C言語による大型計算機システムの有効利用

The effective utilization of the main frame computer system by means of language C

工学部　機械工学科
小　林　信　雄
Nobuo Kobayashi

1．まえがき

コンピュータ用の言語は、FORTRANを始め、各種のものが、それぞれの特徴を生かして各分野で使用されている。これらの言語は、いづれも汎用性に重点をおいて開発されたものであり、使用機種に拘わらずソース・レベルでの互換性を保つように留意されている。一方、コンピュータのシステム資源を管理するOSは、プログラム言語とは独立したジョブ制御言語により制御される。

この両者の分離が極めて厳格に行われてた為に、CPUやOSの高度化にも拘わらず、各言語で記述されたプログラムの継続性が保たれて来た。

ところが、この事は高度化したOSが提供し得る各種機能を制限する壁にもなっている。例えば、現在のOSではデータ管理において、ファイルを区分データセットとして取り扱う事が可能であるが、高級言語の中でも、区分データセットを単一のデータセットとして取り扱え、動的にそのメンバーを制御できるものはない。又、プログラム中から動的にファイルをオープン、クローズする事も不可能である。これらの機能は、OSが分担するものとして言語仕様には含まれていない。従って、これらの機能を使用するには、OSに付随するユティリティを用いるか、もしくはアセンブラでプログラムする以外には方法がない。しかしながらユティリティは機能が制限されており、又アセンブラは決して生産性の良いものとは言えない。

そこで、従来とは逆にOSと密接した言語があれば、汎用性には欠けるがシステムの機能を有効に利用することができ、効率的なプログラム開発が可能となる。

C言語については、周知のように、元来がミニコンピュータのOSであるUNIXを記述する為に開発されたものである。UNIXの評価が高まるにつれ、それを記述し得えたC言語自体についても、その機能の高さと記述性の良さが注目され、ミニ、マイクロコンピュータの標準的な言語に成りつつある。従って、このC言語を大型計算機に導入する事ができれば、OS依存型の言語として使用する事ができる。C言語及びUNIXの開発者であるベル研究所は、IBMのOS 370上で稼働するC言語マンバイラ、C／370を開発した。

本研究では、このC／370を本学の大型計算機に導入し、C言語による効率的なコンピュータ資源の利用について述べる。

2．C／370の構成

C／370の構成は次の通りである。

(1)　プリプロセッサ

(2)　構文解析部

(3) コード生成部

(4) ヘッダーファイル

(5) 入出力ライブラリ

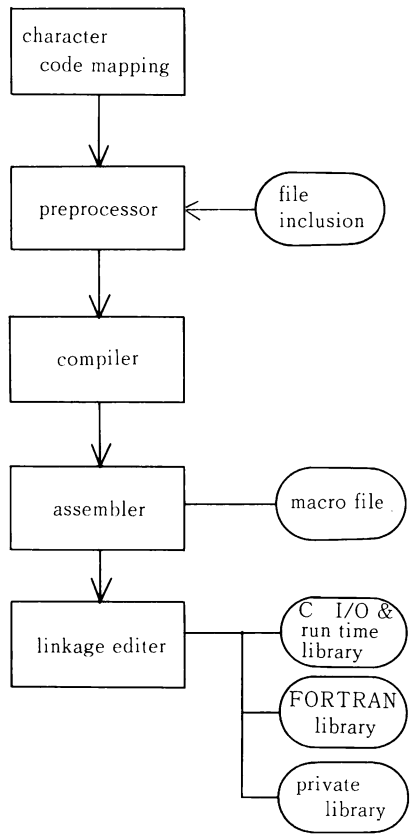
C言語自体はその殆どがC言語自身で記述されているので、元来移植性の高いものであるが、必ずマシン及びシステムに依存する部分が存在する。マシン依存の部分はマクロにより等価なものに置き換える事により対処している。又、システム依存の部分は全てSVC (Supervisor Call) によっている。C／370は、本来IBMのマシンでOS 370上で働くように設計されているが、本学のHITAC 240DのVOS Ⅲが同等なものかどうかが問題となる。公式にはその同等性が保証されているのであるが、次章以降で見られるように、一部相異する点も見受けられる。

又、導入に際しては文字コードの違いも問題となる。ミニコンピュータレベル迄は、文字コードとしてASCII、大型計算機ではEBCDICが使用されて来ている。更にC言語は文字セットが大きく、EBCDICコードで未定義のコード設定に違いが出て来る。C／370の文字コードとHITACのコードとの相違を表1に示す。

この違いに対処する為今回は、プリプロセッサの前段にコード変換部を設け、コードを合わせる方法を持った。(Prog 1、図1参照)

```
000100 (*
000200  * character mnapping for 'c'
000300  *)
000400 program charmap(inf,outf,input,output);
000500 const
000600     cbslash = 224;     (* x'e0 : BACKSLASH *)
000700     clbraket = 173;    (* x'ad : LEFT BRACKET *)
000800     crbraket = 189;    (* x'bd : RIGHT BRACKET *)
000900     clbrace = 192;     (* x'c0 : LEFT BRACE    *)
001000     crbrace = 208;     (* x'd0 : RIGHT BRACE   *)
001010     ctilde  = 95;      (* x'5f : TILDE         *)
001100     ccaret  = 154;     (* x'9a : CARET         *)
001110     cexclam = 90;      (* x'5a : EXCLAMATION   *)
001200 var
001300     inf : text;
001400     outf : text;
001500     c : char;
001600 begin
001700     reset(inf);    rewrite(outf);
001800     while not eof(inf) do begin
001900         while not eoln(inf) do begin
002000             read(inf,c);
002100             if c='¥' then c := chr(cbslash);
002200             if c='[' then c := chr(clbraket);
002300             if c=']' then c := chr(crbraket);
002400             if c='^' then c := chr(ccaret);
002410             if c='`' then c := chr(ctilde);
002500             if c='{' then c := chr(clbrace);
002600             if c='}' then c := chr(crbrace);
002610             if c='!' then c := chr(cexclam);
002700             write(outf,c);
002800             end;
002900         readln(inf);
003000         writeln(outf);
003100         end;
003200 end.
```

Prog 1　character mapping



ASCII	C/370 code (hex)	HITAC code
back slash	E0	¥
left bracket	AD	[
right bracket	BD	]
left brace	C0	{
right brace	D0	}
tille	5F	~
right quate	90	
caret	9A	^
exclamation	90	!

Table 1　character mapping set

```
000000 PROC 1 TYPE
000000 FREE DDN(SYSPROC)
000000 ALLOC DDNAME(SYSPROC) DA(CMD.CLIST) SHR
000008 IF &TYPE=L THEN GOTO LOCAL
000008 IF &TYPE=N THEN GOTO NORMAL
000009 IF &TYPE=C THEN GOTO CC
000008 WRITE ### ILLEGAL PARM.(&TYPE)
000000 WRITE ### TYPE=L (LOCAL)
000000 WRITE ###     =N (NORMAL)
000000 WRITE ###     =C (LANGUAGE C)
000000 WRITE
000000 EXIT
000008 NORMAL: WRITE
000008    WRITE   === NORMAL TYPE ASSIGNMENT FOR TERMINAL ===
000008    WRITENR ### TERMINAL PROFILE IS
000000    TERM LIST
000000    EXIT
000008 LOCAL: WRITE
000008    WRITE   === LOCAL TYPE ASSIGNMENT FOR TERMINAL ===
000000    WRITE
000000    CODE C
000011    TERM I(%) LINES(23) ASIS
000008    WRITENR ### TERMINAL PROFILE IS
000000    TERM LIST
000000    EXIT
100009 CC:    WRITE
200010    WRITE   === C LANGUAGE TYPE ASSIGNMENT FOR TERMINAL ===
300009    WRITE
400009    CODE C
500012    TERM ASIS
600009    WRITENR ### TERMINAL PROFILE IS
700009    TERM LIST
800009    EXIT
```

Prog 2　command procedure for C

3．走行環境

C／370はバッチジョブ及びTSSジョブのいずれとしても動作するが、C言語は文字コードとしてフルセットに近いものを使用するので、カード形式でプログラムを保持する事の利点は少ない。従って現在は、TSSでのみ使用する事ができる。

VOSⅢでは、ソースプログラムで小文字モードを受け付けるのは、ファイル属性がPASCALの時のみである。従って、ソースプログラムを編集する際にはファイル属性をPASCALとしておく事が望ましい。又、C言語には行概念が無いので、ファイル属性としてNONUMBERを付与しておく方が良い。

更にオンライン端末の伝送コードは小文字モードを選択できるものでなければならない。従って、使用端末機種は560－20型に限定される。

初期設定用のコマンドファイルをProg 2に示す。

4．C／370の関数

C／370はオリジナルなUNIXのシステムコールに関係したものは当然削除されているが、その代わりにVOSⅢ特有のシステム関数が用意されている。そこで、これらを例に採りながら、C言語によるOS機能の解析を行う。

4－1　cuserid

使用法　char ＊cuserid (p)

char　＊p；

TSSセッションをオープンしている利用者番号を文字列として返す。これによりファイル名を

完全修飾形で得られるので、任意の形式のファイルにアクセスする事ができる。また、他のユーザー番号を連結する事により、他ユーザーのファイルをダイナミックにアクセスする事が可能となる。但し、VOSⅢではデータ保護機能（SAFE）により、不正なデータアクセスを制限しているので、先ずPERMITコマンドにより、これを解除しておかねばならない。
この例をProg 3に示す。

```
#00001 /*************** cuserid *****************/
#00002
#00003 #include "stdio.h"
#00004
#00005 main()
#00006 {
#00007     char user[12];
#00008     char fname[30];
#00009
#00010     cuserid(user);
#00011     printf("Your USER-ID is %s¥n",user);
#00012
#00013     strcpy(fname,"'");
#00014     strcat(fname,user);
#00015     strcat(fname,".cmd.clist'");
#00016
#00017     if(fopen(fname,"r") == NULL)
#00018         cerror("File can't open");
#00019     else
#00020         printf("File exists:%s¥n",fname);
#00021 }
READY
call a(c)
Your USER-ID is T097200
File exists:'T097200.cmd.clist'
READY
```

Prog 3　sample program for function 'cuserid'

4－2　fopen

使用法　FILE　*fopen (frame, mode)
　　char　*frame, *mode ;

通常のファイル・オープン関数と同じ形式であるが、VOSⅢのファイル管理に対応できる様に機能拡張がなされている。詳細はファイルの動的アロケーションの項で述べる。

4－3　inquire

使用法　inquire (file, query)
　　FILE　*file ;
　　int　query ;

走行環境、状態変数あるいは、既にオープン済のファイルに内する情報を得る事ができる。問い合わせ項目を次の通りである。

1．走行環境　TSSバッチ
　　OSの種類

2．状態変数
・現在のエラーレベル
・最終のエラーコード
・プロンプト記号

3．ファイル属性
・READ／WRITE
・Binary／Ascii
・区分／順データセット
・最大レコード長
・論理レコード長
・論理装置名

・ファイルの DD 名（Data Definition）
・ファイル名
・ファイル・オプション
・レコード形式
・メンバー名（区分データセットの場合）
・DCB 情報（Data Control Blaek）

これ等はいずれもジョブ制御言語の DD 文により指定されるもので、通常はプログラム内からは参照する事は出来ず、あらかじめ設定しておかねばならないものである。しかしながらファイルの動的配置に際しては不可欠のものである。

4－4　intss

使用法　int　intss（　）

現在のプログラムがバッチで行われているのか TSS で実行されているのかを区別するものである。この機能は4－3の inquire 関数にも含まれているものである。これを用いる事により、データの入力先及び結果の出力先を走行環境により切り換えることが可能となる。

4－5　members

使用法　char　＊members（list flag）
　　char　＊list；
　　int　flag；

ファイルが区分データセットであり、fopen 関数により開かれたときに、メンバー名リストを得る。flag の機能は

flag＝0　　メンバー名は8バイト長
　＝1　　領域情報を含む
　＝2　　メンバーの別名を含む

である。これにより区分データセットを単一データセットと同じように取り扱う事が可能となる。（第5項参照）

4－6　nextmem

使用法　char　＊nextmem（name, fp）
　　char　＊name；
　　FILE　＊fp；

ファイルが区分データセットであり、popen 関数（もしくは preopen 関数）で開かれたとき、データセット中のメンバー名を返すものである。members 関数との違いは1回の呼出につき1個のメンバーを返す点にある。この機能により、区分データセットをメンバーの集合として一括処理が可能となる。

4－7　popen, preopen

使用法　FILE　＊popen（name, opt）
　　char　＊name, ＊opt；

区分データセットのメンバーを独立にアクセスする手段を与えるものである。第5項参照

4－8　system

使用法　system（cmd, dcb, pool）

```
char   *cmd, dcb ;
int pool ;
```

TSSのコマンドcmdをプログラム中から実行する事ができる。DCBを経由すればユーザ定義のロードモジュールを実行する事ができる。この機能は大変強力なものであり、この関数だけでもTSSコマンドの組み合わせでファイルの動的開閉やシステムユティリティを自由にプログラム中から使用することができる。

5．ファイルの動的配置

大型計算機は多数の人が同時に使用するので、システム内のファイルはユーザカタログ簿により管理されている。登録されたファイルが多数になると所用のファイルを検索するのにも時間がかかってしまう。更にシステムの記憶装置の利用効率も悪化する。従って同種の属性を持つファイルをまとめてOS側から見て単一のファイルとして管理が出来、ユーザ側からはファイル中のメンバーを独立にアクセスできるような区分データセットが導入されたのは必然であった。

又、ファイルの管理手法は、計算機システムの利用効率に大きく影響する為、これは全てOSが統一的に管理しており、各種言語とは独立したものである。従って、プログラムから指定できるのはDD名（Data Definition）であり、実際のファイルとの結び付きは、プログラム言語とは独立のジョブ制御言語によりOSに通知しなければならない。この事は、プログラムが使用するシステム資源は、走行前に全てが定まっていなければならない事を意味する。そして、確保された資源は、実行中の利用如何に拘ずプログラムが終了する迄システムを占有する。このようなシステム資源の静的配置は、利用効率が悪いばかりではなく、計算機の機能を必要以上に制限してしまうものである。例えば、計算結果をファイルに格納する時にも、あらかじめ出力量が予想できれば良いが、計算過程によっては出力量が大きくなり過ぎて、OSにより強制的に異常終了させられ、それ迄の計算が全く無駄になってしまう事がある。このような場合にもプログラム中から動的にファイルを配置できれば適切な処理を行う事が可能となる。

しかしながら、これを行う為にはOSのデータ管理機能をプログラム中から直接呼び出す事が必要となるが、一般の高級言語では望むべくもない機能である。これ迄は、アセンブラにより直接OSに依頼するのが唯一の方法であった。しかしながらアセンブラは生産性の極めて低いものであり手軽に利用できるものではない。C／370のファイル配置用の関数であるfopen, popenは、アセンブラと同等の機能をコンパクトな形で利用出来るものである。

この関数は、2つの引数を持つ。第1引数はファイル名であり、次のものが可能である。

(1) DD名
(2) 部分修飾データセット名
(3) 完全修飾データセット名
(4) 区分データセットのメンバ
(5) ターミナル　(*)
(6) 標準出力ファイル（SYSOUT data set）
    *pr　システムプリンタ
    *j　ジョブストリーム

(7) メモリ・ファイル (*inc)

外部記憶装置ではなく、主記憶内に作られる為、高速アクセスが可能である。また、他ジョブとの共有が可能であるので、ジョブ間の通信ができる。

```
#00001 /******************** nextmem ************************/
#00002 #include <stdio.h>
#00003
#00004 main()
#00005 {
#00006     char *fname="cmd.clist";
#00007     char mem[9];
#00008     FILE *fp;
#00009
#00010     if((fp=popen(fname,"r")) == NULL)
#00011         cerror("File can't open¥n");
#00012     while(nextmem(mem,fp)){
#00013         printf("name: %s¥n",mem);
#00014     }
#00015 }
```

```
READY
call a(c)
 name: BEGIN
 name: CC
 name: CE
 name: LLIST
 name: MK¥CC
 name: MK¥CPP
 name: POCOND
 name: PSCOND
 name: RANASM
 name: RANC
 name: RANFORT
 name: USRPROC
 READY
```

Prog 4 sample program for function 'nextmem'

```
#00001 /**************** members ******************/
#00002 #include <stdio.h>
#00003
#00004 main()
#00005 {
#00006     char *fname ="cmd.clist";
#00007     char *cp,mem[9];
#00008     FILE *fp;
#00009
#00010     if((fp=fopen(fname,"r")) == NULL)
#00011         cerror("File can't open");
#00012     cp = members(fname,0);
#00013     printf("%d members in dataset %s¥n",strlen(cp)/8,fname);
#00014     printf("%s¥n",cp);
#00015
#00016     strncpy(mem,cp,8);
#00017     strcat(fname,"(");
#00018     strcat(fname,mem);
#00019     strcat(fname,")");
#00020     if((fp=fopen(fname,"r")) == NULL)
#00021         cerror("File can't open");
#00022     else
#00023         printf("File successfully opened: %s¥n",fname);
#00024 }
****** ----+----+----+----+----+----END--+----+-(CHAR.ASIS.NONUM.NOTABS.NOHIGH)
```

```
READY
call a(c)
12 members in dataset cmd.clist
BEGIN   CC      CE      LLIST   MK¥CC   MK¥CPP  POCOND  PSCOND  RANASM  RANC
 RANFORT USRPROC
File successfully opened: cmd.clist(BEGIN   )
READY
```

Prog 5 Sample program for dinamic file allocation by means of 'fopen'

Prog 4 に区分データセットのアロケーション例を示す。popen 関数で開かれた区分データセット `cmd. clist' はファイル・ポインター fp により管理され、各メンバーへの位置付けは nextmem 関数によりなされ、各メンバーへの処理が独立に行う事ができる。

一方、区分データセットは通常の fopen 関数によっても開く事が出来るが、この場合には members 関数によって得られるデータセットのディレクトリ・リストから、改めてメンバー名を指定したファイル名でアクセスする事になる。

この例を Prog 5 に示す。

いずれの方法によっても区分データセットを単一ファイルとして開き、各メンバーに独立にアクセスする事ができる。この機能は、どのような高級言語においても実現できないものであり、ファイルユティリティの作成には不可欠の機能である。

次に、ファイルの動的アロケーションについて述べる。ファイル・ユティリティでは、プログラム内から必要なファイルを指定できる事が望ましい。しかしながら、現在のデータセットユティリティではファイル指定は全て DD 文により行わねばならず、プログラムの実行中にファイルを切り換える事は不可能である。このような状況では、真に効率的なユティリティを作成する事はできない。プログラム実行中に任意のファイルの生成、開閉ができる機能—動的ファイル・アロケーション—が必須である。動的ファイル・アロケーションの例を Prog 6 に示す。この例では、必要なファイルを端末からデータとして受け取り、動的なアロケーションが行われている事が解る。

また、既存のファイルをオープンした時には、そのファイル属性が不明であるので、ファイル属性もプログラム中で参照できなければならない。この例を Prog 7 に示す。

```
#00001 /************** dynamic file allocation ***********/
#00002 #include "stdio.h"
#00003
#00004 main()
#00005 {
#00006     char fname[30],mem[9],c;
#00007     FILE *fp;
#00008
#00009     puts("Input file name:");
#00010     scanf("%s",fname);
#00011     if((fp=fopen(fname,"r")) == NULL){
#00012         puts("File not exist");
#00013         exit();
#00014     }
#00015     if(inquire(fp,_PDS) != _PDS){
#00016         puts("File isn't PDS");
#00017         exit();
#00018     }
#00019     fclose(fp);
#00020     if((fp=popen(fname,"r")) == NULL){
#00021          puts("File can't open");
#00022          exit();
#00023          }
#00024     while(nextmem(mem,fp)){
#00025         printf("%s¥n",mem);
#00026         while((c=getc(fp)) != EOF)
#00027             putchar(c);
#00028         }
#00029 }
****** ----+----+----+----+----+----END--+----+-(CHAR.ASIS.NONUM.NOTABS.NOHIGH)
```

Prog 6　Sample program for dynamic file allocation by means of 'popen'

このようにC／370におけるファイル操作は極めて柔軟性に富むものであり、他の言語には見られない特性である。

```
#00001 /***************** inquire *********************/
#00002 #include <stdio.h>
#00003
#00004 main()
#00005 {
#00006     char *fname="cmd.clist";
#00007     int q;
#00008     FILE *fp;
#00009
#00010     if((fp=fopen(fname,"r")) == NULL)
#00011         cerror("File can't open¥n");
#00012
#00013     if(inquire(fp,_READ) == _READ)
#00014         printf("File opened for read¥n");
#00015     if(inquire(fp,_WRITE) == _WRITE)
#00016         printf("File opened for write¥n");
#00017     if(inquire(fp,_PDS) == _PDS)
#00018         printf("File is partitioned dataset¥n");
#00019 /* value */
#00020     printf("max record length %d¥n",inquire(fp,_MAXSIZ));
#00021     printf("logical record length %d¥n",inquire(fp,_LRECL));
#00022     printf("DD name %s¥n",inquire(fp,_NAME));
#00023     printf("file name %s¥n",inquire(fp,_FILENAME));
#00024     printf("options %s¥n",inquire(fp,_OPTIONS));
#00025     printf("member name %s¥n",inquire(fp,_MEMBER));
#00026     printf("DCB pointer %x¥n",inquire(fp,_DCB));
#00027 #if 0
#00028     if(inquire(fp,_DEVICE) == _DISC)
#00029         printf("device: disc¥n");
#00030     else
#00031         printf("not disc¥n");
#00032 #endif
#00033     if((q=inquire(fp,_RECFM)) && RECFM_FB)
#00034         printf("RECFM FB¥n");
#00035     else if(q && RECFM_VB)
#00036         printf("RECFM VB¥n");
#00037     else
#00038         printf("RECFM U¥n");
#00039 /* grobal */
#00040     if(inquire(NULL,_TS))
#00041         printf("in TSS¥n");
#00042     else
#00043         printf("not in tss¥n");
```

Prog 7　Sample program for function 'inquive'

```
#00044     if(inquire(NULL,_MVS))
#00045         printf("OS MVS¥n");
#00046     else
#00047         printf("not in MVS¥n");
#00048 /* grobal variable */
#00049     printf("current error level: %d¥n",inquire(NULL,_ERRLVL));
#00050     printf("last error number: %d¥n",inquire(NULL,_ERRNO));
#00051     printf("prompt  %s¥n",inquire(NULL,_PROMPT));
#00052 }
****** ----+-----+---+----+----+----END--+----+-(CHAR.ASIS.NONUM.NOTABS.NOHIGH)
```

```
 READY
call a(c)
 File opened for read
 File is partitioned dataset
 max record length 255
 logical record length 255
 DD name SYSPROC                  ::
 file name cmd.clist
 options r
 member name
 DCB pointer bb5c4
 RECFM FB
 in TSS
 OS MVS
 current error level: 1
 last error number: 0
 prompt  ::
 READY
```

6．FORTRAN との結合

C／370は数学関数を持っていない。しかしながら FORTRAN とリンクする為に宣言子 fortran を導入し、fortran ライブラリの資産を共有する事ができる。Prog 8 にプログラムとその結果を示す。注意する点は、FORTRAN では引数は参照渡しであるが、Cでは値渡しである。その為引数は、アドレス演算子によってポインタに変換したものを FORTRAN に渡す必要がある。又、Cの環境設定の為に FORTRAN プログラムはサブプログラムに限られ、Cが必ず主関数でなければならない。

```
#00001 /************ call fortran function ******************/
#00002
#00003 #include  <stdio.h>
#00004 #define PAI 3.141592
#00005
#00006 main()
#00007 {
#00008     fortran  double  sin(), cos();
#00009     float   x,y,z;
#00010
#00011     for(x=0; x<PAI/2; x+=0.1){
#00012         y=sin(&x);
#00013         z=cos(&x);
#00014         printf("x=%f y=%f z=%f ¥n",x,y,z);
#00015     }
#00016 }
****** ----+----+----+----+----+----END--+----+-(CHAR.ASIS.NONUM.NOTABS.NOHIGH)


 READY
call a(c)
 x=0.000000 y=0.000000 z=1.000000
 x=0.100000 y=0.099833 z=0.995004
 x=0.200000 y=0.198669 z=0.980067
 x=0.300000 y=0.295520 z=0.955337
 x=0.400000 y=0.389418 z=0.921061
 x=0.500000 y=0.479425 z=0.877583
 x=0.600000 y=0.564642 z=0.825336
 x=0.700000 y=0.644217 z=0.764842
 x=0.800000 y=0.717356 z=0.696707
 x=0.900000 y=0.783327 z=0.621610
 x=1.000000 y=0.841471 z=0.540303
 x=1.099999 y=0.891207 z=0.453597
 x=1.199999 y=0.932039 z=0.362359
 x=1.299998 y=0.963558 z=0.267500
 x=1.399998 y=0.985449 z=0.169969
 x=1.499997 y=0.997495 z=0.070740
 READY
```

Prog 8　Sample program to coll FORTRAN function

しかし、この事は FORTRAN とCとのリンクを制限するものではなく、むしろ FORTRAN とCのプログラムを自然な形で融合させるものである。

FORTRAN のプロッターライブラリや（GPSL）数値計算ライブラリ（MSL）を自由に使用する事ができ、これまでの FORTRAN プログラムを置き換える必要もなく、C言語に移行することができる。

7．System 関数による TSS ユティリティとの結合

計算機の使用形態が TSS 中心になるにつれ、計算機システムの新しい機能は先ず TSS で利用できるように考慮されている。

ファイル・コピー、比較などのユティリティやカタログ情報、データセット属性、ネットワーク機能等計算機システムを有効に利用できるものが多い。従って、これらの機能をプログラム内から実行できれば、同等の機能を各自で実現するよりも信頼性が高く、高速性が期待できる。

```
#00001 /*************** system() ***************/
#00002
#00003 # include <stdio.h>
#00004
#00005 main()
#00006 {
#00007      system("copy c.pascal cccc.pascal");
#00008      system("listd cccc.pascal");
#00009      system("st");
#00010 }
****** ----+----+----+----+----+----END--+-----+-(CHAR.ASIS.NONUM.NOTABS.NOHIGH)


READY
call a(c)
 JED03180I NEW DATA SET(T097200.CCCC.PASCAL) CREATED
 JED03260I 10 MEMBERS COPIED
 T097200.CCCC.PASCAL
 --RECFM-LRECL-BLKSIZE-DSORG
   FB     80     3120     PO
 --VOLUMES--
   VOS311
 JET35427I JOB T097200K(JOB06951) IN OUTPUT QUEUE
 JET35446I JOB T097200(TSU09603) IN EXECUTION
 READY
```

Prog 9　Sample program for function 'system'

Prog 9 にその例を示す。

全ての TSS コマンドが実行できるかどうかは確認していないが、ファイル・ユティリティとカタログ情報関係は正常に実行できる。

機能的に難点のあるコマンド・プロセッジャーと比較しても、はるかに柔軟なプロセス制御が可能である。

8．計算速度の比較

計算機言語は、機能が大きくなればそれに反比例して計算速度が低下するのが通例であるが、C言語はシステム開発用言語としての性格上、実行時の制限は可能な限り小さく押さえている。

従って他の高級言語に比べれば、計算速度の向上が期待できる。

この点を乱数生成プログラムを例に採り比較を行う。モンテカルロ法などの計算機シミュレーションにおいては、多数の乱数を必要とする為、高速のアルゴリズムが要求される。

乗数合同法による疑似乱数の生成を、FORTRAN、C／370、アセンブラで行ったときプログラムを Prog10に結果を表 2 に示す。

この結果からも解るように、Cの計算速度は FORTRAN とアセンブラの中間に位置する。従って、C言語は数値計算用言語としても十分に使用する事ができる。言語構造が FORTRAN と比較して、はるかに整備されており、FORTRAN の代替として適当なものである。

```
000001  CALL CLOCK
000010  DO 10 I=1,10000
000020    RAN=FLTRNF(X)
000030 "  WRITE(6,*) RAN
000040 10 CONTINUE
000041 "
000042  CALL CLOCK(TIME,4)
000043  WRITE(6,100)TIME
000044  100 FORMAT(1H ,'ELAPSED TIME FOR 10000 GENERATIONS OF RANDOM NUMBER:' -
000045             G12.4,'(SEC)')
000050  STOP
000060  END

#00154 C     .....................................................................
#00155 C
#00156       SUBROUTINE RANDU(IX,IY,YFL)
#00157       COMMON /LRANL/IRAN
#00158 C     IY=IX*65539
#00159       IY=IX*48828125
#00160   8   IF(IY)5,7,6
#00161     7 IRAN=IRAN+1
#00162       IY=IRAN
#00163       GO TO 8
#00164     5 IY=(IY+2147483647)+1
#00165     6 YFL=IY
#00166       YFL=YFL*.4656613E-9
#00167       RETURN
#00168       END
#00001 /*---------------random---86.1.8---OK----*/
#00002 #include<stdio.h>
#00003 main()
#00005        fortran clock();
#00006        int ix,iy,iran,k,a,b;
#00007        float ran,fltranf(),p;
#00008        char c;
#00009
#00010              ix=1;
#00011       clock();
#00012       for(k=0; k<10000; k++){
#00013           ran = fltranf(ix);
#00014           printf("ran= %f¥n",ran);
#00015           }
#00016       k = 4;                    /* k=4:get time in sec unit in float */
#00017       clock(&p,&k);              /* get time in 'p' in sec unit */
#00018       printf(
#00019             "Elapsed time for 10000 generation of random number: %f¥n"
#00020             ,p);
#00021 }
#00022
#00023 /*
#00024  * fltranf
#00025  */
#00026  int iran=1;
#00027
#00028 float fltranf(ix)
#00029       int *ix;
#00030 {
#00031        int iy;
#00032        float yfl;
#00033
#00034        if((iy=(*ix)*48828125)==0){
#00035               ++iran;
#00036               iy=iran;
#00037               }
#00038        if(iy<0)
#00039              iy=(iy+2147483647)+1;
#00040       yfl= (float) iy;
#00041       yfl=yfl*0.4656613E-9;
#00042       *ix=iy;
#00043       return (yfl);
```

Prog10　program to test calcuration speed in C and FORTRAN

言　語	所要時間 (msec)
FORTRAN	55.00
C/370	33.70
ASM	15.21

Table 2　Time required in 1000 random number generation

まとめ

C言語は、それ自体が優れた構造化言語である。しかしながら、C言語の特徴は他の言語とは逆に計算機システムに密着したシステム依存型の言語である点にある。

この事により、従来の言語では不可能であったシステムプログラムの開発を行う事ができるようになった。

本研究において、C／370がVOSⅢのシステムプログラム開発用の言語として十分な機能を有するものである事が明らかとなった。その特徴は以下の通りである。

1）プログラム内から、順データセット及び区分データセットを動的に生成、開閉する事ができる。この機能によりユーザ仕様のデータセットユティリティを作成する事ができる。

2）プログラム内から、TSSコマンド及び任意のロード・モジュールを実行する事ができる。この機能により、動的にジョブ制御を行う事が可能となる。

3）FORTRANのサブプログラムを直接呼び出すことができる。この機能により、これまでFORTRANによって蓄積されて来た成果をそのまま引き継ぐ事ができ、計算速度もFORTRANよりは高速である。

今後は、以上のような特徴を生かして、大型計算機の機能を十分に生かした利用法の開発が課題となるであろう。

なお、本研究は昭和60年度産業研究所の特別研究費の助成を受けて行われたものであります。

参考文献

1）J. F. Gimpel, A Programmer's Guide to C/370 for OS/370, Bell Tel. Lab, 1979
2）B. W. Kernighan, D. M. R：tchie, The C Programming Language, Prentice Hall, 1978
3）日立製作所、VOS 3　ジョブ制御言語
4）　同上　、VOS 3　拡張アセンブラ言語
5）　同上　、VOS 3　最適化 FORTRAN 77言語