談　話　室

PV 操作について

徳田雄洋* 徳田英幸**

投稿のことば

"情報処理"(文献1)) および "Computing Surveys" (文献2)) に発表された，2つの解説記事中の，semaphore についての PV 操作の定義に**疑問点**があることを指摘し，また従来行われていた定義よりもインプリメントしやすい多重 PV 操作の定義を**提案**している．

1. 序　論

近年，semaphore についての議論が盛んで，さまざまな雑誌に論文や解説記事が掲載されますことは大変意義深いことであると思われます．しかしながら，最近の「情報処理」および "Computing Surveys" 誌に発表されました2つの tutorial につきまして，基本操作 (PV 操作) の定義に疑問があるように思われますので，広く会員の皆様に検討して頂きたく，私共の意見を述べさせて頂きます．

2. 単純 PV 操作

主にプログラマとしての立場からですが，PV 操作の定義の仕方には2種類あると考えられます．第1の方法は semaphores S を非負整数変数とするもので，例えば次のようなものです．Q_s は S の queue を示し P 操作，V 操作の順に表記しています．

$$\begin{cases}\textbf{if } S>0 \textbf{ then } S:=S-1 \textbf{ else block}; & (1)\\ \textbf{if not empty } (Q_s) \textbf{ then wake else} \\ \quad S:=S+1; & (2)\end{cases}$$

そしてこの場合，$S>0$ の時は Q_s が空で，$S=0$ の時は queuing process がある場合とない場合の2通りが存在します．

第2の方法は S を整変数とするものです．

$$\begin{cases}S:=S-1; \textbf{ if } S<0 \textbf{ then block}; & (3)\\ S:=S+1; \textbf{ if } S\leqslant 0 \textbf{ then wake}; & (4)\end{cases}$$

この場合，両操作の定義が変数 S のみで済んでいる点が特徴です．例えば Boolean の empty のみでは両操作の定義に不十分です．この第2の方法では，$S<0$ の時，$|S|$ が queuing process の個数に一致するので，次のように定義しても等価です．

$$\begin{cases}S:=S-1; \textbf{ if } S<0 \textbf{ then block}; & (3)\\ S:=S+1; \textbf{ if not empty } (Q_s) \textbf{ then wake}; & (4')\end{cases}$$

いずれにしても，第2の方法では $S>0$, $S=0$ の時はqueuing process はなく，$S<0$, の時，$|S|$ 個の queuing process が存在します．

以上の準備をもとに，2つの tutorial を調べてみます．まず，文献 1) では次のような PV 操作の定義が述べられています．

$$\begin{cases}\textbf{if } S\geqslant 1 \textbf{ then } S:=S-1 \textbf{ else block in queue}; \\ S:=S+1; \textbf{ if } S<1 \textbf{ then wakeup the first} \\ \quad \textbf{member in queue};\end{cases}$$

この定義を整理しますと，次と等価になります．

$$\begin{cases}\textbf{if } S>0 \textbf{ then } S:=S-1 \textbf{ else block}; & (1)\\ S:=S+1; \textbf{ if } S\leqslant 0 \textbf{ then wake}; & (4)\end{cases}$$

すなわち，非負整数を扱うPと整数を扱うVの "混合体" で，初期値 $S_0\geqslant 0$ から出発すると，block された process は永久に wakeup されないことになります．恐らく何らかのミスによるものであると思われます．

次に，文献 2) には次のような定義が述べられています．

- **if** $E\geqslant 1$ **then** $E\leftarrow E-1$
  and the process issuing the P continues its progress.
  else an entry of the form (name of process issuing P, address of P) is placed in the waiting list associated with E, and the process issuing the P enters the blocked state.
- $E\leftarrow E+1$
  if $|L_E|\geqslant 1$ **then** remove one of the entries from

* 東京工業大学理学部情報科学科
** 慶応義塾大学工学部管理工学科

the waiting list and change the status of the corresponding process to the ready state. The process issuing the S is placed in the ready state.

else the process issuing the S continues its progress.

この定義を整理しますと，一見(1)と(4′)の混合のように見えます．しかしながら，P操作でblockを行なう時のステートワード[8]の処理に注意すると，P操作のアドレスを格納しています．従って，blocked stateからreadyになり，再びrunningになったとするとP操作の先頭から再開すると解釈できます．このように解釈すると，この一対の定義は正しく動作することになりますが，ステートワードの処理に関して，多少非能率的と思われます．これは，後述する多重PV操作の定義と形式を合わせるためとも考えられますが，単純P操作を"再入"する形で定義するのはあまり好ましいこととは思えません．

以上から，上述した2組の定義には疑問があると思われます．

3. 多重PV操作

最初に多重P操作を主張したPatilは文献3)で次のように論文を結んでいます．

"Petri netを実現するための構造は，著者の初期の仕事(文献4))により究明されてきた．この構造はkがsome large but fixed numberのときは$P\{S_1,\cdots,S_k\}$命令を実現するのに用いることができる．この操作を施すsemaphoreの数が，この数より大きいときは，本論文で呈示された方法により，conditional statementの助けを借りて多重度の小さい命令のいくつかへと，コンパイラが分解することができる．"

残念ながら私共の手元に文献4)がないため，Patilがどのような構造を考えていたのかは定かではありません．しかしながらsmokers' problemを直接的に解くために考察された多重P操作は，実際にインプリメントしようと思うと，大変に大袈裟なものになってしまうようです．例えば，文献1)では次のように多重PV操作を定義しました．

if $(S_1\geqslant 1)\wedge\cdots\wedge(S_n\geqslant 1)$ **then** $(S_1:=S_1-1;$ $\cdots;$ $S_n:=S_n-1)$ **else** block in queue;

$S:=S+1$; **if** $S<1$ **then** wakeup the first member in queue;

この一対の定義に表われるqueusという言葉がそれぞれ何を示すかは，簡潔な解説のため不明ですが，私共が想像する範囲では，次のような解釈になると思われます．

"全部のsemaphore"の部分集合の1つ1つにqueueを対応させる．もし，あるsemaphore S_i にV操作が施されたとして，wakeupが必要な時は，S_i が原因でqueuingしていた全processの中からある1つのprocessが選ばれて，別のqueueへ転送されるかあるいは，完全に各種のqueueをすべて抜け出て，wakeupされる．

この解釈通りに多重PVを実現すると，"queueの列"の運営が大変になり，primitiveと呼ぶには複雑過ぎるように思われます．

文献2)では，文献1)と異なり，多重P操作と多重V操作を定義しています．blocked stateに入ったprocessは，その原因となったsemaphoreの1つのwaiting listに並べられ，多重V操作が施される度にV操作の対象となった全semaphoreのwaiting list中の全processを追い出し，多重P操作の最初の判断を再評価させようというものです．以下では少しだけ異なったやり方で，多重PV操作を定義してみます．記法は文献5)に従いますが，結局，"conditional critical region"として，多重PVを定義し，Boolean expressionのtestを何度か行わせるものです．余り明瞭とは言えぬ多重PV操作を認めるためには，この形のbusy form of waitingは仕方がないと思われます．

- P $(S_1,\cdots S_n)$ {ただし．**var** e: **event** v}

```
region v do
begin
  while not (S1>0 &···& Sn>0) do await(e);
  S1 := S1−1; ···; Sn := Sn−1
end
```

- V $(S_1,\cdots,S_m)$

```
region v do
begin
  S1 := S1+1; ···; Sm := Sm+1;
  {if not empty (Qe) then} cause (e)
end
```

4. 結 論

PV操作が，私共にわかりにくい理由を考えてみますと，第1にはDijkstra氏自身が極めて文学的にその定義を与えてしまったこと[6]，そして第2には，この種の対操作は，どのような順序で用いられるとどの

ような効果を持つのかが一目ではわかりにくいこと[7]によると思われます．もう少しわかりやすいstyleで，この種の基本操作が，将来は定義されることを希求します．

参考文献

1) 斎藤信男：OSの基礎理論，情報処理，Vol. 15 No. 11, pp. 887～899 (1974).
2) L. Presser : Multiprogramming Coordination, Comp. Surveys, Vol. 7, No. 1, pp. 21～44 (1975).
3) S. S. Patil : Limitations and Capabilities of Dijkstra's Semaphore Primitives among Processes, Comp. Str. Group Memo 57, MIT (1971).
4) S. S. Patil : Coordination of Asynchronous Events, MAC-TR-72 MIT (1970).
5) P. B. Hansen : Operating System Principles, Prentice Hall, Englewood Cliffs, N. J., (1973).
6) E. W. Dijkstra : Co-operating Sequential Processes, **Programming Languages**, Academic Press, New York (1968).
7) A. N. Habermann : Synchronization of Communicating Processes, Comm. ACM, Vol. 15, No. 3 (1972).
8) B. W. Lampson : "A Scheduling Philosophy for Multiprocessing Systems", Comm. ACM, Vol. 11, No. 5 (1968).

(昭和50年8月1日受付)

◇

付記

文献2)に対するPresser氏自身の訂正がComp. Surveys, Vol. 7, No. 4, p. 257 (1975)に，また同文献に対するM. S. Doyle氏の見解，A. H. Habetmann氏の見解が，それぞれComp. Reviews誌の1975年9月号，1976年4月号に掲載されています．