オペレーティング・システムの記述に関する一考察†

野口 健一郎†† 元岡 達†††

Abstract

It is pointed out sometimes that some description language of operating systems is necessary for designing operating systems and for communication purpose. It is desirable that clear description of the complex logic of operating systems is possible with the description language. To attain this, the description language must reflect the structure of operating systems.

In this paper the basic structure of operating systems is examined and a description method of operating systems is presented. The logical structure of a system is viewed as a collection of machines which are logical or physical. Each machine can be modeled based on inner states, program steps, and incoming event signals. Corresponding to this modeling, a description language is presented.

With this description method the logical structure of operating systems is described clearly. As an example of system description, part of description of an experimental TSS which was developed by authors and others is presented.

1. まえがき

計算機システムにおいて，ソフトウェア，特にオペレーティング・システム（ここでは，オペレーティング・システムとは control program の意味で用いる）の生産性の問題が解決すべき問題として指摘されている．そしてこの問題の解決のために，オペレーティング・システムの論理の適切な表現手段の必要性がいわれている．複雑なオペレーティング・システムの論理および構造を表現するための適当な設計図ともいうべきものが，情報交換の手段として，また設計の手段として求められている．

オペレーティング・システム（以下 OS と略す）の外部仕様レベルの記述は，通常日常の言語（自然語）を用いて行なわれる．自然語は柔軟性があり強力であるが，記述が論理性，厳密性に欠け，OS の論理の表現手段として適当なものであるとはいえない．OS プログラムのフローチャート，ないしはあるプログラミング言語により記述された OS プログラムは，OS の論理の厳密な記述といえる．しかし，これは処理のレベルの記述であり，各処理の意味する全体的な関連をつかみにくく，したがってまた記述の中に論理的な誤りが含まれていても発見しにくい．OS の設計手段として，また情報交換の手段として用いられる OS の表現手段（記述系）は，OS の論理を必要な程度厳密に，また OS の複雑に関連した構造を適切に表現できるものでなければならない．このためにはこの記述系は，OS の構造というものに適合しこれを反映したものである必要があろう．

OS の構造についてはこれまで多くの検討が行なわれている．OS におけるプロセス（タスクとも呼ばれる）の概念が整理された[1),2)]．プロセスはシステムの中で独立な制御の流れを持つものであり，OS の制御によって実現されるいわばロジカルなマシンである．これらは資源を共用し，またお互いの間で情報交換が行なわれる．システムはこれらのロジカルなマシンを含む系として考えることができる．

ここで述べるものは，OS の構造をひとつの観点からとらえ，それに基づいて OS を記述してみようとするひとつの試みである．システムをいくつかのロジカルなないしはフィジカルなマシンの集合としてとらえる．各マシンはいくつかの内部状態を持つものとして考える．また，マシン間には通信が行なわれる．システムをこのように構造化，すなわち分割し，それぞれ

† A Consideration on Operating System Description, by Kenichiro Noguchi (Software Works, Hitachi, Ltd.) and Tohru Moto-Oka (Faculty of Engineering, University of Tokyo)

†† 日立製作所ソフトウェア工場
††† 東京大学工学部

の要素に対して論理を記述する．この記述は，システム内のロジカルなないしはフィジカルなマシンの構造を明らかにし，またそれらの間の関連を明らかにする．ここで記述の対象となるものは，OS の論理というよりもシステムそのものの論理である．OS とはシステムのソフトウェア部分であり，システムの記述は OS の論理の記述ともなる．OS は種々の側面を持つものであるが，ここで述べる OS の記述はそのうちのいわゆる task management の部分を主対象としたものである．

以下ではまずシステムの構造について考察し，つぎにこれに基づいた OS の記述法について述べる．また応用例として，筆者らが作成した実験 TSS の記述の一部を示す．

2. システムの構造に関する考察

2.1 システムの構造

計算機システムにおけるマルチプログラミングの技術より，プロセスの概念が生まれた．マルチプログラミングされたシステム内に存在するいくつかの論理的に独立な制御の流れを区別し，それをプロセスとして扱うことによって，システムの構造が整理された[1),2)]．プロセスにはおのおの仮想の CPU が与えられていると考えればよく，これに対して，実際の CPU はシステム資源として扱われる．

プロセスの仮想 CPU はユーザ・マシンと呼ばれる．プロセスは，おのおののユーザ・マシンにおいて処理される，と考えればよい．プロセスの利用できるシステム機能は通常システム・マクロ命令の形で与えられる（メタ命令とも呼ばれる[1)]）が，これはユーザ・マシンの機械語であると考えることができる．

プロセスのユーザ・マシンは，おのおのまったく独立に動作するわけではない．プロセス間では同期の機能が利用され，また資源の共用も行なわれる．これらを果たすために，ユーザ・マシン間には情報交換の機能がある，と考えることができる．

プロセスのユーザ・マシンは，システムの論理的な構成要素である．すなわち，論理的にみればシステムを構成する独立した機械である．ユーザ・マシンの他に，システム内の入出力装置等も，独立に動作しまた他の機械と信号をやりとりする独立した機械である．システムは，これらのロジカルなないしはフィジカルな機械の集合である，としてとらえることができる．

システムに含まれる機械の構造は，内部状態および入力出力の観点から整理することができる．機械はいくつかの内部状態を区別することができる．ユーザ・マシンであれば，プロセス実行中の状態，何らかの事象待ちの状態等があり，さらに細かい要因によっていくつかの内部状態が区別される．機械に対する入力としては外部（他の機械）からの信号がある．また，プロセスのユーザ・マシンにおいては，プログラムの命令も 1 種類の入力である．機械の出力としては，システムないしはその機械の何らかの情報を変化させること，および他の機械への信号の送出，という機械の動作を対応させることができる．

以上のように，計算機システムおよびそれに含まれる（ロジカルなおよびフィジカルな）機械の構造を整理することができる．

OS の構造をプロセスの概念により整理したものは種々あるが，それらにおいてはプロセスのユーザ・マシンとしてそれぞれ特定の機能を持ったものが導入されている[1)-5)]．それに対しここではプロセスの概念およびそのもととなるシステムの基本的構造を整理し，任意の機能を持つユーザ・マシンおよびそれを含むシステムを記述する方法を述べる．たとえば，割込みの機能について，これをユーザ・マシンにも持たせたもの[4)]，あるいはユーザ・マシン間の情報交換として扱ったもの[5)]があるが，これらもここで述べる記述の対象とすることができる．

ここではシステムに含まれる個々の機械の記述に有限オートマトンの概念を応用している．OS の核の部分を有限オートマトンとして構成したシステムもあるが[6)]，それはシステム全体をひとつの有限オートマトンとして扱ったものである．

2.2 モデル化

システムを記述するためには，そのもととなるシステムの基本的な枠組みを与えること，すなわちモデル化が必要となる．2.1 での考察に基づき，システムを以下のようにモデル化する．

（1） オートマトン

システム内の独立な機械をオートマトンと名付ける．これはプロセスのユーザ・マシンのような概念的な機械であってもよい．

オートマトンは有限個の内部状態を有する．オートマトンにはプログラムを持つもの，持たぬものの 2 種類がある．

プログラムを持つオートマトンの場合，オートマトンの入力はプログラムの各命令（ステップと名付ける）

と外部からの信号（イベント信号と名付ける）の2種類がある．オートマトンは動作状態（後述する）にあれば，プログラムのステップの処理を順次行なう．（このためにオートマトンは命令カウンタを所有している．）外部からのイベント信号が到達すると，それは現在処理中のステップ（またはイベント）の処理が終了するまで記憶され，そのあとに割込んでイベント処理が行なわれる．オートマトンはステップまたはイベントをシーケンシャルに処理してゆく．

プログラムを持たぬオートマトンの場合，入力はイベント信号のみである．

ステップまたはイベントの処理に伴って，オートマトンの内部状態の遷移が行なわれる．プログラムを持つオートマトンの場合，内部状態は動作状態および休止状態の2種類のクラスに分類される．動作状態においてのみステップの処理が行なわれる．

ステップまたはイベント処理時のオートマトンの動作は，そのときの内部状態および入力の種類（ステップまたはイベントの種類）によって決定される．オートマトンの動作には次のものがある．ひとつは，システムの持つ変数ないしはそのオートマトンの持つ変数を変化させることである．もうひとつは，他のオートマトンへのイベント信号の発信である．イベント信号はオートマトン間で直接通信を行なうためのものである．オートマトン間の情報交換は，間接的にはシステムの持つ変数へのアクセスによってなされ，直接的にはイベント信号によってなされる．

オートマトンの時間的にみた動作の例を Fig. 1 に示す．これはプログラムを持つオートマトンの場合である．休止状態 (blocked status) においては，次のイベント信号が到達するまでオートマトンはその状態にとどまる．

（2） プロセス

プログラムを持つオートマトンの場合，オートマトンとプログラムとを合わせたものもひとつの単位の機械である．この単位をプロセスと呼ぶ．

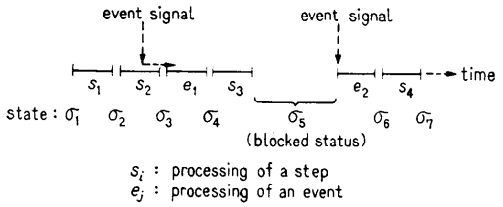

Fig. 1 An example of timing behavior of an automaton with a program.

（3） システム

いくつかのオートマトンないしはプロセスの集合がシステムである．

さらに，システムはどのオートマトンにも属さないいくつかの変数を持つ．これをパブリックな変数と呼ぶ．パブリックな変数へのアクセスは，システム内のどのオートマトンからも可能である．これに対し，各オートマトンの持つ変数はプライベートな変数と呼ばれ，これへのアクセスはそのオートマトンにしか許されない．

以上のモデル化において，オートマトンの各動作に要する時間については何も規定しない．このモデル化では，事象の時間的な順序関係のみが意味を持つものとする．

3. 記 述 言 語

システムの記述は，**2.2** で述べたシステムのモデル化に基づき，システムの各要素を記述することによって行なわれる．ここでは，記述系として閉じたものを与えることを目的として，モデル化に基づいた記述言語について述べる．

3.1 オートマトンの記述

オートマトンの記述は次のものを与えることによって行なわれる．

（1） オートマトンの名前．
（2） プライベートな変数の宣言．
（3） 状態の宣言．
（4） ステップの宣言．
（5） イベントの宣言．
（6） 状態遷移およびセマンティクスの記述．

オートマトンを区別するために名前を付ける．

オートマトンの持つレジスタ類はプライベートな変数として宣言する．なお，後の例では，プログラムを持つオートマトンの場合，命令カウンタ (IC) は暗黙のうちに定義されたプライベートな変数として扱っている．

オートマトンのとるすべての内部状態は，名前を付けて宣言する，プログラムを持つオートマトンにおいては，各状態が動作状態に属するか休止状態に属するかも示す．なお，ここでいう内部状態は，プライベートな変数の値により区別される状態は含めない．（オートマトンの動作パタンに関係するものを内部状態として宣言する．）

オートマトンのすべてのステップについて，名前

と，パラメータを有する場合には形式的パラメータとを宣言する．

オートマトンの受取るすべてのイベント信号について，名前と，パラメータを有する場合には形式的パラメータとを宣言する．

状態遷移については，ある状態においてあるステップまたはイベントが入力された結果，次にどの状態に遷移するかを記述する．

おのおのの状態遷移に対応して，オートマトンがどのような動作をするかを記述する．これはセマンティクスの記述である．セマンティクスの記述については次に述べる．

3.2 セマンティクスの記述

オートマトンの，ある状態におけるあるステップまたはイベントの実行は，ある定まった動作となる．これは，その状態におけるそのステップまたはイベントの意味，すなわちセマンティクスである．このセマンティクスを，あるセマンティクス記述言語による記述で与える．セマンティクス記述言語には，何らかのプログラミング言語を用いる．

セマンティクス記述は，**3.1** で述べた状態遷移のそれぞれに対応して与える．セマンティクス記述では，オートマトンのプライベートな変数の変化，システムのパブリックな変数の変化等を記述する．また，他オートマトンへのイベント信号の発信も，セマンティクス記述の中で記述する．イベント信号の発信の記述においては，相手オートマトン名とイベント名とを指定する．

後の例では，セマンティクス記述言語として，PL/I にいくつかの関数を導入したものを用いている．

3.3 プロセスの記述

プロセスの記述は次のものを与えることによって行なわれる．

（1） オートマトンの記述．

（2） プログラムの記述．

プロセスのプログラムは，プロセスのオートマトンの機械語によって記述される．すなわち，オートマトンの記述において定義されたステップを用いて記述される．

3.4 システムの記述

システムの記述は次のものを与えることによって行なわれる．

（1） パブリックな変数の宣言．

（2） 含まれるプロセス及びオートマトンの記述．

3.5 記 述 例

記述言語による記述を例を用いて示す．

例として記述されるものは n 個の端末を含むシステムであり，TSS システムを単純化したものである．各端末からの信号は簡単のためクイット信号のみとする．ユーザ・マシンにおいて，端末からのクイット信号は割込みとして処理される．ユーザ・マシンには，クイット信号による割込みをマスクする機能がある．このシステムを記述したものが Fig. 2 である．また，これを図式的に記述したものが Fig. 3 である．

```
system STSS;
 I=1: n
  {automaton   UM(I);
    state      A(S1, S2, S3), B(S0);
    step       SQM,
               CQM(P): P FIXED,
               LOGOUT;
    event      QUIT;
    semantics
      S0×QUIT→S1: BEGIN; IC=ENTRY; END;
      (S1, S2, S3)×SQM→(S1, S2, S1):;
      (S1, S3)×CQM(P)→(S1, S1): BEGIN; IC=P; END;
      S2×CQM(P)→S1: BEGIN; LOC(0)=P; IC=2; END;
      (S1, S2)×QUIT→(S2, S1):;
      S3×QUIT→S1: BEGIN; LOC(0)=IC; IC=2; END;
      (S1, S2, S3)×LOGOUT →(S0, S0, S0):;
    automatonend; }
 I=1: n
  {automaton   TERMINAL(I)/*HARDWARE*/;
    state      S;
    event      QUIT_KEY;
    semantics
      S×QUIT_KEY→S: BEGIN; EVENT(QUIT, UM(I));
      END;
    automatonend; }
 I=1: n
  {automaton   USER(I)/*HUMAN*/;
    state      S;
    step       DEPRESS_QUIT_KEY;
    semantics
      S×DEPRESS_QUIT_KEY→S: BEGIN;
      EVENT (QUIT_KEY, TERMINAL(I)); END;
  automatonend; }
systemend;
```

Fig. 2 An example of system description. (Simplified TSS.)

Fig. 2 において，ユーザ・マシン (UM)，端末 (TERMINAL)，および端末ユーザ (USER) がオートマトンとして記述され，かつそれぞれ n 組あることが示されている．オートマトン UM は4つの状態を持ち，S1～S3 は動作状態である．オートマトン UM は3つのステップと1つのイベントを持つ．(UM のその他の状態変化に関係せぬ命令は，ステップとして宣言することは省略する．) **semantics** において，状

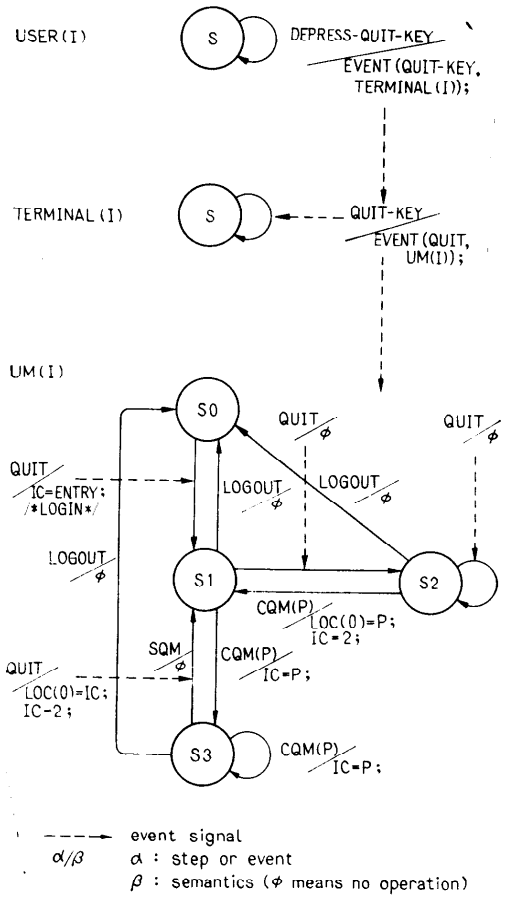

Fig. 3 Graphical representation of system description. (Simplified TSS.)

態遷移とともにそれに伴うセマンティクスが記述されている．状態遷移は，同一の入力に対して同じセマンティクスを持つものはまとめて示されている．セマンティクス記述中，LOC(0) はメモリの0番地を示す．状態遷移のみで，セマンティクス動作が何もない場合もある．

クイット信号は，イベント信号として記述されている．これは USER から TERMINAL へ，および TERMINAL から UM へと伝達される．TERMINAL および USER のセマンティクス記述中の EVENT (e, a); はオートマトン a へイベント信号 e を発信することを意味している．

オートマトン UM は対応するプログラムと共にプロセスを形成するものであるが，この記述ではプログラムは示されていない．この記述では，OS によって提供されるユーザ・マシンのみでなく，TERMINAL および USER を記述の中に含めている．これによりシステムの記述を閉じたものとしている．

Fig. 3 は Fig. 2 を図式的に表現したものであり，各オートマトンの状態遷移図とステップおよびイベントの関係が示されている．図式表現により，オートマトンの構造およびシステム全体の関連が分かりやすく表現される．

4. OS の記述と設計

ここでは **3** で与えた記述言語により記述されたものの持つ意味，およびこの記述言語と OS 設計との関連について述べる．

この記述言語により記述されるものは，システムの論理的な構造である．記述されたシステムがいかにして実現されるかは，記述とは別の問題である．記述に含まれるオートマトンがハードウェアとして実現されるか，ないしは何らかの形でソフトウェア的に実現されるかは，そのシステムの実現方法によっている．

この記述言語に基づいて OS の設計を考えると次のようになる．この記述言語により，システムの外部仕様レベルの記述が行なわれる．この外部仕様レベルの記述に対して，OS の具体的な内部処理の論理を付加することにより，詳細なレベルの記述とすることができる．このように，システムの外部仕様の記述を行ない，それを内部処理の論理を付加した記述へと変換してゆくこととして，OSの設計を考えることができる．たとえば，**3.5** の例はシステムの外部仕様レベルの記述である．このユーザ・マシンの記述に対して，CPU の管理（スケジューリング）の記述を追加することができる．これにより，ユーザ・マシンの各動作状態はランニング状態とレディ状態(これは休止状態となる)へと分化する．

なお，この記述言語による記述と実際の OS プログラムとの関係は次のようになる．プログラムを持つオートマトンが実際の CPU に等しいレベルが最も詳細な記述レベルであるが，OS プログラムはそのレベル

Table 1 Variables of an experimental TSS concerning user machine description.

	Name	Type	Comment
1.	Public variables		
	FBPS	SET	Facility blocked process set
	FBUSY	BIT(1)	Facility busy indicator
2.	Private variables of UM		
	II	BIT(16)	Interruption indicator
	DREC	FIXED	Drum request count

Table 2 Steps and events of a user machine of an experimental TSS.

Name	Comment
1. Steps	
(1) Meta-instructions	
EXTIO	Execute terminal I/O
GETER (L, IOAP)	Get from terminal
PUTER (L, IOAP)	Put to terminal
PGETER (PL, PIOAP, GL, GIOAP)	Put and get to/from terminal
EXCP (IOBP)	Execute channel program
WAIT	Wait for drum I/O completion
SEIZE	Seize facility
RELEASE	Release facility
LOGOUT	Logout
SIM	Set interruption mask
CIM(P)	Clear interruption mask
(2) Internal interruption	
EXCOND(I)	Exceptional conditions (Overflow etc.)
2. Events	
QUIT	Quit
TIOCOMP	Terminal I/O completion
DIOCOMP	Drum I/O completion
FWAKEUP	Facility wakeup

Table 3 States of a user machine of an experimental TSS with their interpretation.

State No.	Active	TB	DB	FB	IM	II	DU	FU
1								
2	∨				∨			
3	∨				∨		∨	
4	∨				∨	∨		
5	∨				∨	∨	∨	
6	∨							
7	∨						∨	
8	∨				∨			∨
9	∨				∨		∨	∨
10	∨				∨	∨		∨
11	∨				∨	∨	∨	∨
12		∨			∨			
13		∨			∨		∨	
14		∨			∨	∨		
15		∨			∨	∨	∨	
16		∨						
17		∨					∨	
18		∨				∨		
19		∨				∨	∨	
20				∨	∨			
21				∨	∨		∨	
22				∨	∨	∨		
23				∨	∨	∨	∨	
24			∨		∨		∨	
25			∨		∨	∨	∨	
26			∨				∨	
27			∨			∨	∨	
28			∨		∨		∨	∨
29			∨		∨	∨	∨	∨

TB Terminal blocked.
DB Drum blocked.
FB Facility blocked.
IM Interruptions are masked.
II Some bits of interruption indicator are ON.
DU Drum using.
FU Facility using.

の記述中に含まれるものである.

この記述言語による記述においては，上で述べたように論理の記述のレベルを適当に選ぶことができる.

Table 4 State transition of a user machine of an experimental TSS.

State \ Step & event	EXTIO	EXCP	WAIT	SEIZE	RELEASE	LOGOUT	SIM	CIM	EXCOND	QUIT	TIOCOMP	DIOCOMP	FWAKEUP
1	—	—	—	—	—	—	—	—	—	2	—	—	—
2	12	3	φ	8/20	φ	1	φ	6	4	4	—	—	—
3	13	3	24	9/21	φ	φ	φ	7	5	5	—	2/3	—
4	14	5	φ	10/22	φ	1	φ	2	4	4	—	—	—
5	15	5	25	11/23	φ	φ	φ	3	5	5	—	4/5	—
6	16	7	φ	φ	φ	1	2	φ	2	2	—	—	—
7	17	7	26	φ	φ	φ	3	φ	3	3	—	6/7	—
8	φ	9	φ	φ	2	φ	φ	φ	10	10	—	—	—
9	φ	9	28	φ	3	φ	φ	φ	11	11	—	8/9	—
10	φ	11	φ	φ	4	φ	φ	φ	10	10	—	—	—
11	φ	11	29	φ	5	φ	φ	φ	11	11	—	10/11	—
12	—	—	—	—	—	—	—	—	—	14	2	—	—
13	—	—	—	—	—	—	—	—	—	15	3	12/13	—
14	—	—	—	—	—	—	—	—	—	14	4	—	—
15	—	—	—	—	—	—	—	—	—	15	5	14/15	—
16	—	—	—	—	—	—	—	—	—	18	6	—	—
17	—	—	—	—	—	—	—	—	—	19	7	16/17	—
18	—	—	—	—	—	—	—	—	—	18	2	—	—
19	—	—	—	—	—	—	—	—	—	19	3	18/19	—
20	—	—	—	—	—	—	—	—	—	22	—	—	8
21	—	—	—	—	—	—	—	—	—	23	—	20/21	9
22	—	—	—	—	—	—	—	—	—	22	—	—	10
23	—	—	—	—	—	—	—	—	—	23	—	22/23	11
24	—	—	—	—	—	—	—	—	—	25	—	2/24	—
25	—	—	—	—	—	—	—	—	—	25	—	4/25	—
26	—	—	—	—	—	—	—	—	—	27	—	6/26	—
27	—	—	—	—	—	—	—	—	—	27	—	2/27	—
28	—	—	—	—	—	—	—	—	—	29	—	8/28	—
29	—	—	—	—	—	—	—	—	—	29	—	10/29	—

— The case does not exist.
φ No state transition and semantically no operation.
/ Conditional transition.
SEIZE ……〔FBUSY=0〕/〔FBUSY=1〕
DIOCOMP……〔DREC=1〕/〔DREC>1〕

さらに，記述の範囲についても，ひとつのオートマトンの記述からシステムの全要素の記述まで，適当な範囲を含めることができる.

5. 実験 TSS の記述

記述言語の実システムへの適用例として，筆者らが作成した小形計算機を用いた実験 TSS に関する記述を示す．ここでは TSS のユーザ・マシン，すなわち TSS のユーザ・プロセスに対する仮想 CPU について，外部仕様レベルの記述を示す．システムのその他の要素の記述は省略する.

記述は，オートマトンの記述の各要素について表形式で示す．Table 1 にユーザ・マシンの記述に関連するパブリック変数，およびユーザ・マシンのプライベート変数を示す．Table 2 にユーザ・マシンのステッ

Table 5 Semantic description of a user machine of an experimental TSS.

Transition	Semantics
(1) EXTIO (GETER/PUTER/PGETER)	
2→12 3→13 4→14 5→15 6→16 7→17	EVENT({CALL(L, IOAP), GETER_PP {CALL(L, IOAP), PUTER_PP {CALL(PL, PIOAP, GL, GIOAP), PGETER_PP);
(2) EXCP (IOBP)	
all cases	DREC=DREC+1; EVENT(CALL(IOBP), EXCP_PP);
(3) WAIT	
3→24 5→25 7→26 9→28 11→29	ϕ (semantically no operation)
(4.1) SEIZE [FBUSY=0]	
2→8 3→9 4→10 5→11	FBUSY=1;
(4.2) SEIZE [FBUSY=1]	
2→20 3→21 4→22 5→23	JOIN(*, FBPS);
(5) RELEASE	
8→2 9→3 10→4 11→5	T=SELECT(FBPS); IF T¬=0 THEN DO; REMOVE(T, FBPS); EVENT(FWAKEUP, T); END; ELSE FBUSY=0;
(6) LOGOUT	
2→1 4→1 6→1	ϕ
(7) SIM	
6→2 7→3	ϕ
(8) CIM (P)	
2→6 3→7	IC=P;
4→2 5→3	LOC(0)=P; LOC(1)=II; II=0; IC=2;
(9) EXCOND (I)	
6→2 7→3	II(I)=1; LOC(0)=IC; LOC(1)=II; II=0; IC=2;
other cases	II(I)=1;
(10) QUIT	
1→2	IC=ENTRY;
6→2 7→3	II(0)=1; LOC(0)=IC; LOC(1)=II; II=0; IC=2;
other cases	II(0)=1;
(11) TIOCOMP	
18→2 19→3	LOC(0)=IC; LOC(1)=II; II=0; IC=2;
other cases	ϕ
(12.1) DIOCOMP [DREC=1]	
27→2	DREC=0; LOC(0)=IC; LOC(1)=II; II=0; IC=2;
other cases	DREC=0;
(12.2) DIOCOMP [DREC>1]	
all cases	DREC=DREC−1;
(13) FWAKEUP	
all cases	ϕ

プとイベントを示す．CPU の一般の命令語はステップとして示すことを省略する．Table 3 にユーザ・マシンの内部状態をおのおのの意味と共に示す．Table 4 に状態遷移を示す．状態遷移は条件付きで行なわれる場合がある．Table 5 にセマンティクス記述を，ステップまたはイベントとそれに伴う状態遷移ごとに示す．

セマンティクス記述について説明を加える．GERTE-PP，PUTER-PP，PGETER-PP および EXCP-PP はシステムのプロセス（に対するオートマトン）であり，ユーザ・マシンにより起動される．これらは端末入出力ないしはドラム入出力をつかさどる．イベント信号 TIOCOMP，DIOCOMP はこれらから送られてくる動作完了の信号である．

JOIN，SELECT，REMOVE はセマンティクス記述用の関数である．JOIN は第1パラメータで指定された要素を第2パラメータの示す集合へ入れること，SELECT は集合よりひとつの要素を選択すること（無ければ0），REMOVE は第1パラメータで指定された要素を第2パラメータの示す集合より取去ることを意味している．また＊はそのオートマトン自身を意味する変数である．

ユーザ・マシンの機能について簡単に説明する，端末入出力の起動はメタ命令 GETER，PUTER，PGETER で，ドラム入出力の起動はメタ命令 EXCP で行なわれる．ドラム入出力の完了は WAIT で受取る．WAIT はすべてのドラム入出力の完了まで待つという単純な機能のみ有する．ユーザ・プロセスは，

システムでシリアルな処理を実行するためにメタ命令 SEIZE を発行する．SEIZE を出して動作可能なユーザ・プロセスは一時にはひとつだけである．クイットにより，ないしは内部割込みにより，ユーザ・マシンは割込まれる．

ここではユーザ・マシンの記述を表形式で示したが，これを状態遷移図に基づく図式表現で表わすことにより，ユーザ・マシンの論理的構造が分かりやすく表現される．

なお，ここで示したものはユーザ・マシンについての記述のみであるが，この他のシステムの構成要素，すなわちいくつかのシステム・プロセス，入出力装置についても記述が行なわれている[8]．また，実験 TSS の外部仕様の記述のみでなく，スケジューリングおよびスワッピングのアルゴリズムまで含めたシステムの内部仕様の記述も行なわれている[8]．

6. むすび

この論文では，OS の論理の記述という課題に対し，システムの構造についてかなり簡単な枠組みをもうけ，この枠組みの中で OS（システム）の記述を行なうという試みを述べた．この記述方式は **5** で述べた実験 TSS の整理——システムの論理の落ちのない記述およびスケジューリング・アルゴリズムの正確な記述——に有効であった．なお，ここで述べた記述言語は単に記述用のものであり，シミュレーション，コンパイル等のための処理プログラムは今のところ作られていない．

ここで述べた記述方式は，特定の OS 構造を前提とせず，任意の構造の OS をこの記述方式により記述（定義）することを意図している．OS 記述に対してここでのものと類似の動機を持ったシステム記述言語が文献 7）に提案されている．それと比較し，ここで述べた記述方式はシステムの構造に関する枠組みがより basic である．

ここで述べた OS の記述は，OS の仕様レベルの記述すなわち OS の論理の記述を主対象としたものである．この方向のアプローチは，当面は OS の論理の整理ないしは OS シミュレーションのためのもの，ということができよう．これに対し，OS プログラムを対象としこれを何らかの高級言語で記述しようということは，OS 作成の面からの実際的アプローチである．OS の記述に対し，このそれぞれの面からの取組みおよびその総合が今後必要であると考える．

謝　辞

この論文は東京大学工学部電気工学科元岡研究室において行なわれた研究に基づいている．実験 TSS の作成にあたっては日本ソフトウェア株式会社，富士通株式会社，および元岡研究室の方々にご協力いただいた．ここに深く感謝する次第である．

参考文献

1) J. B. Dennis and E. C. Van Horn: Programming semantics for multiprogrammed computations, Comm. ACM, Vol. 9, No. 3, pp. 143～155 (1966).
2) J. H. Saltzer: Traffic control in a multiplexed computer system, MAC-TR-30, M. I. T. (1966).
3) E. W. Dijkstra: Co-operating sequential processes, in Programming Languages (F. Genuys, ed.), Academic Press, pp. 44～112 (1968).
4) 淵一博，田中穂積：ETSS における GPP-Supervisor，電気試験所彙報，ETSS 特集号，Vol. 32, No. 8, pp. 750～760 (1968).
5) 高橋秀俊，亀田壽夫：オペレーティングシステムの一構成法，情報処理，Vol. 11, No. 1, pp. 20～31 (1970).
6) R. E. Heistand: An executive system implemented as a finite-state automaton, Comm. ACM, Vol. 7, No. 11, pp. 669～677 (1964).
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：計算機オペレーティング・システムの設計法，東京大学学位論文 (1969).
9) 野口健一郎：オペレーティング・システムの記述と設計法，情報処理学会 OS シンポジウム報告集，pp. 84～102 (1970).

(昭和 47 年 4 月 4 日受　付)
(昭和 47 年 6 月 29 日再受付)