ＯＳ/ｏ のアーキテクチャと第一版の実現

高橋延匡　並木美太郎　武山潤一郎　中川正樹
（東京農工大学　工学部　数理情報工学科）

１．　はじめに

我が国は、半導体産業に於ては世界のトップレベルに立つに至った。半導体技術に依存して計算機産業も目覚しい発展をとげつつある。特に、パーソナル・コンピュータや日本語ワードプロセッサなどはオフィス・オートメーションやホーム・オートメーション或いはパーソナル・オートメーションなどの発想から、その市場規模の伸びは著しいものがある。ひるがえって、そのようなパーソナル・コンピュータをみる時、マイクロプロセッサのアーキテクチャからそのソフトウェアに至るまで、ほとんど米国で開発されたものである。特にその中核を成すオペレーティング・システム（ＯＳ）は米国の数社のものに独占されている。

その理由は、種々あると思う。たとえば、ユーザがそれを欲しているというのも一つの理由であろうし、パーソナル・コンピュータを米国に輸出する場合に都合が良いという事情もあろう。また、ＯＳが標準化されていればいる程、そのうえの応用ソフトウェアは作り易いし、ソフトウェア・ハウスなどマーケット戦略上も好ましい。その一例はIBM5550のブランドの強さが物語っている。

我々は、大学で計算機科学の教育にたずさわる立場では、ＯＳがブラック・ボックスでは好ましくない。特に、設計者を養成する立場では汎用大形計算機のＯＳはその規模の点でも教育をする対象としては全く不向きである。一方、パーソナル・コンピュータ用のそれは、その全貌が把握できる範囲にもかかわらず、これもブラック・ボックス化しているし、又、そのＯＳ自身も必ずしも情報科学の経験が生かされていない面がある。

一方、計算機科学の研究にたずさわる立場になると、もっと根本的な問題も含む。何故なら、ＯＳはその開発された社会環境（マネジメントの環境）を色濃く反映しているからである。たとえば、米国では、原則的に「日本語情報処理」の必要性はない。そのためのオーバーヘッドを最初からＯＳに内蔵させることはあり得ない。ところが、我が国では、「日本語」中心の文化であり、そのような各種の処理がＯＳの標準装備として普通に行なえるのが常識である。にもかかわらず、汎用ＯＳでは最近までそのような常識を持ち得なかったのが現実である。又、管理体制（マネジメント）は管理対象によって変化させるのが常識である。にもかかわらず、現にそのＯＳで支援していることしか利用できない、すなわち新しいマンマシンインタフェース用の機器などは追加できない、という結果になっている。特に、研究用システムを考えた場合このような制約は致命的であると考える。

我々は上記の理由から、「研究用計算機システム」の開発の必要性を痛感した。この研究用計算機システムは、研究対象として、

（１）人工知能の研究
（２）日本語情報処理の研究
（３）特にオンライン手書き文字認識の研究
（４）計算機を用いた教育の研究
（５）計算機システム、特に並列処理を含んだＯＳ自身の研究

などがあり、それらの使用に耐える必要がある。

本報告は、1980年2月より開始された学科内プロジェクト（プロジェクトPIE）用の初期のシングルプロセッサ用の核となるＯＳ/ｏ(omicron)の第一版の実現について論ずる。これは長い道程の第一歩にすぎない。

２．　ＯＳ/ｏ の開発環境

２.１　ハードウェア環境

ＯＳ/ｏ 開発の現在のハードウェア環境を図１に、そしてＯＳ/ｏ を実際に稼働させ、オンライン手書き文字認識、日本語情報処理などの研究を行なう研究用システムを図２に示す。ＯＳ/ｏ は、この他、図２のなかのレーザ・ビーム・プリンタ用インテリジェント・インターフェース／コントロール、パタン認識・人工知能を目的としたマルチ・マイクロプロセッサ・アーキテクチャ／πにも、それぞれ縮小版、拡張版としてインプリメントすることを計画している。

２.２　ソフトウェアの開発環境

開発システムのＯＳはCP/M-68Kである。開発用のコンパイラはCP68、C068、C168、AS68、アセンブラはAS68、リンカはLO68、さらにライブラリの管理はAR68を用いた。

当初、このCP/M-68KはBIOS(Basic I/O System)の影響で両面単密のディスケットしか使えないなどの欠点があった。そこで、我々はBIOSを書き換え、両面倍密のディスケットを使用可能とし、さらにLSIファイル(メモリの一部をディスクとして使用する[８,９])も導入し、開発効率の向上を図った。

CP/M-68Kにはデバッギングツールとして DDT(Dynamic Debugging Tool)、NM68(シンボリック表の出力)などがある。しかし、プログラム管理のツールなどは無い。そこで、クロスリファレンスリスト、コーリングストラクチャジェネレータなどのツールの他、モジュール管理のツールも作成した(表１はこのツールを使用した)。デバッギングツールだけでなく、これらの管理ツールも生産性向上に貢献した。

また、ＯＳ/ｏをターゲットマシンに移植するために、ロードモジュールコンバータ、ファイルコンバータを作成したことは言うまでもない。

3．ＯＳ/ｏ 第一版のソフトウェア構成

ＯＳ/ｏ 第一版は、ＯＳ自身を開発することを主な目的として、基本的な部分を実現した。ＯＳ/ｏ の現状においてのモジュール構成を表１に示す。

ＯＳ/ｏ は、通常のタスクの概念と、いくつかのタスクが１つの実行環境を共有するタスクフォースの概念を導入している[6,10]。ＯＳ/ｏ 第一版では、この概念を割込み処理に、部分的に実現した。割込みが起きるごとに、それに対応する割込み処理をタスクとして、生成・起動し、処理を行なっている。255 種の割込みを直接受け付けるのが、d_excp_entモジュールである。ここでは、レジスタの仮の退避、ＯＳの実行環境の設定を行なっている。その後、制御をexcp_entryモジュールに渡し、割り込まれたタスクの状態をタスク管理表に退避する。割込みに対応する処理ルーチンをタスク（タスクフォースメンバ）として、生成・起動し、処理を行なう。割込み処理終了にあたって、exit_excpを用い、このタスクを終了・消滅している。タスクスィッチングを行なっているルーチンは、dispatchであり、アセンブリ言語で記述されている。

メモリ管理では、特定領域、任意領域のメモリ要求に合わせて、いくつかのメモリ管理表のアクセス関数を持っている。この部分がmmt_acceモジュールであり、それを補助しているのがmmt_ac_auxモジュールである。タスク管理は、alloc_freeモジュールで、ファイルシステムは、 mm_fsysモジュールでメモリ管理とインタフェースをとっている。

ファイルシステムの各処理モジュールは、ほとんどがファイルに関するＳＶＣに対応している。ディレクトリサーチ（dirsrch)や、表の操作管理モジュールは、各処理モジュールで共通に使用されている。I/O ファイルの管理は、fioモジュールでなされ、具体的な入出力の処理は、sopen, sclose, sread, swrite, slseek より、 biocallを通じ、I/O管理のモジュールを呼び出すことによって実現されている。 I/O管理内でハンドラを呼び出し、物理的なI/O 処理を行なっている。このようにして入出力機器をファイルとして扱っている。

現在、ＯＳ/ｏ は、開発システム上で、稼動している。コマンドインタプリタを除いたＯＳの核は、ＯＳの使用するスタック等を含めて、 152Kバイトのメモリを使用している。

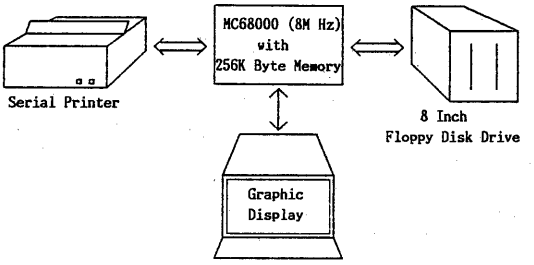

図１　ハードウェア構成（開発システム）

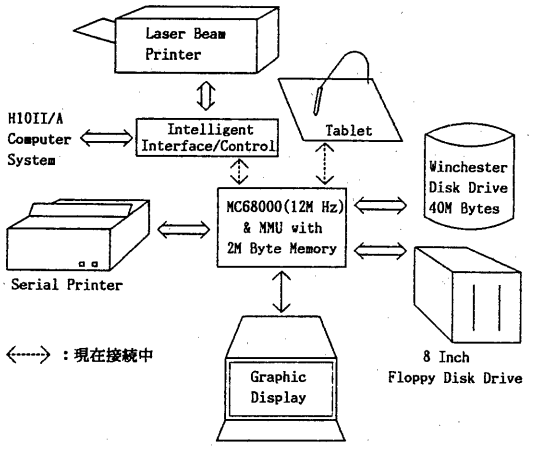

図２　ハードウェア構成（ターゲット・システム）

ＯＳ/ｏ のシステムソフトウェアとして、LISPインタプリタ、言語Ｃコンパイラ[５]等の言語処理系、テキストエディタ、リンケージエディタ[11]等のユーティリティ・ソフトウェアも同時に開発を進めてきた。表２に各ソフトウェアのサイズ等を示す。

表１　ＯＳ/ｏ モジュール表

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト) 手続き	データ	計	備　考
タスク管理	310	C assembly	1588 3013	22.5K	21.2K	43.7K	表１(a)参照
ファイルシステム	113	C	3341	25.0K	2.6K	27.6K	表１(b)参照
メモリ管理	19	C	793	4.7K	1.3K	6.0K	表１(c)参照
I/O 管理	4	C	75	0.4K	0.9K	1.3K	表１(d)参照
ハンドラ	32	C	903	5.5K	0.4K	5.9K	表１(e)参照
コマンドインタプリタ	84	C assembly	2027 311	16.2K	1.6K	17.8K	表１(f)参照
計	562	C assembly	8727 3324	74.3K	28.0K	102.3K	

表1(a)　ＯＳ/ｏモジュール表(タスク管理)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
task_trans	create	C	253				タスク生成
	execute	C	17				タスク起動
	suspend	C	21	3018	0		タスク中断
	terminate	C	17			3654	タスク終了
	disappear	C	30				タスク消滅
	dispatch	assembly	176	636	0		タスク切換え
loader	load	C	268	1626	48	1674	ローダ
	modify_data						ロード時アドレス修正
tmt_mng	4	C	66	348	2	350	タスク管理表エントリ管理
tmt_acce	2	C	92	846	0	846	管理表アクセス関数
d_excp_ent	255	assembly	2805	8128	0	8128	RAW割込み受付
excp_entry	2	C	210	3992	0	3992	論理割込み受付
excp_proc	6	C	159	1372	184	1556	割込み処理ルーチン
excp_exit	2	C	27	144	0	144	割込み出口ルーチン
emt_mng	6	C	141	982	0	982	割込み管理表エントリ管理
sys_error	sys_err　他 4	C	69	430	62	492	システムエラー処理
tm_fsys	4	C	51	302	2	304	ファイルシステム・インタフェース
ptr_conv	6	C	81	656	0	656	ポインタ変換
tm_etc	6	C	86	608	0	608	雑関数
	4	assembly	32				
TMT					2688	2688	タスク管理表(現在 16エントリ)
EMT					18700	18700	割込み管理表(255エントリ)
計	310	C	1588	23088	21686	44774	
		assembly	3013				

表1(b)　ＯＳ/ｏモジュール表(ファイルシステム)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
sopen	5	C	225	1562	0	1562	ファイル・オープン
sclose	3	C	85	760	0	760	ファイル・クローズ
sread	3	C	148	1190	0	1190	ファイル・リード
swrite	3	C	152	1228	0	1228	ファイル・ライト
slseek	3	C	117	1628	0	1628	
allclose	1	C	14	76	0	76	
adskflsh	2	C	38	332	0	332	
sbioexc	2	C	86	482	0	482	
iplmnp	5	C	27	60	0	60	
sinit	7	C	234	1884	0	1884	ディスクの初期化
smount	1	C	183	1552	0	1552	ディスクのマウント
dismount	2	C	77	760	0	760	ディスクのディスマウント
fio	2	C	111	774	0	774	I/O ファイルの管理
screat	1	C	163	1654	0	1654	ディレクトリ、ファイルの生成
sdelete	5	C	217	1978	0	1978	ディレクトリ、ファイルの消去
tpchk	1	C	15	52	0	52	
srename	2	C	116	902	0	902	名前の付換え
excchk	1	C	51	438	0	438	
dircopy	1	C	45	284	0	284	デレクトリ情報のコピー
undcopy	1	C	66	570	0	570	デレクトリ情報のコピー
dmtstat	2	C	75	556	0	556	ディスク・マウント表のコピー
setwdir	1	C	27	132	0	132	ワーキングデイレクトリのセット
namewdir	4	C	101	636	0	636	ワーキングデイレクトリ名の取出し
workmnp	3	C	23	74	4	78	
dirsrch	5	C	136	936	0	936	ディレクトリ・サーチ
dut	14	C	240	1624	0	1624	表操作
freepage	9	C	138	956	16	972	表操作
hndaft	4	C	40	238	0	238	表操作
hnddmt	4	C	42	200	0	200	表操作
hndnmlst	4	C	151	1056	0	1056	ファイル名切出し
alcdir	2	C	23	68	0	68	
capa	4	C	61	432	0	432	
dskio	5	C	89	438	0	438	
alloc	2	C	17	54	0	54	
iocall	1	C	8	24	0	24	
AFT					2080	2080	アクセス・ファイル表(現在32エントリ)
DMT					512	512	ディスク・マウント表(現在８エントリ)
計	113	C	3341	25590	2612	28202	

表1(c)　OS/oモジュール表(メモリ管理)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
mmt_mng	init_mmt 他3	C	127	766	28	794	メモリ管理表エントリ管理
mmt_acce	srch_free 他2	C	264	1558	0	1558	メモリ管理表アクセス関数
mmt_ac_aux	find_bro 他2	C	105	472	0	402	メモリ管理表アクセス補助関数
alloc_free	get_mm 他1	C	70	356	28	384	メモリ割付・解放処理
heap_mng	get_heap 他1	C	87	754	0	754	ヒープ領域割付・解放処理
mm_fsys	getbuf 他4	C	140	934	56	990	ファイルシステム・インタフェース
	MMT					1216	1216 メモリ管理表(現在 32エントリ)
計	19	C	793	4840	1328	6168	

表1(d)　OS/oモジュール表(I/O 管理)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
initiotb	1	C	14	66	0	66	
ioattach	1	C	23	98	0	98	
biocall	1	C	32	196	0	196	
astrncp	1	C	6	28	0	28	
IO					928	928	I／O管理表
計	4	C	75	388	928	1316	

表1(e)　OS/oモジュール表(ハンドラ)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
useretc	3	C	46	348	124	474	
fdc	8	C	305	1832	206	2038	
prtr	4	C	91	516	0	516	
cnsl	6	C	200	1074	64	1138	
mem	11	C	261	1832	60	1892	
計	32	C	903	5602	454	6056	

表1(f)　OS/oモジュール表(コマンドインタプリタ)

モジュール名	モジュールメンバまたはメンバ数	開発言語	行数	モジュールサイズ(バイト)			備考
				手続き	データ	計	
main	4	C	93	964	18	982	
loadgo	2	C	100	816	52	868	
mount	1	C	34	320	84	404	
dismount	1	C	34	312	68	380	
cfd	4	C	93	882	180	1062	
delete	1	C	20	158	46	208	
l	5	C	126	1128	112	1244	
h	5	C	145	1218	96	1314	
d	4	C	111	1298	160	1458	
dmt	2	C	55	862	50	912	
syscall		assembly	311	760	0	760	
cnvstd	2	C	86	620	52	672	
iostd	4	C	52	246	24	270	
prmgt	4	C	92	478	0	478	
io	11	C	225	1672	2	1674	
cutcmd	1	C	126	1414	8	1422	
mem	7	C	125	786	0	786	
charmnp	9	C	116	772	0	772	
flgmnp	2	C	42	218	102	320	
lrcpy	2	C	21	130	2	132	
numcnv	6	C	133	906	0	906	
sbnmcnv	6	C	101	588	0	588	
aafcreat	1	C	6	30	0	30	
extdef	0	C	84	0	550	550	
計	84	C assembly	2027 311	16578	1606	18184	

4．　CP/M モジュールからＯＳ/ｏ モジュールへの移行

既存のＯＳを利用して、新しいシステムを開発するには、いわゆるクロス・ソフトウェアを利用するのが能率的である。我々の場合、

（1）Ｃコンパイラの開発は東大大型センタのVAX/UNIX上で開始した（1983年4月）。

（2）その後、CP/M-68Kを研究室で入手したので(1983年8月)、ＯＳ/ｏ の制御プログラムは、CP/M-68Kの環境上で開発することにした。

現実に、CP/M-68Kを用いてＯＳ/ｏ の実行環境を提供するには種々の工夫が必要になる。図３に、その過程を詳細を示す。しかし、ここで問題なのは、オブジェクト・プログラムのレジスタの割付けなどアドレシング方式まで、ＣＰ/M 形式からＯＳ/ｏ 形式に変換することが難しいということである。CP/M-68Kは絶対アドレシングを用いた実行環境を設定し、CP/M-68K上のコンパイラ、アセンブラ、リンカは、絶対アドレシングのコードを生成している。一方、ＯＳ/ｏ は、文献［11］で示すようにベースレジスタからの変位による相対アドレシングを用いて、リロケータブルでリエンタラブルな実行環境を提供している。このことから、ＯＳ/ｏ をCP/M-68K上で開発すれば、ＯＳ/ｏ 自身、絶対アドレシングを行ない、しかも絶対アドレシングを行なうモジュール（ＯＳ/ｏ のリンカ、アセンブラ、テキストエディタなど）の実行環境をも提供しなければならなかった。なお、図３の「モジュール変換」はCP/M-68Kのロードモジュール形式からＯＳ/ｏ のロードモジュール形式への変換を行なう。「ファイル変換」はＯＳ/ｏ のファイルシステムの形式への変換を行なう。図３の２はＯＳ/ｏ 第一版を用いて（ＣＰ/Mモジュールを排除した）ＯＳ/ｏ を完成する手順を示している。

表２　ＯＳ/ｏ システムソフトウェア

ソフトウェア	開発言語	ソースプログラム行数
言語Ｃコンパイラ+	Ｃ	18,356
LISPインタプリタ+	Ｃ	3,120
アセンブラ	Ｃ	3,670
テキストエディタ	Ｃ	955
言語Ｃプリティプリンタ	Ｃ	1,082
リンケージエディタ	Ｃ	880

+ LISPインタプリタは、東大大型計算機センタVAX/UNIXで開発された。言語Ｃコンパイラも同システムで開発している。現在、各モジュールの単体テストを終了し、総合テストを行なっている。

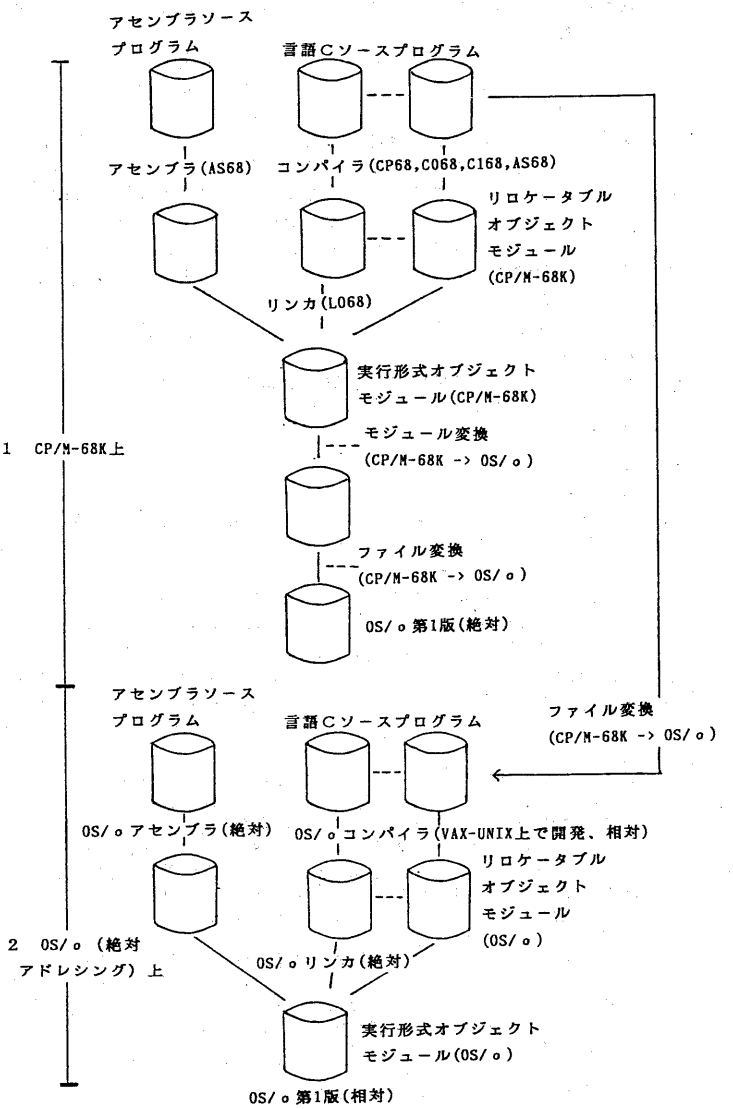

図３　ＣＰ/M モジュールからＯＳ/ｏ モジュールへの移行

5．　ＯＳ/ｏ 第一版の特徴

ＯＳ/ｏ 本来の特徴は文献[６,７,８,10,11]に書かれている通りであるが、３で述べたＯＳ/ｏ 第一版は最も基本的な機能を持たせてある。以下に特徴を示す。

（1）木構造化されたファイルシステム

ファイルシステムは木構造化されているのでファイルの管理が容易である。また、任意のディレクトリの下にディスクがマウント可能であるので、ディスクの管理をユーザは自由に行なえる。

（2）Ｉ／Ｏファイルの導入

ディスク以外の入出力装置をファイルという形式によって仮想化した。これにより、ユーザは入出力を統一的に扱える。

（3）実行中のユーザ・タスクからの別タスクの生成、起動、実行

たとえば、ユーザ・プログラムからコンパイラをサブプログラムのように使用できる。これにより、ユーザはすでに作成したプログラムを有効に活用すること

ができ、サブシステムの作成が容易となる。
（4）入出力の切り替えが可能なコマンドインタプリタ
入出力を切り替えることにより、スクリーン上に出力されるデータを容易にファイルに保存することができる。また、コマンドのバッチ処理も可能となっている。

6. 今後の計画

ＯＳ/ｏ 第一版はその核が稼動し始めたばかりである。したがって、ＯＳ/ｏ を我々の実際の研究対象に投入して、アプリケーション、アーキテクチャの両面からＯＳ/ｏ を評価するまでには、まだ多くの課題が残されている。それらにはＯＳ/ｏ のバージョン・アップ、各種ユーティリティ／ツールの開発・整備、そして、ＯＳとして今までにない野心的機能の実現などが含まれている。本節では、それらのうち、特に、最後の課題に焦点を絞って、今後、早急に行なうこと３点、および、その次のステップで実現することを以下に列挙する。
（1）日本語文字コード化
ＯＳ/ｏ 第一版は開発環境が、既存の、ASCII コードのCP/M-68K（コンパイラ開発もASCII コードのVAX/UNIX）であったため、その基本コードをASCII コードせざるを得なかった。しかし、日本語によるドキュメンテーションがソフトウェアの生産性、保守性、信頼性に対する寄与を考えると、できるだけ早い時期に基本コードを日本語文字コードに変換することが望ましい。そのために、ＯＳ/ｏ 及びそのＣコンパイラは日本語化への変更が容易に行なえるように設計してある。
（2）仮想フロッピーディスク、ＬＳＩファイルの実現
仮想フロッピーディスクは、フロッピーディスクのstaging areaをハードディスクにとって、また、ＬＳＩファイルは、ディスクファイルのそれをＬＳＩメモリにとって、それぞれのアクセス速度を向上させるものである。仮想記憶方式では、主メモリのアクセス速度を犠牲にして、記憶媒体の性能階層が見えない一様の広域なアドレス空間を提供した。一方、実記憶方式で、記憶媒体の性能階層を明示しているＯＳ/ｏ では、高速媒体でのstaging をユーザが指示することによって、物理的媒体ではなく、その媒体の性能を仮想化できるように設計した。
（3）完全なマルチ・タスクの実現とマルチＣＰＵ化
マルチＣＰＵの大型計算機のように複数のユーザ・ジョブを同時処理してリソースの有効利用を図るというのではなく、一つの問題解決（タスクフォース）のためにマルチＣＰＵを利用して、パターン認識や人工知能の問題に応用してみたいと考えている。
（4）プロテクションの強化とプログラム・データ共有の実現
（5）ドキュメンテーション・ツールの開発
（6）浜田方式とIEEE規格の両浮動小数点演算をサポートすること[3,4]
（7）システムプログラムのＲＯＭ化
（8）性能評価

7. おわりに

実記憶方式のオペレーティング・システム ＯＳ/ｏ 第一版について述べた。 マイクロプロセッサの出現により、dedicated システムのためのコントロール・ソフトウェア開発のニーズも高い。また、オンライン・アプリケーションも増大している。この背景には、ＣＰＵ、メモリをはじめ、ハードウェア・リソースの低価格化がある。我々は、いたずらに汎用ＯＳの歩んだ道にとらわれず、低価格のハードウェア・リソースを活用する見地から、もう一度、実記憶系のシステム構成を見直す必要があるのではないかと考えている。
なお、本研究は当学科の共同研究プロジェクト／πの一環として開始された。ここに、プロジェクト／πの参加者各位、特に中森眞理雄助教授、阿刀田央一助教授、鵜沢繁行助手に深謝する。また、本研究の基礎を築いた大学院生、武部桂史（現日立）、林努（現日立）、藤森英明（現日電）に謝意を表す。

参考文献

[1] 武部桂史，鵜澤繁行，中川正樹，阿刀田央一，高橋延匡，“MC68000アドレス空間の問題点と構成方針”，情報処理学会マイクロコンピュータ研究会，資料19-1，1981.12
[2] 林努，高橋延匡，“MC68000用ＯＳの基本設計(1) ── ファイルシステムの設計 ──”，同上25-3，1982.12
[3] 藤森英明，篠田佳博，清水敬子，中川正樹，高橋延匡，“MC68000 上で実現した浮動小数点演算方式 ── IEEE方式と浜田方式の比較 ──”，情報処理学会第24回全国大会予稿，1982.3
[4] 篠田佳博，藤森英明，清水敬子，中川正樹，高橋延匡，“MC68000 上での浮動小数点演算方式の実現 ── IEEE方式と浜田方式の実現 ──”，同上
[5] 中川正樹，篠田佳博，藤森英明，高橋延匡，“MC68000 ユニ＆マルチ・プロセッサ・システム用システム記述言語Ｃ処理系の開発”，情報処理学会計算機システムの制御と評価研究会，資料21-7，1983.12
[6] 高橋延匡，武山潤一郎，並木美太郎，中川正樹，“MC68000 用小型ＯＳ：ＯＳ/ｏ の開発”，同上，資料21-6
[7] 高橋延匡，“ＯＳ／omicron の設計思想”，情報処理学会第29回全国大会予稿，1984.9
[8] 並木美太郎，中川正樹，高橋延匡，“ＯＳ/ｏ のファイルシステム”，同上
[9] 並木美太郎，中川正樹，高橋延匡，“フロッピーディスクのLSIファイルによるＯＳの一性能向上方式 ── CP/M-68Kにおける適用結果 ──”，同上
[10] 武山潤一郎，並木美太郎，中川正樹，高橋延匡，“ＯＳ/ｏ のタスクとタスク管理”，同上
[11] 中川正樹，篠田佳博，高橋延匡，“ＯＳ/ｏ のプログラム実行環境とプログラム・リンケージ”，同上