COMMAND_PROCESSOR);
          ENTRY EOC; END OUT_CTRL;
   MODULE JOB_PROCESSOR TASK; TSPAN CONTROLLED (EXEC_CTRL);
          ENTRY EJP; END JOB_PROCESSOR;

   SPACE TYPE CR % CARD %, DA % DIRECT ACCESS STORAGE %, LP % LINE PRINTER SHEET %;
   SPACE BASE DECKS CR, SHEETS LP;
   SPACE BASE SYSTEM_RESIDENCE_VOLUME DA (1 VOL, (1 VOL=200 CYL, 1 CYL=10 TRK, 1 TRK=3000 BYTE)),
              SYSTEM_WORK_VOLUMES DA (2 VOL),
              USER_VOLUMES DA (10 VOL);
   SPACE BASE HCM MEM (1024 KB, (1 KB=1024 BYTE, 1 BYTE=8 BIT));
              ELM 1 RESIDENT_AREA          192 KB,
                  1 SYSTEM_WORK_AREA        64 KB,
                  1 TRANSIENT_AREA         768 KB;
   MAP JOB_STREAM ON DECKS, LIST_STREAM ON SHEETS;
   MAP (SYSIN, SYSOUT) ON SYSTEM_WORK_VOLUMES,
       (INPUT_QUEUE, OUTPUT_QUEUE) ON SYSTEM_WORK_VOLUMES BDRY 1 CYL;
   MAP (SYSCATLG, SYSACCT, SYSSVCLIB, SYSEXECLIB, SYSJCLLIB) ON SYSTEM_RESIDENCE_VOLUME;
       USER_FILE ON USER_VOLUMES;
   MAP (CSCB, CSCB_QUEUE) ON SYSTEM_WORE_AREA;
   MAP (TASK_CTRL, IO_CTRL, MASTER_CTRL, CONSOLE_COMMUNICATION, COMMAND_PROCESSOR) ON
       RESIDENT_AREA BDRY 8 BYTE;
   MAP (IN_CTRL, EXEC_CTRL, OUT_CTRL, JOB_PROCESSOR) ON TRANSIENT_AREA BDRY 4 BYTE;

   ACCESS IN_CTRL (USE (JOB_STREAM, SYSJCLLIB), SET SYSIN), JOB_PROCESSOR (USE SYSIN, UPDATE USER_FILE,
          SET SYSOUT), OUT_CTRL (USE SYSOUT, SET LIST_STREAM);
   ACCESS (IN_CTRL, EXEC_CTRL) UPDATE INPUT_QUEUE, (EXEC_CTRL, OUT_CTRL) UPDATE OUTPUT_QUEUE;
   ACCESS EXEC_CTRL UPDATE (SYSCATLG, SYSACCT), TASK_CTRL USE (SYSSVCLIB, SYSEXECLIB);
   ACCESS (MASTER_CTRL, CONSOLE_COMMUNICATION) UPDATE (CSCB_QUEUE, CSCB), COMMNAD_PROCESSOR USE
          CSCB;
```

```
   CALL FROM MASTER_CTRL TO (CONSOLE_COMMUNICATION, COMMAND_PROCESSOR),
        FROM COMMAND_PROCESSOR TO (IN_CTRL, EXEC_CTRL, OUT_CTRL),
        FROM EXEC_CTRL TO JOB_PROCESSOR;
   CALL TO (TASK_CTRL, IO_CTRL) FROM (MASTER_CTRL, CONSOLE_COMMUNICATTON, COMMAND_PROCESSOR,
        IN_CTRL, EXEC_CTRL, OUT_CTRL, JOB_PRNCESSOR);

END MOS;
MODULE CONSOLE_COMMUNNICATION TASK (MOS);
   ENTRY ECC;
   DATA EXT REFINED CSCB ORG;
            ELM 1 NXTCSCB   PTR (CSCB),
                1 STATUS    BIT(8),
                  2 PENDING BIT(1),
                  2 *       BIT(7),
                1 COMMAND ORG,
                  2 CODE UBIN(8),
                  2 OPERAND_LENGTH BIN(7),
                  2 OPERAND_IMAGE CHAR(*);
            TSPAN CONTROLLED (COMMAND_SCHEDULER, COMMAND_PROCESSOR, MASTER_CTRL);
   DATA EXT CSCB_QUEUE ORG;
        ELM 1 FIRST_CSCB PTR (CSCB),
              1 LAST_CSCB PTR (CSCB);
        TSPAN CONTROLLED (MASTER_CTRL);
   MODULE TYPE PGM % LOAD MODULE %;
   MODULE CONSOLE_WAIT PGM; TSPAN CONTROLLED (MASTER_CTRL); ENTRY ECW EQU ECC; END CONSOLE_
          WAIT;
   MODULE CONSOLE_IO PGM; TSPAN CONTROLLED (CONSOLE_WAIT); ENTRY ECI; END CONSOLE_IO;
   MODULE COMMAND_SCHEDULER PGM; TSPAN CONTROLLED (CONSOLE_IO); ENTRY ECS; END COMMAAND_
          SCHEDULER;

   ACCESS EXT COMMAND_SCHEDULER (SET CSCB, UPDATE CSCB_QUEUE);

   CALL EXT FROM MASTER_CTRL TO CONSOLE_WAIT;
   CALL FROM CONSOLE_WAIT TO COSOLE_IO,
        FROM CONSOLE_IO TO COMMAND_SCHEDULER;
END CONSOLE_COMMUNICATION;
```

Fig. 1 An example description by DEAPLAN

3.6 割付とインタフェイス

OS のように大規模で複雑な論理構造を持つシステムの場合，割付け情報やインタフェイス情報を明示することは，その論理構造の理解，性能予測あるいは共同作成といった点で極めて重要であろう．

DEAPLAN では，このために SPACE, MAP, ACCESS, CALL の各宣言が設けられている．(具体例は次節参照.)

スペースは現在のハードウェア構成を考慮して，REG 型(レジスタ)，MEM 型(主記憶)が標準として扱われるが，他の型も自由に定義できる．一方，割付け指示の対象となるのは，

(1) call by value 方式のパラメータ，

(2) 内部データ，

(3) 成分モジュール (ST 型，内部 PROC 型，OP 型を除く)

などである．(3)の場合はそれ自身Mと，Mに含まれるパラメータ，内部データおよび成分モジュールの全てに適用される．しかし，再指定によってMに含まれる(1)，(2)あるいは(3)は変更することができる．

ST 型，OP 型はその展開形，内部 PROC 型はその定義，を直接含む親モジュールと一体で扱われる．

インタフェイスの場合も，モジュール側は ST 型，内部 PROC 型，OP 型以外であり，またデータ側には call by value 方式のポインタパラメータで間接参照されるデータも含まれる．

4. 記述例

OS/360 風の簡単なモデル OS (MOS と名付ける) を DEAPLAN で記述したものを Fig. 1 に示す．紙数の都合で，説明は記述上のことに止める．内容的なことに関しては，たとえば文献 12) を参考にされたい．

Fig. 1 において，まず MOS のために SYSTEM なる新モジュール型が導入される．OS 全体でなく，例えばコンパイラのみを記述する場合には，SYSTEM

の代りに他の適当な名称の型を導入した方がよいであろう.

MOS は内部データや成分モジュールの単なる集合体なので, LOGIC 宣言は持たない. また MOS の入口 EOS も, 成分モジュールの一つである MASTER_CTRL の入口 EMC に受けつがれ, MOS に対するコマンドはとりもなおさず MASTER_CTRL に対するコマンドである. 次に内部データであるが, 例えば SYSIN(システム入力データ)は構造体 (ORG) データで, IN_CTRL および EXEC_CTRL なる成分モジュールによってそれぞれ動的に生成消去される. その場合 SYSIN が, SYSTEM_WORK_VOLUMES なる2次記憶媒体上に割付けられることが SPACE 宣言および MAP 宣言で明らかにされる. SPACE 宣言では, CR(カード)・DA(2次記憶)・LP(ラインプリンタ用紙)なるスペース型が新たに定義される. これに基づいて, DECKS, SHEETS, SYSTEM_RESIDENCE_VOLUME などの基底 (BASE) スペースが導入される. 基底スペースとは, 計算機システムに固定的に用意されている記憶媒体のことである.

SYSTEM_RESIDENCE_VOLUME や HCM の場合, かっこの中は容量および単位系である. たとえば前者は, 1 VOL (ボリューム)の容量を持つ. そして 1 VOL は 200 CYL(シリンダ)で, 1 CYL は 10 TRK(トラック)また 1 TRK は 3000 BYTE (バイト)である. 単位系は矛盾しない限り自由に導入できる.

MAP 宣言では, INPUT_QUEUE, OUTPUT_QUEUE などのように BDRY を用いて割付け境界を指定することもできる. さて次は, 成分モジュールであるが, MOS は TASK_CTRL, IO_CTRL, MASTER_CTRL など9種の成分モジュールを持つ. この中, TASK_CTRL および IO_CTRL は NON_TASK(非タスク, スーパバイザモード)型で, 残りは TASK (タスク, ユーザモード) 型である. そして TASK_CTRL, IO_CTRL, MASTER_CTRL は最初から最後まで存在している (STATIC) が, 残りはそれぞれ他のモジュールによって動的に生成消去される (CONTROLLED).

入口は簡単のため, すべて仮パラメータ部を省略してある. また TASK_CTRL および IO_CTRL の ENTRY 宣言において, PGM_INTERRUPT と IO_INTERRUPT はハードウェア(化されたモジュール)からの割込入口であり, 残りはいわゆるシステムマクロである. なお, DEAPLAN ではハード化された(成分) モジュールも同じように取り扱うことができる. ACCESS 宣言や CALL 宣言は図から明らかであろう. ただし, CALL 宣言の場合 ENTRY によって, その呼出しに使われる入口を明らかにすることもできる. さて次は, MOS の成分モジュールの内部構造を設計し, その各々をそれぞれ独立した記述単位として DEAPLAN で表現し, 以下これを繰返すわけであるが, ここでは簡単のため console communication の記述を MOS の後に示すにとどめる.

5. 用途

最後に DEAPLAN の応用の可能性について簡単に触れる. DEAPLAN の処理システムは今の所作られていないが, 原理的にはそれが単なるコンパイラに止まらず, テキスト編集, 結合編集, システム編集およびデバックなどの諸サポートプログラムを統合した一つの製造支援システムも含むことができよう.

しかしそれらのサポートプログラムは, すでに現在の OS の中に分離された形ではあるが存在している. したがって, 処理システムとして意義のあるのは主に設計支援を目指すものであると思われる. そして,

(1) モジュール間インタフェイスの検証,
(2) システムの改良,
(3) プロジェクト管理への利用,
(4) 設計技術の蓄積と標準化

などに用いるのが現実的なところであろう.

(1), (2)は言うまでもないが, (3)は設計作業の進捗状況を OS 記述を集中管理することによって具体的に把握しようということである. また(4)はデータやモジュールあるいはコマンドなどの名前付けを専門用語で統一的に行なうことにより, 開発済 OS の OS 記述を新 OS を設計する際に技術情報として利用しようということである. これはまた計算機による言語理解 (language understanding) という興味ある研究分野とも全く無関係ではないであろう.

6. むすび

この論文では, OS の設計用言語で必要とされる機能について考察し, そのような機能を備えたものとして DEAPLAN 言語を提案した.

この言語は特定の OS 構造を前提とせず, 以下のような特長を持っている.

(1) 手続き (procedure) よりも水準の高いモジュールあるいはモジュール型が自由に導入でき

る.

(2) 内部構造の未定義なモジュールが自由に導入できる.

(3) 内部構造の未定義な任意のデータまたはデータ型が導入できる.

(4) 効率のよい structured design を可能にする機能 (ST 型モジュールなど) を持つ.

(5) 割付方法やインタフェイスが記述できる.

この DEAPLAN を中核とした設計支援システムによって, 巨大化しつつある OS の開発[11]が容易になることが期待される.

謝辞: この論文の草稿に注意深く目を通し, 有益な助言を種々与えてくれた井上謙蔵氏† に厚くお礼を申し上げる.

† 富士通(株)ソフトウェア部, 工博

参考文献

1) Dijkstra E. W.: Notes on structured programming, EWD 249, Technical U. Eindhoven, The Netherlands (1969).
2) Dijkstra E. W.: A constructive approach to the problem of program correctness, BIT 8, pp. 174～186 (1968).
3) Wirth N.: Program Development by Stepwise Refinement, CACM, vol. 14, No. 4, pp. 221～227 (1971).
4) Snowdon P. A.: PEARL: An Interactive System for the Preparation and Validation of Structured Programs, ACM SIGPLAN Notices, vol. 7, No. 3, pp. 9～26 (1972).
5) Gries D.: Writing Programs, text for JIPDEC seminar, March 16, Tokyo, Japan (1972).
6) 電気試験所彙報: ESDL 特集号, vol. 34, No. 5 & 6 (1970).
7) 田中穂積, 張素華: 計算機システム記述用言語に関する一考察, 情報処理, vol. 12, No. 12, pp. 746～753 (1971).
8) 野口健一郎, 元岡 達: オペレーティング・システムの記述に関する考察, 情報処理, vol. 14 No. 2, pp. 98～105 (1973).
9) IBM System/360 Operating System SRL: PL/I(F) Langnage Reference Manual GC 28-8201.
10) Wulf W. A., Russel D. B. & Haberman A. N.: BLISS: A Language for Systems Programming, CACM vol. 14, No. 12, pp. 780～790 (1971).
11) Inoue K.: A problem of large scale software, The Infotech State of the Art Report 8, pp. 363～373, INFOTECH Ltd., England (1972).
12) IBM Sytem/360 Operating System SRL: Concepts and Facilities, GC 28～6535, System Programmer's Guide. GC 28～6550, Supervisor Services, GC 28～6646; Data Management Services, GC 26～3746; Supervisor and Data Management Macro Instructions, GC 28～6647.

(昭和48年2月3日受付)
(昭和48年7月3日再受付)