병행 프로그래밍을 위한 효율적인 동기화 구조에 관한 연구

(A Study on the Efficient Synchronization Mechanism for Concurrent Programming)

吳 炳 均*, 金 成 洛**, 李 相 範***

(Byung Kyun Oh, Sung Rak Kim, and Sang Burm Rhee)

要 約

최근 컴퓨터 시스템의 구성이 여러 개의 프로세서를 결합하는 방식으로 변화함에 따라, 이러한 시스템을 지원할 운영 체제에 대한 연구가 요구되고 있다.

본 논문에서는 여러 프로세스들의 병행 실행을 동기화 구조를 적용하여 병행 프로그래밍 언어 pascal-s의 프로그래밍 기법으로 구현하였다.

이때 프로세스들의 병행성은 "cobegin/coend"문으로 표현하였으며, 프로세스 사이의 동기화 구조는 semaphore를 적용하였다.

또한 프로세스의 실행은 pascal-s를 이용하여 병행 시스템을 구성하고, 인터프리터가 프로세스의 실행 알고리즘으로 병행 시스템의 프로세스들을 수행한다.

따라서 본 논문의 동기화 기법은 여러 프로세스들이 동시에 병행으로 실행되도록 하므로서 운영 체제의 도구로 사용될 수 있음을 보였다.

Abstract

Recently, as the configuration of computer systems are changed to the system equipped with several processors, operating system to support these systems has been needed.

The purpose of this paper is to implement concurrent execution of processes as synchronization mechanism by using concurrent programming language, pascal-s.

In this paper, concurrency of processes is represented by "cobegin/coend" statement, semaphore is adopted in the synchronization mechanism among processes, and then pascal-s is constructed concurrent system, the interpreter simulates concurrent system with the implementing algorithm for processes.

Conclusively, the synchronization mechanism in this paper shows the concurrent execution of processes as tools of system program.

*正會員, 木浦大學校 電算統計學科
(Dept. of Computer Sci. and Sta., Nat'l Univ.)

**正會員, 關東大學校 情報管理學科
(Dept. of Inf. Man., Kwandong Univ.)

***正會員, 檀國大學校 電子工學科
(Dept. of Elec. Eng., Dankook Univ.)

接受日字 : 1989年 9月 11日

(※ 이 연구는 1989년도 한국과학재단 일반 기초 연구비 지원에 의하여 이루어졌음. 과제번호 : 891-1103-005-1)

I. 서 론

최근 성능이 향상된 프로세서(processor)의 개발과 컴퓨터 설계 기술의 발달로 인하여 컴퓨터 시스템의 구성은 다중처리 시스템(multiprocessing system), 분산처리 시스템(distributed system), 독립된 입출력 장치를 갖는 시스템과 같이 여러 개의 프로세서를 결합하는 방식으로 전환되고 있다[1, 2, 3].

따라서 이러한 시스템을 지원할 운영 체제(operating system)에 대한 연구가 요구되고 있으며, 그 중

의 하나가 프로세스들의 병행 실행(concurrent execution)을 위한 동기화 구조(synchronization mechanism)에 관한 연구이다[2, 3, 4].

프로세스들의 병행 실행이란 여러 프로세스들이 동시에 작업(task)을 수행할 때, 이들이 서로 상호작용(interaction)하지 않을 경우는 병렬로 실행하고, 서로 협동하여 수행할 경우는 동기화 절차에 따라 실행하는 것을 말한다.

이때 프로세스들이 서로 협동하여 작업을 수행할 경우는 어떠한 형태로든지 프로세스 사이에 정보교환(communication)이 이루어져야 한다. 그러나 각 프로세스는 독립적으로 작업을 수행하기 때문에 정보교환을 해야할 시점을 미리 예측할 수 없고, 그 시점을 미리 정해 놓을 수도 없다. 따라서 서로 협동해야 할 프로세스들은 자기가 필요한 시점에서 정보교환을 요구할 수 있도록 하고, 비동기적으로 발생하는 프로세스 사이의 정보교환 요구를 조직적으로 통제하기 위한 절차로서 동기화 구조가 필요하게 된다[5, 6, 7].

일반적으로 동기화 구조는 프로세스 사이에 정보교환이 이루어지도록 지정된 임계 영역(critical region)의 유무에 따라 동기화 구조가 구분된다.

임계 영역을 갖는 시스템에서는 공통 변수(common variable)를 이용하여 임계 영역에 대한 액세스(access)를 통제하므로서 프로세스 사이의 정보교환을 동기화하며, 이러한 동기화 구조로는 semaphore, monitor, path-expression, serializer 등이 있다[5, 7].

또한 임계 영역이 없는 분산 시스템에서는 메세지 전달(message passing)에 의해 프로세스 사이의 정보교환을 동기화하며, 이러한 동기화 구조로는 send/receive, DP, CSP, RPC 등이 있다[6, 8].

본 논문은 여러 프로세스들이 CPU를 공유하면서 병행으로 실행되도록 동기화 구조를 적용하여 병행 프로그래밍 기법으로 구현하기 위한 연구이다. 이를 위하여 프로세스들의 병행성 표현과 동기화의 실행은 다음과 같은 방법으로 구현하였다.

첫째, 기존의 pascal 언어에 병행성 기능을 갖는 "COBEGIN/COEND"문과 동기화 기능을 갖는 "WAIT(s)/SIGNAL(s)"문을 부가하여 pascal-s라는 병행 프로그래밍 언어를 구성하였으며, 이 언어를 이용하여 프로세스들의 병행성을 표현하고, 프로세스들의 동기화를 위한 병행 프로그램을 작성하였다.

둘째, 동기화의 실행은 pascal-s에 의해 생성된 프로세스의 테이블(PTAB)을 스택에 할당하여 병행 시스템을 구성하고, 병행 프로세스들은 세마포어를 이용하여 프로세스들 사이의 상호작용을 동기화 하였다.

셋째, 프로세스의 수행은 병행 시스템에 대한 실행 알고리즘을 구성하고, 프로세서들의 실행을 시뮬레이션하였다.

Ⅱ. 병행 프로세스에 대한 고찰

1. 프로세스의 병행성

일반적으로 프로세스(process)는 운영 체제에 의해 프로그램을 주 기억공간(main memory)에 할당하므로서 생성되고, 프로그램의 실행에 따라 프로세스의 상태가 변한다.

또한, 인터럽트(interrupt)가 발생하면 프로세스의 상태는 일시적으로 고정되고, 프로그램의 실행이 종료되면 프로세스는 소멸된다.

따라서 프로세스란 운영체제 내에서 실행 중인 연산 과정이나 실행 중인 프로그램(executing program)을 말한다.

한편, 프로그램은 처리 과정에서 여러 개의 독립적인 프로세스를 형성하며, 이들 프로세스는 서로 상호작용하지 않을 경우 병렬로 실행한다. 그러나 어떤 자원을 대상으로 서로 협동할 경우는 동기화 절차에 따라 실행된다.

이와 같이 여러 프로세스가 동시에 실행되는 것을 프로세스들의 병행성(concurrency of processes)이라고 하며, 그림 1은 이러한 병행 프로세스들의 실행 상태를 나타낸 것이다.

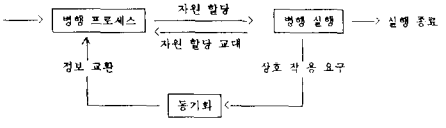


그림 1. 병행 프로세서의 실행 상태도
Fig. 1. Executing status diagram of concurrent processes.

2. 프로세스의 동기화

여러 프로세스가 동시에 실행할 때, 프로세스들이 서로 상호작용하지 않으면 병렬로 실행하고, 서로 협동할 경우는 조직적으로 통제된 절차에 따라 실행한다.

일반적으로 어떤 자원을 대상으로 서로 협동하고 있는 프로세스 사이의 관계는 경쟁 관계이거나 생산자와 소비자 관계이므로 프로세스들이 서로 협동하기 위한 상호작용을 프로세스의 동기화(synchronization of processes)라고 한다.

서로 협동하고 있는 프로세스 사이의 동기화 방법은 필요한 자원을 할당받아 작업을 수행할 수 있는 임계 영역(critical region)을 설정하여 공통 변수(common variable)에 의해 한 시점에서 하나의 프로세스만 임계 영역을 액세스할 수 있도록 상호 배제(mutual exclusion)하므로서 동기화를 수행한다.

그림 2는 진입 코드(entry code)와 진출 코드(exit code)를 공통 변수로 하여 임계 영역에 대한 상호 배제 알고리즘(algorithm)을 나타낸 것이다.

```
REPEAT
    단계 1 : IF〈entry section〉  THEN 〈단계 2〉
                                 ELSE 〈단계 4〉
    단계 2 : 선정된 프로세스에 임계 영역 할당
    단계 3 : IF〈exit section〉   THEN 〈단계 4〉
                                 ELSE 〈단계 2〉
    단계 4 : 병행 프로세스의 대기
UNTIL false
```

그림 2. 공통변수를 이용한 임계영역의 상호 배제 알고리듬

Fig. 2. Mutual exclusion algorithm by using common Variable.

Ⅲ. 병행 프로그래밍 언어

1. pascal-s

병행 프로세스들의 실행 방법과 절차를 프로그래밍하기 위해서는 병행 프로그래밍 언어가 필요하다[10, 11, 12].

본 논문에서는 기존의 pascal 언어에 병행성(concurrency) 기능을 갖는 "COBEGIN p1;p2;..;pn COEND"문과 동기화(synchronization) 기능을 갖는 "WAIT(s)/SIGNAL(s)"문을 첨가하여 pascal-s라는 병행 프로그래밍 언어를 구성하였다.

이 언어는 블럭 구조(block structure)와 기계 종속성(machine dependency) 표현이 용이하며, 병행 프로그램을 실행할 수 있는 인터프리터(interpreter)이다.

pascal-s는 가상 기계(hypothetical machine)를 설정하고 코드를 선언하는 부분과 병행 프로그램을 컴파일하여 생성된 요소를 선언된 코드에 할당하여 스택에 저장하는 부분, 그리고 스택에 저장된 코드를 해석하여 실행하는 부분의 세 영역으로 구분되며, 그 구조는 그림 3과 같다.

```
PROGRAM pascal-s;
  CONST   list of hypothetical machine;
  VAR     code : ARRAY[1..codemax] OF instruction;
PROCEDURE BLOCK;
  BEGIN
    〈compile the pascal-s program〉
    〈store the compiled instruction into the array code〉
  END;
PROCEDURE INTERPRET;
  VAR     stack : ARRAY[1..stackmax] OF integer;
  BEGIN
    〈simulate the instruction in code〉
    {the array stack serves as the memory of the simulated
    machine}
  END;
BEGIN {main program}
  initialize;
  BLOCK;
  INTERPRET
  END.
```

그림 3. pascal-s의 구조

Fig. 3. Structure of pascal-s.

2. pascal-s의 컴파일러와 의사 코드

a) 컴파일러

pascal-s의 컴파일 요소는 병행 프로그램을 시뮬레이션 할 수 있는 요약된 내용만을 취급하였으며, 식별자(identifier)들에 대해서도 간단한 자료 구조를 사용하였다.

표 1은 pascal-s의 컴파일 요소를 나타낸 것이고, 그림 4는 식별자 테이블 TAB(identifier table)의 자료 구조와 그 구성 요소를 나타낸 것이다.

이때 TAB은 블럭의 내포 구조(nest structure)로 스택에 저장되며, 각 TAB은 링크 필드의 인덱스를 이용하여 액세스한다.

표 1. Pascal-s의 컴파일 요소

Table 1. Compiling factors of pascal-s.

컴파일 내용	컴파일 요소
data type	integer, boolean, char
structure	array stack
strings	'letters....' form
operator	integer (+, −, *, div, mod), boolean (and, or, not)
relation	=, 〈, 〉, 〈=, 〉=, 〈〉
declaration	const, type, var, procedure, function
statement	assignment, if, while, repeat, for
*concurrency	COBEGIN / COEND
*synchronization	WAIT(s) / SIGNAL(s)

name	link	obj	typ	ref	normal	level	adr

name : packed array[1..10];
link : index of nesting block;
obj : const, variable, procedure, function;
typ : integer, boolean, chars, arrays;
ref : index into the block table;
normal : boolean of process;
level : level of nesting block;
adr : index into code table;

그림 4. 식별자 테이블의 자료 구조
Fig. 4. Data structure of identifier table.

b) 의사 코드

pascal-s의 각 기능을 수행하는 의사 명령(pseudo instruction)은 컴파일러에 의해 의사 코드(p-code)로 변환된다. 이때 의사 코드는 명령부(operation field)와 두 개의 오퍼런드부(operand field) X와 Y로 구성되며, X는 의사 코드가 포함된 블럭의 내포 구조에 대한 정적 레벨(static level)을 가리키고, Y는 명령부에 전달할 자료의 주소(address)를 가리킨다.
그림 5는 의사 코드의 구조를 나타낸 것이다.

operation code	X (level)	Y (address)

그림 5. 의사 코드의 구조
Fig. 5. Structure of pseudo code.

인터프리터는 컴파일러에 의해 데이타를 정수화하여 배열 스택(array stack)에 저장한 code (pcode), tab(identifier table), btab(block table) 등을 해석하여 실행한다. 표 2는 pascal-s에 설정된 의사 명령을 컴파일러에 의해 의사 코드로 변환하여 정수화한 변수와 그 기능을 요약하여 나타낸 것이다.

c) 변수 액세스

일반적으로 블럭 구조화 언어(block-structured language)에서 각 블럭은 내포된 구조(nested structure)로 메모리 세그먼트(memory segment)에 저장되므로 블럭 구조(block structure)에서 변수의 주수는 순서쌍(level, address)로 나타낸다. 이 때 level은 변수가 선언된 블럭의 내포된 깊이(nesting depth)를 가리키고, address는 블럭이 할당된 메모리 세그먼트 내의 변수에 대한 오프셑(offset)를 가리킨다.

표 2. pascal-s에 설정된 의사 코드의 정수 형태
Table 2. Integer type of predefined p-code in pascal-s.

정수화한 의사 코드	기 능
0	load address
1	load value
2	load indirect
4	cobegin
5	coend
6	wait
7	signal
8	end of line and end of file
18	markstack
19	procedure call
31	halt
32	return procedure
33	return function
62	readln
63	writeln
others	omitted in the this table

블럭 구조화 언어인 pascal-s에서는 의사 명령을 수행하기 위하여 변수의 액세스 방법으로 display를 이용한다. 이때 display는 블럭의 내포된 레벨(nesting level)과 스택 세그먼트의 배열의 인덱스(index of array)에 대한 정보를 갖는다.

따라서 변수의 상태가 (level, address) = (x, y)이면, 변수의 주소는 display[x]+y로 계산하여 구할 수 있다.

그림 6은 주 프로그램 P의 프로시듀어 r과 s(s는 r의 프로시듀어)를 display에 의해 액세스하는 과정을 나타낸 것이다. 이때 dynamic link는 블럭의 호출 순서(p → r → s)를 나타내고, static link는 변수의 액세스 순서(s → r → p)를 나타내고 있다.

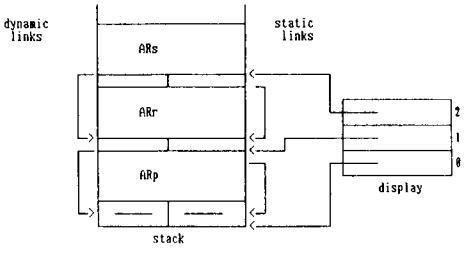


그림 6. display에 의한 변수의 액세스 방법
Fig. 6. Access method of variable by display.

3. 프로시듀어 호출

pascal-s 인터프리터는 프로시듀어를 포함하지 않는 정상적인 프로그램(normal program)을 수행할 때, pc(program counter), t(top pointer), b(bottom pointer), stacksize, display 등의 필수적인 정보가 저장된 프로세스 테이블 PTAB(process table)을 이용하여 실행한다.

그러나, 프로시듀어를 포함할 경우는 프로시듀어 호출(procedure call)에 대한 정보와 이것을 저장할 기억 장소(memory)가 필요하므로 이들의 정보를 의사 코드의 오퍼런드(operand)가 저장된 스택에 할당하여 실행한다. 이때 프로시듀어의 정보를 저장하기 위해 할당된 기억 장소를 활동 레코드 AR(Activation Record)라 하며, 활동 레코드에는 그림 7과 같은 다섯 가지의 프로시듀어에 대한 정보가 저장된다.

```
AR[0]=function result;
AR[1]=return address;
AR[2]=static link;
AR[3]=dynamic link;
AR[4]=table pointer;
```

그림 7. 프로시듀어에 대한 활동 레코드의 고정된 요소

Fig. 7. Fixed factors of activation record for procedure.

한편, pascal-s 컴파일러는 프로시듀어를 컴파일하여 블럭 테이블 BTAB(block table)을 구성하며, 마크스택(markstack) 명령이 BTAB을 스택에 할당하고, 호출(procedure call) 명령이 BTAB을 인터프리터에 전달하여 실행한다.

그림 8은 마크스택 명령에 의해 BTAB이 스택에 할당된 상태를 나타낸 것이며, 여기서 BTAB[1]은 pascal-s의 선언부에서 정의된 식별자들의 "environment"에 대한 블럭이고 BTAB[2]는 주 프로그램의 엔트리를 포함하고 있는 블럭이다.

Ⅳ. 병행 프로그램의 구현

1. 병행 시스템의 구성

일반적으로 여러 프로세스들을 병행으로 실행하려면 각 프로세스의 상태를 나타내고 있는 프로세스 이미지(process image)와 이들 프로세스의 지역 변수(local variable)를 저장하고 있는 스택의 두 가지 필수적인 정보가 필요하다.

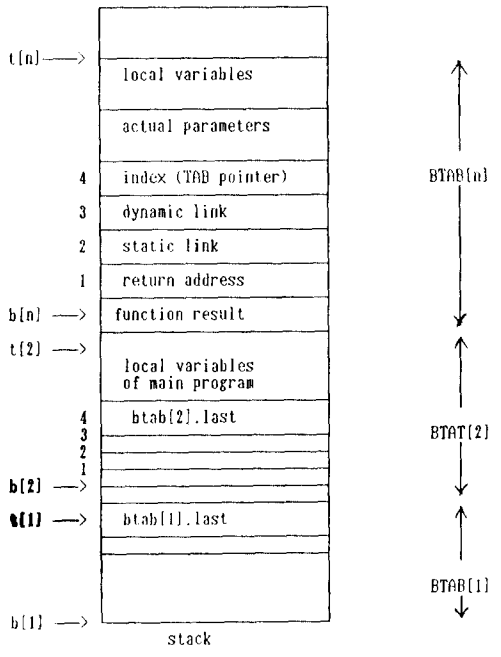


그림 8. 블럭 테이블이 스택에 할당된 상태

Fig. 8. Allocated status of stack for BTAB.

이때, 특정 프로세스를 실행하려면 스택에 저장된 프로세스 이미지를 물리적 레지스터에 적재하여 실행한다.

또한, pascal-s의 프로세스 이미지는 t, b, pc, stacksize, display등이므로, 이들 요소로 각 프로세스에 대한 PTAB (process table)을 구성한다.

이때, 특정 프로세스를 실행하려면 "WITH PTAB[curpr] DO"문에 의해 해당 프로세스의 PTAB을 인터프리터에 적재하여 실행한다.

한편, pascal-s의 컴파일러는 단일 스택(single physical array stack)을 여러 개의 스택 세그먼트(stack segment)로 분할하여 각 세그먼트에 병행 프로세스들의 PTAB을 하나씩 할당하므로서 스택 테이블(stack table)을 구성한다. 이러한 스택 테이블을 병행 시스템(concurrent system)이라고 한다.

병행 시스템에서 프로세스를 실행하려면 스택 테이블의 인덱스를 이용하여 실행할 프로세스를 액세스하고, 실행 알고리즘에 의해 수행한다.

그림 9는 각 프로세스 테이블 PTAB의 구조를 나타낸 것이고, 그림 10은 병행 프로세스들의 PTAB을 스택에 할당하여 구성한 병행 시스템을 나타낸 것이다.

t	b	pc	stacksize	display	suspend	active

t : 스택의 top 포인터
b : 스택의 bottom 포인터
pc : program counter
stacksize : 프로세스가 저장된 스택의 크기
display : 실행중인 프로세스의 레벨
suspend : wait 중인 세마포어 변수
active : 프로세스의 실행 / 중단 상태 표시 flag

그림 9. 프로세스 테이블의 구조
Fig. 9. Structure of process table

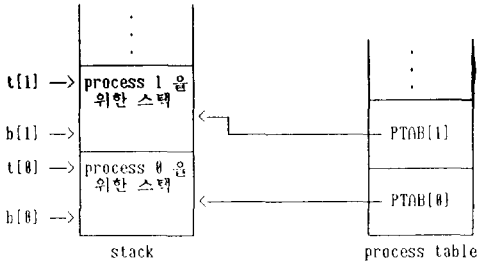


그림10. 병행 프로세서들에 대한 병행 시스템
Fig. 10. Concurrent system for concurrent processes.

2. 병행 프로세스의 동기화 알고리즘

pascal-s에서 프로세스들의 병행성은 "COBEGIN p1;p2;...pn COEND"문으로 표현한다.

이때, COBEGIN문의 기능은 COBEGIN문과 COEND문 사이에 지정된 프로시듀어 $P_1, P_2, \cdots, P_n$을 동시에 호출하도록 지시하는 명령이고, COEND문의 기능은 주 프로세스의 실행을 정지하고, 병행 프로세스들을 동시에 실행하여 그 실행이 완료되면, 다시 주 프로세스가 실행되도록 지시하는 명령이다.

그림 11은 병행 처리문 "COBEGIN p1;p2;...pn COEND"의 수행 과정을 나타낸 것이다. 이때 pflag (concurrent call flag)는 COBEGIN문에 의해 병행프로세스의 호출을 지시하는 변수이다.

```
단계 1 : IF pflag THEN 〈concurrent procedure call〉
                  ELSE 〈단계 5〉
단계 2 : PTAB[0]=false, PTAB[i]=true
단계 3 : PTAB[1], PTAB[2], .... PTAB[n]의 병행 실행
단계 4 : IF curpr=0 THEN 〈procedure exit〉
                  ELSE 〈단계 3〉
단계 5 : PTAB[0]=true, PTAB[i]=false
```

그림11. "COBEGIN p1, p2…, pn COEND"의 수행 알고리즘
Fig. 11. Executing algorithm of "COBEGIN p1, p2; …; pn COEND."

3. 병행 프로세스의 구현 알고리즘

pascal-s의 병행 프로그램을 실행하려면, 병행 프로세스들의 상호 배제 가능성과 실행할 프로세스의 선택 가능성 그리고 프로세스 대기의 제한성 문제가 해결되야 한다.

본 논문에서는 첫째, 병행 처리문 "COBEGIN p1;p2;... ;pn COEND"으로 실행할 병행 프로세스 p1, p2,...,pn을 설정하고, 둘째, 병행 프로세스들 사이의 동기화는 세마포어 알고리즘을 이용하여 실행하며, 프로세스들 사이의 교대는 stepcount를 이용하여 병행 프로세스의 구현 알고리즘을 이용한다.

그림 12는 병행 프로세스들의 구현 알고리즘을 나타낸 것이다.

그림 13은 pascal-s의 인터프리터에 의해 주 프로세스와 병행 프로세스 사이의 동기화 실행과 병행 프로세스들의 병렬 실행을 시뮬레이션하기 위한 예제

```
Initialize PTAB  (*process table*)
  {PTAB[0] for main program}
  {PTAB[curpr] for procedures}
REPEAT
  IF PTAB[0], active THEN curpr := 0
  ELSE IF stepcount = 0 THEN chooseprocess
       ELSE stepcount := stepcount - 1;
  WITH PTAB[curpr] DO
  BEGIN ir := code[pc]; pc := pc + 1 END;
  IF pflag THEN BEGIN
                  IF markstack THEN npr := npr + 1;
                  curpr := npr
                END;
  WITH PTAB[curpr] DO
    CASE ir.f OF  {ir.f is p-code's variable number}
      ir.f  :  p-code instruction;
        .          .
        .          .
        .          .
        .          .
UNTIL ps 〈 〉 run;
```

그림12. 병행 프로세스들의 구현 알고리즘
Fig. 12. Implementing algorithm of concurrent processes.

프로그램이다. 그리고 표3은 주어진 예제 프로그램을 pascal-s에 의해 시뮬레이션한 결과로서 주 프로세스 PTAB[0]와 병행 프로세스 PTAB[1], PTAB[2], PTAB[3] 사이의 동기화 과정 및 병행 프로세스들의 병행 처리 결과를 나타낸 것이다.

```
PROGRAM concurrentexecution;
VAR   i, j, k : integer;
PROCEDURE samplel;
    BEGIN
      for i := 1 to 3 do
      writeln('process-1')
    END;
PROCEDURE sample 2;
    BEGIN
      for j := 1 to 2 do
      writeln('process-2')
    END;
PROCEDURE sample 3;
    BEGIN
      for k := 1 to 5 do
      writeln('process-3')
    END;
BEGIN
    COBEGIN samplel; sample 2; sample 3 COEND
END.
```

그림13. 병행 실행을 위한 예제 프로그램
Fig. 13. Sample program for concurrent execution.

표 3. 예제 프로그램의 시뮬레이션 결과
Table 3. Result of simulation for sample program.

```
interpret
process - 1
process - 2
process - 3
process - 1
process - 2
process - 3
process - 1
process - 3
process - 3
process - 3

process      active      suspend      pc
   0         TRUE          0          36
   1         FALSE         0           0
   2         FALSE         0           0
   3         FALSE         0           0

   global variables
   j  =  3
   j  =  2
   K  =  5
```

4. 병행 실행의 결과 분석

여러 프로세스들이 병행으로 실행하기 위해서는 첫째, 프로세스들의 병행성 표현방법, 둘째, 프로세스 사이의 상호 작용을 위한 동기화 방법, 셋째, 프로세스 간의 정보 교환 방법 등이 해결되야 한다. 따라서 본 논문에서는 pascal-s라는 병행 언어를 구성하였으며, 이 언어를 이용하여 프로세스들의 병행성 표현과 동기화 방법 및 정보 교환 방법을 병행 프로그래밍 기법으로 구현하였다.

다음은 병행 프로그래밍 기법으로 프로세스들의 병행 실행을 구현한 결과 분석이다.

1) 병행언어 pascal-s는 블럭 구조로 프로그램을 구성하므로서 프로세스들의 병행성 표현과 병행 프로그램의 작성및 실행이 용이하였다.

2) 주 프로세스와 병행 프로세스 사이의 동기화는 COBEGIN문에 의한 프로시듀어 호출 변수 pflag 를 이용하였으며, 병행 프로세스들의 동기화는 실행 알고리즘에 의해 동기화를 수행하였다.

표4는 주 프로세스 process-0와 병행 프로세스 process-1, process-2, process-3 사이에 초기 실행 단계, 병행 실행 단계, 병행 종료 단계에서 동기화의 실행 과정을 나타낸 것이다.

표 4. 병행 프로세스들 사이의 동기화 실행 단계
Table 4. Executive step of synchronization between concurrent processes.

실행단계 / 프로세스	초기실행단계	병행실행단계	병행종료단계
process-0	true	false	true
process-1	false	true	false
process-2	false	true	false
process-3	false	true	false

3) 프로세스들의 병행 실행은 병행 처리문 "COBEGIN p1; p2; ... COEND"에서 지정된 프로시듀어 p1, p2, ..., pn을 동시에 호출하여 실행 알고리즘에 의해 병행으로 실행하였다.

표5는 병행 프로세스 process-1, process-2, process-3가 서로 독립적이므로 병렬로 실행된 결과이며, 실행 횟수는 전역 변수(global variable)를 이용하였다.

표5에 의하면 순차적으로 실행할 경우 프로시듀어가 10회를 실행해야 하는데, 병렬로 실행하므로서 5회의 수행으로 완료됨을 알 수 있다.

표 5. 병행 프로세스들의 병행 실행
Table 5. Concurrent execution of concurrent processes.

병행프로세스 실행횟수	process-1	process-2	process-3
1	active	active	active
2	active	active	active
3	active		active
4			active
5			active

표 6. pascal-s 컴파일러에 의해 생성된 TAB과 CODE
Table 6. Created TAB and CODE by pascal-s comfiler

구 분 종 류	CODE	TAB
reserved number	63	14
created number	x	y

4) pascal-s의 컴파일러는 주어진 병행 프로그램을 컴파일하여 식별자 TAB과 의사 명령 CODE를 생성하고, 이들을 스택에 저장한 후, 인터프리터가 저장된 스택의 요소들을 해석하여 프로세스의 실행 알고리즘으로 수행한다.

표 6은 pascal-s 컴파일러가 병행 프로그램을 컴파일하여 구성한 TAB과 CODE 수를 나타낸 것이다. 이때 x와 y는 예제의 병행 프로그램에 대한 TAB 과 CODE의 갯수이다.

5) 병행 프로세스들 사이의 상호 배제의 가능성, 진행의 가능성 및 대기의 유한성 문제를 해결하기 위하여 동기화 구현 알고리즘을 다음과 같은 방법으로 작성하였다.

첫째, 주 프로세스와 병행 프로세스 사이의 동기화는 pflag를 사용하였고,

둘째, 실행할 프로세스의 선정은 병행 처리문 "COBEGIN p1;p2;...;pn COEND"에 의해 동시에 p1, p2, ... pn을 호출하며,

셋째, 프로세스 사이의 교대는 프로세스의 수행 단계 수(stepcount)를 제한하여 제한된 횟수를 초과하면 실행 중인 프로세스를 suspend시키고, 다른 프로세스를 실행 상태로 하므로서 프로세스의 교대가 이루어지도록 하였다.

본 논문에서는 네 개의 병행 프로세스로 병행 시스템을 구성하여 이들의 병행 실행과 동기화를 시뮬레이션 하였다.

앞으로 병행 프로그래밍 언어 pascal-s의 기능을 확장하므로서 병행 시스템의 구성이 확대될 수 있고, 병행 프로세스의 실행 알고리즘을 개선하므로서 프로세스들의 병행 실행을 효율화할 수 있다.

V. 결 론

최근 성능이 뛰어난 프로세서의 개발과 컴퓨터 설계 기술의 향상으로 인하여 컴퓨터 시스템의 구성이 여러 개의 프로세스를 결합하는 방식으로 전환되고 있다.

따라서 이러한 시스템을 지원하기 위한 여러 가지 연구 중의 하나가 프로세스들의 병행 실행에 관한 연구이다.

본 논문에서는 프로세스들의 병행 실행을 동기화 구조를 적용하여 병행 프로그래밍 기법으로 다음과 같이 구현하였다.

첫째, 여러 프로세스들의 병행 프로그래밍을 위해 기존의 pascal 언어에 병행성 기능을 갖는 "COBEGIN/COEND"문과 동기화 기능을 갖는 "WAIT(s)/SIGNAL(s)"문을 부가하여 pascal-s라는 병행 프로그래밍 언어를 구성하였다.

둘째, 프로세스들의 병행성은 병행 처리문 "COBEGIN p1;p2;...;pn COEND"으로 표현하여 병행 프로시듀어 p1, p2, ...,pn을 동시에 호출하였다.

셋째, 병행 프로세스들 사이의 동기화는 세마포어를 적용하여 구성한 동기화 알고리즘으로 실행하였다.

본 논문에서는 프로세스들의 병행 실행을 동기화 구조를 적용하여 병행 프로그래밍기법으로 구현한 결과 다음과 같은 결론을 얻었다.

① 병행 프로그래밍 언어 pascal-s는 블럭 구조로 프로그램을 구성하므로 병행 프로그램의 작성과 오류 수정이 용이하였다.

② 병행 프로그램의 실행은 pascal-s 컴파일러가 병행 프로그램을 컴파일하여 식별자들의 TAB과 의사 명령 CODE를 생성하고, 이들을 블럭 단위로 스택에 저장한 다음, 인터프리터가 이들 TAB과 CODE를 해석하여 실행 알고리즘으로 프로세스들을 병행으로 실행하기 때문에 프로그램의 실행이 쉽다.

③ 프로세스 사이의 상호 배제 가능성과 프로세스의 진행성 및 프로세스간의 교대 문제를 해결하기 위하여 첫째, 프로세스들 사이의 동기화는 세마포어를 적용하고, 둘째, 프로세스의 선정은 병행 처리문 COBEGIN/COEND"으로 병행 프로시듀어들을 호출하였으며, 셋째, 프로세스간의 교대는 프로세스의 수

행 단계 수를 제한하여 설정된 단계수를 초과하면 프로세스간의 교대가 이루어지도록 하였다.

이상의 과정으로 프로세스들을 병행 실행한 결과 동기화가 효율적으로 이루어졌다.

④ 고급 언어에 의한 병행 프로그래밍의 기법은 컴퓨터 시스템의 구성에 관계없이 병행 프로세스의 실행 알고리즘에 의해 프로세스들의 병행 수행이 용이하였다.

앞으로 병행 프로그램을 위한 병행 프로그래밍 언어의 연구도 계속되야 하겠다.

參 考 文 献

[1] Peterson, J.L., Silberschatz, A., "Operating system concepts," Addison-Wesley Pub. Co., 1985.

[2] Andrews, G.R., Schneider, F.B., "Concepts and Notations for concurrent programming," Com. Survey, vol. 15, no. 1, pp. 3-43. 1983.

[3] Ben-Ari, M., "Principles of concurrent programming," Prentice-Hall International, Inc., 1982.

[4] Hansen, P.B., "Operating system principles," Prentice-Hall, cliffs, N.J., 1973.

[5] Bloom, T., "Evaluating synchronization mechanisms," Proc. of 7th SOSP , ACM, pp. 24-32, 1979.

[6] Hansenm, P.B., "Distributed process A concurrent programming concept," CACM, vol. 21, no. 11, pp. 934-941, 1978.

[7] Hoare, C.A.R., "Monitor: an operating system structuring concept," CACM, vol. 17, no. 10, pp. 549-557, 1974.

[8] Hoare, C.A.R., "Communicating sequential process:, CACM, vol. 21, no. 8, pp. 666-667, 1978.

[9] Lamport, L., "Solved problems, unsolved problems and non-problems in concurrency,' Proc. of 3rd PDOC, OSR, vol. 19, no. 4, pp. 34-44, 1985.

[10] Hansen, P.B., "The architecture of concurrent programs," Prentice-Hall, Englewood Cliffs, 1977.

[11] Lamport, L., "A new solution of dijkstra's concurrent programming problem," ACM, no. 17, pp. 453-455, 1974.

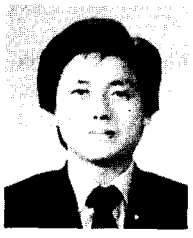
[12] Holt, R.C., Graham G.S., "Structured concurrent programming with operating systems application," Addison-Wesley Publishing Company, 1986.

[13] 안순신, "concurrent 프로그램의 새로운동기화 구조," 한국 정보과학회 논문지, vol, 11, no. 4, pp. 245-255, Nov., 1984.

[14] 박향재, 신현식, "시간 구동형 병행 파스칼 인터프리터를 이용한 분산/병렬처리 시스템 시뮬레이션," 한국 정보과학회 논문지, vol. 15, no. 5, pp. 412-421, Oct. 1988.

[15] 이동춘, 오병균, 이상범, "프로그램에 의한 구조적 소프트웨어의 계층적 개발," 한국 정보과학회 논문지, vol. 14, no. 3, pp. 225-233, Aug., 1987.

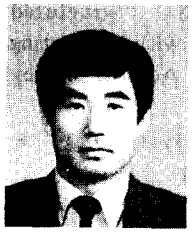
著 者 紹 介



吳 炳 均(正會員)

1945년 5월 5일생. 1970년 공주 사범대학 수학과 졸업(이학사). 1983년 조선대학교 대학원 전자계산 전공(공학석사). 단국대학교 대학원 전자공학과 박사과정 재학중. 1985년 4월~현재 국립 목포대학교 전산 통계학과 조교수. 주관심분야는 컴퓨터 구조, 운영체제, 자료 구조 등임.

●



金 成 洛(正會員)

1948년 5월 13일생. 1973년 명지대학교 전자공학과 졸업(공학사). 1979년 명지대학교 대학원 전자공학과(공학석사). 단국대학교 대학원 전자공학과 박사과정 재학중. 1981년~현재 관동대학교 정보 처리학과 부교수. 관심분야는 컴퓨터 구조, 음성 인식 등임.

李 相 範(正會員)

1951년 2월 3일생. 1974년 연세대학교 전자공학과 졸업(공학사). 1978년 서울대학교 대학원 전자공학과(공학석사). 1986년 연세대학교 대학원 전자공학과 공학박사 학위 취득. 1983년~1984년 미국 IOWA대학 객원교수. 1979년~현재 단국대학교 전자공학과 부교수. 주관심분야는 컴퓨터 구조, 마이크로 프로세서 응용, 음성 인식 등임.