# PIMOSとユーザプログラムの間の通信方式

ICOT

第4研究室

佐藤裕幸

## PIMOSの概要

(a) KL1で記述された，並列推論マシン用OS
(b) 本格的なOSとして必要な機能を網羅
(c) フロントエンド・プロセサを接続
(d) 集中型単一OS
(e) 使い勝手の良いシステム
(f) シングル・ユーザ，マルチ・タスク
(g) クロス・システムの充実

# 世の中の始まり（単純な方式）

```
boot :-true  |
        pimos(Req), user(Req)
```

- 共有変数Reqで通信（Reqは唯一の通信路）
  Req=[create_window(Window)| ReqT]
  →Windowも通信用変数となる

- ユーザに通信用変数を渡す
  =ユーザにアクセス権（capability）を与える

# 荘園（資源管理，異常処理）機能

- ユーザとPIMOSはAND関係
  → ユーザの誤りでシステム・ダウン
- ユーザ・プログラムの制御が不可能

↓

execute(GOAL, CONTROL, REPORT)

- GOAL　　：実行するプログラム
- CONTROL：制御用ストリーム
- REPORT　：報告用ストリーム

# 世の中の始まり（荘園を用いた方式）

```
boot :- true |
        pimos(Req, Cnt, Rep),
        execute(user(Req), Cnt, Rep).
```

- 変数Cntでユーザ・プログラムの実行を制御
- 変数Repでユーザ・プログラムの資源を管理

タスク：荘園（言語定義資源の管理）＋
OS定義資源（入出力資源等）の管理

# OS／ユーザ間の通信の例

ユーザ側：

① Req=[getb(N, String) | ReqT]
② Stringの具体化を待つ
③ Stringの値を使う

OS側：

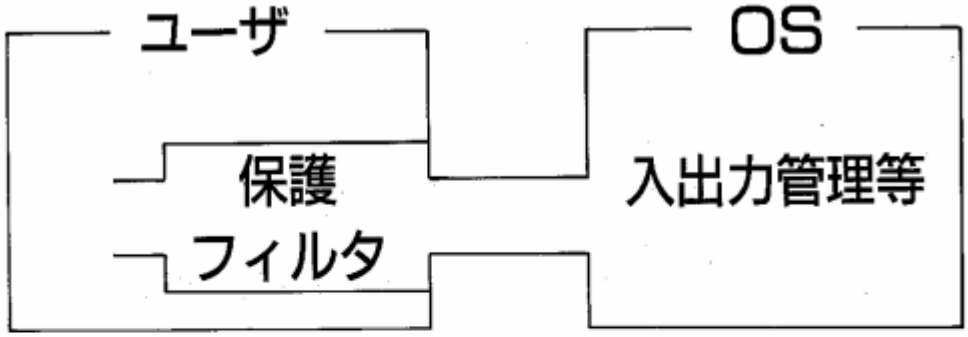
⑴ Reqの具体化を待つ
⑵ 読み込む文字数Nの具体化を待つ
⑶ 読み込み操作をする
⑷ Stringを読み込んだものに具体化する

# 問　題　点

Ⅰ）ユーザがStringを変な値に具体化する
　　Stringが未定義変数であるかの判定は不可能
　　→　OS側の具体化が失敗する
　　→　システム・ダウン

Ⅱ）ユーザがNを具体化する前に実行を放棄
　　→　OSがNの具体化を永遠に待ち続ける
　　→　ゴミ・プロセスが残る

# 保護フィルタ




(1) ユーザが与えるデータの具体化を待つ
(2) OSには未定義変数を送り，
　　結果をユーザ変数に具体化する

失敗するならユーザ側

# 保護フィルタ・プログラムの例

```
filter([getb(N,String)|ReqT],OS_Req) :-
  wait(N) |
    OS_Req=[getb(N,OS_String) |OS_ReqT],
    wait_and_unify(OS_String,String),
    filter(ReqT,OS_ReqT).

wait_and_unify(OS_Var,USER_Var) :-
  wait(OS_Var) |
    OS_Var=USER_Var.
```

# 世の中の始まり（保護フィルタ付き）

```
boot :- true | pimos(OS_Req,Cnt,Rep),
        execute(
          ( user(USER_Req),
            filter(USER_Req,OS_Req) ),
            Cnt,Rep),
```

- 保護フィルタによりユーザの誤りから
  システムを保護
- 保護フィルタの挿入
  ＝ユーザのアクセス権の制限

# 保護フィルタの生成

- 保護フィルタはOSで提供
- 通信プロトコルが分かれば機械的に生成可能
  → プロトコル定義言語から自動生成

例:
```
kbd_manager = stream (kbd_command).
kbd_command = {
        getb(length, -buffer) ;
        getl(-line) ; getc(-character) }.
length  = integer.   buffer    = string.
line    = string.    character = integer.
integer = atomic.    string    = atomic.
```

# 今 後 の 予 定

- 各部分機能の詳細化
  タスク間での資源の共有等

- FGCS' 88 に向けて,
  マルチPSI第2版上での開発

- 実用化システムに向けての改良
  並列プログラミングのツールとして