분산 프로그래밍 언어의 kernel에 관한 연구

○김 영 석, 이 광 희, 안 순 신
고 려 대 학 교 전 자 전 산 공 학 과

A Study on the Kernel Supports for a Distributed Programming language

Young-Seok Kim, Kwang-hui Lee, Sun-Shin An
Dept. of Elect. & Comp. Eng., Korea Univ.

ABSTRACT

In designing and implementing of a distributed system, a programming language which can describe and implement the various interactions between distributed processes in distributed systems is indispensible. High level language constructs such as concurrency, process synchronization between distriuted processes and mutually exclusive access to common data could be built in a distributed programming language under the proper support of a language kernel. In this paper, we studied the language constructs a distributed programming language must have and specified the kernel supports necessary in implementing that high level language constructs.

1. 서 론

컴퓨터 통신 기술의 급격한 발전에 힘입어 지리적으로 분산된 여러 컴퓨터들을 이용하여 컴퓨터 네트웍을 구성함으로서 자원의 공유, performance. 확장성, 신뢰도 면에서 사용자 에게는 보다 나은 service 를 제공하는 분산 시스템(Distributed Computing System)의 연구 개발이 활발히 진행되고 있다.

이러한 분산 시스템의 software를 구성하는데 있어서 여러 노드(node)에 분산된 대상(object) 들의 상호 관계와 분산 시스템의 특성을 기술하고 구현하기 위한 프로그래밍 언어의 연구 개발은 필수적으로 따르게 된다.

분산 프로그래밍 언어가 지녀야 할 필수적인 요소로는 분산 프로세스간의 통신과 동기를 위한 IPC(Interprocess Communication)기능, 분산 software의 structuring을 위한 기능, information hiding과 protection을 위한 abstraction 기능들을 들수 있으며 이러한 기능의 구현은 반드시 kernel의 적절한 지원아래 이루어 지게 된다.

따라서 본 연구에서는 분산 시스템의 설계 및 구현시 이용되는 분산 프로그래밍 언어(Distributed Programming Language)의 특성과 이의 구현에 필요한 kernel level 에서의 support를 고찰 해 보도록 한다.

2. 분산 시스템을 위한 프로그래밍 언어

분산 프로그램(Distributed Programs)은 지리적으로 분산된 서로 다른 노드(node)에 존재하면서 서로 통신하는 프로그램 모듈(Program Module)로 정의될 수 있다. 이러한 분산 프로세스(Distributed Process)들의 통신 방법에 의하여 분산 프로그래밍 언어를 분류해 보면 다음과 같다.

(1) Concurrent Programming Language(CPL)

공통 변수(shared variable)에 바탕을 둔 monitor concept을 통해서 resource의 공유에 의해 분산 프로세스간의 통신이 이루어 지며 Concurrent Pascal, Modula-2, Clu, Mesa, Concurrent Euclid 등이 그 예이다. 이들 언어는 분산 프로세스 간의 상호 접속을 기술할 수 있는 매우 고급의 언어 특성(language constrcts)을 가지고 있다.

(2) Message Passing Language(MPL)

공통 변수에 근거한 통신 방식은 공통 memory를 갖는 시스템에서의 프로세스 관계를 기술하기에는 적당하나 지리적으로 분산된 프로세스들이 하나의 task를 수행하기 위해 서로 협력하는 분산 시스템에는 적당하지 못하다. 따라서 이러한 환경에는 통신 네트웍을 통한 message passing에 의한 통신 방식이 적당하며 CSP(Communicating Sequential Process), Gypsy, PLITS(Programming Language In The Sky)등을 들수 있다.

(3) Remote Procedure Call Language(RPL)

프로그램 모듈이 집중되어 있든지 혹은 분산되어 있든지 이에 상관 없이 remote site에 대한 procedure call 형태로 통신이 일어나는 것으로서 Argus, Ada, DP(Distributed Process)등이 이에 속한다.

3. 분산 프로그래밍을 위한 language kernel의 support

Language kernel은 high-level language construct를 support 하기 위한 software kernel로서 operating system kernel과는 별개의 것이며, user program이 compile될때 같이 linking된다.

Language kernel은 simulation kernel과 bare machine kernel로 나눌 수 있다. bare machine kernel은 하나의 완전한 kernel로서 operating system전반의 모든 기능을 갖추고 있으며 interrupt와 device I/O를 처리하고 operating system의 support없이 cpu time 을 고유함으로서 program을 수행한다. 반면에 simulation kernel은 operating system의 support를 받으며 하나의 process로 cpu time을 공유하며 program을 수행 시킨다.

다음에서 언급하고자 하는 language kernel은 UNIX OS 하의 Concurrent Euclid를 위한 simulation kernel로서 execution efficiency를 위해서 assembly language로 구성되어 있으며, object module로 존재하다가 source program의 compile시 같이 linking되어 language의 concurrency feature를 support하게 된다.

이를 그림으로 표시하면 다음과 같다.

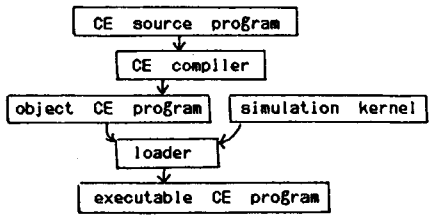

그림 3-1 CE 프로그램의 생성

CE program은 UNIX OS 하의 하나의 process로 cpu time을 공유하며, 하나의 CE program은 여러개의 CE process로 simulation kernel의 support에 의해서 자체적으로 switching되면서 수행된다. 따라서 UNIX OS는 CE program을 하나의 UNIX process로 간주할뿐 몇개의 CE process로 구성되어 있는지는 알수 없다.

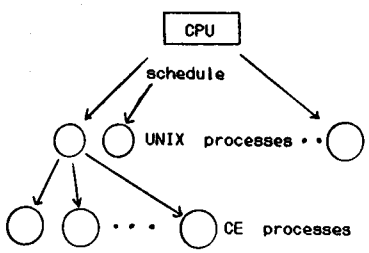

그림 3-2 CE 프로세스와 UNIX 프로세스

이상에서 알수 있듯이 simulation kernel은 high-level language feature를 구현하기 위한 것으로서 분산 프로그래밍 언어(distributed programming language) 구현의 필수적인 요소라 할수 있다.

다음에서는 high-level language construct를 support하기 위한 simulatioon kernel의 내부 기능을 Concurrent Euclid simulation kernel을 이용해서 살펴 보기로 한다.

4. Simulation kernel supportsfor Concurrent Euclid under UNIX OS

(1) Concurrent Euclid 언어

Concurrent Euclid 언어는 Toronto 대학에서 개발한 병렬 프로그래밍 언어(concurrent programming language)로서 strong type checking을 통한 reliable하고 verifiable한 system software용으로 분산 프로그래밍을 위한 여러가지 high-level language construct를 가지고 있다.

Concurrent Euclid compiler는 multipass compiler로서 총 네개의 pass로 구성되며 네번째 pass인 code generator는 여러 machine (VAX, MC68K, PDP11, MC6809, NS16032) 을 support함으로써 매우 높은 retargetability를 갖는다.

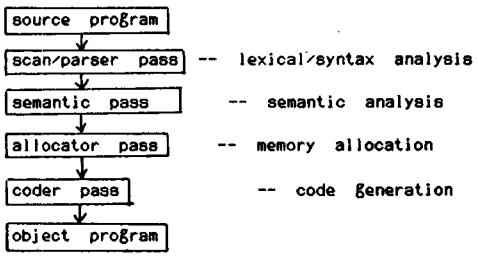

그림 4-1 CE 컴파일러의 구조

분산 프로그래밍을 위한 language feature로는 concurrent activity를 위한 process, 공유 데이타의 mutual exclusive access를 위한 monitor, 그리고 process synchronization을 위한 signal/wait construct를 들수 있으며 이를 support하기 위한 language kernel support를 다음에서 살펴 보기로 한다.

(2) Simulation kernel의 구조

Language kernel의 중요한 목적은 각각의 process로 하여금 cpu time을 공유함으로써 각각의 process에 대해 virtual cpu time을 실현하는데 있으며, IPC(Interprocess Communication)을 위한 process/process interface와 device management를 위한 process/device interface 기능을 가지고 있다.

Kernel은 process의 concurrent한 execution과 monitor를 support하기 위하여 process descriptor로 이루어진 몇개의 queue를 manipulation한다.

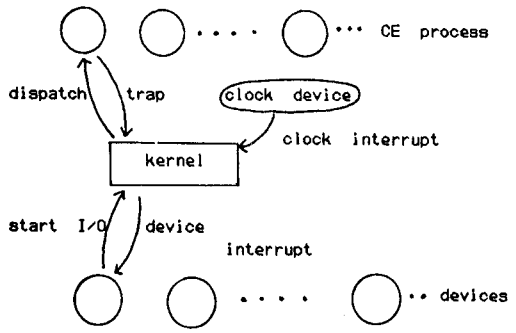

그림 4-2 kernel의 구조

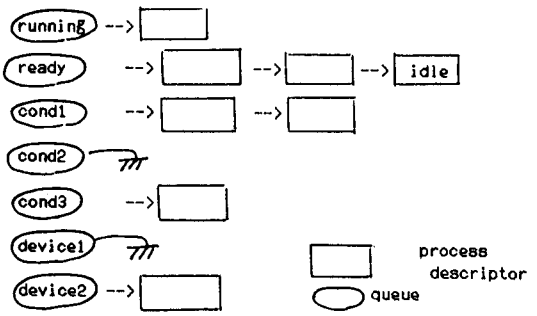

그림 4-3 queue의 구조

Process Descriptor는 각각의 process마다 할당되며 process switching에 필요한 process의 instruction counter, register(general, floating, address register) information, priority value등을 내용으로 한다.

위 그림에서 running process는 하나이며 ready queue에는 세개의 process가 priority value에 의해 ordered되어 있으며, condition과 device queue는 FIFO 원리에 의해 구성되어 있다.

(3) Concurrency control

앞에서 언급했듯이 CE program 내의 여러 process는 time slice에 의해 cpu time을 공유함으로써 virtual cpu time을 실현한다. ready queue는 priority value에 의해 ordered되어 있으며 같은 priority value일때는 FIFO queue의 역할을 하게 된다.

Kernel entry인 TimeSlice에서는

```
TimeSlice :
        clock interrupt
        save status of the running
        disable interrupt
        insert the running into ready queue
        remove a new from ready queue
        restore status of the new
        enable interrupt
```

clock interrupt에 의한 process switching을 함으로써 concurrency를 support한다. 만약 interrupt된 process의 priority value가 ready queue의 어느 process보다 높을때 interrupt된 process는 곧바로 다시 switching됨으로써 process status를 save하고 restore하는데 필요한 overhead를 줄일수 있다.

(4) Process Synchronization

Process의 wait(event)와 signal(event)를 support하기 위해 condition queue가 존재하며 이 condition queue는 실제로 process descriptor의 linked list에 의해 구현된다.

Kernel entry인 wait와 signal은 shared resource에 대한 mutual exclusive access를 제공한다. 즉 한 process가 resource를 얻으면 다시 버릴때 까지 이를 원하는 다른 process는 wait 상태에 들어가게 됨으로써 process간의 synchronization을 이루게 된다.

```
Wait(cond) :
        save status of the running.
        insert the running into
              condition queue 'cond.
        remove a new from ready queue.
        restore status of the new.

Signal(cond) :
        if nonempty(cond) then
              save status of the running.
              insert the running into ready.
              remove a new from
                    condition queue 'cond.
              restore status of the new.
        endif
```

(5) Mutual exclusion

Information hiding과 protection을 support하기 위한 abstraction mechanism으로서 monitor gate를 support한다. 즉 export된 monitor procedure들을 통해서만 monitor내의 resource에 접근할 수 있으며 하나의 process가 export된 procedure entry를 통해 monitor 내부에 있을때 다른 process의 접근을 방지함으로서 common data에 대한 mutual exclusion을 보장할 수 있다.

5. 결 론

본연구에서는 분산 시스템 소프트웨어에 이용되는 high-level concurrent programming language con-

struct를 support하는 simulation language kernel의 기능에 대해서 고찰해 보았다. 현재 이러한 simulation kernel의 support를 받는 분산 프로그래밍 언어(Distributed Programming Language)의 design과 이의 compiler구현에 관한 연구가 병행되어 지고 있으며, 높은 신뢰도 (reliability) 와 보다 향상된 IPC (Interprocess Communication) 기능을 요구하는 분산 시스템의 설계및 구현에 이용될 수 있을 것이다.

```
EnterMonitor :
        disable interrupt

ExitMonitor :
        if firstready<running then
                save status of the running.
                insert the running
                                into ready queue.
                remove a new from ready queue.
                restore status of the new.
        endif;
        enable interrupt.
```

참 고 문 헌

(1) Hoare, C.A.R
"Monitors: An Operating System Structuring Concept" CACM Vol.17, No.10, Oct. 1974

(2) Brinch Hansen
"Distributed Processes: A Concurrent Programming Concept" CACM Vol.21, No.11, Nov. 1978

(3) Hoare, C.A.R
"Communicating Sequential Processes" CACM Vol.21, No.68, Aug. 1978

(4) R.C. Holt and E.D. lazowska
"Structured Concurrent Programming with Operating Systems Applications" Addison-Wesley Pub. Com.

(5) R.C. Holt
"Concurrent Euclid, The UNIX System, and Tunis" Addison-Wesley Pub. Com.

(6) G.L. Reijns, E.L. Dagless
"Concurrent Languages in Distributed Systems" North-Holland 1985

(7) John A. Stankovic
"Reliable Distributed Systems Software" IEEE Computer Society Press, 1985

(8) J.R. Cordy, R.C. Holt
"Concurrent Euclid : Comparison with C and Pascal" University of Toronto

(9) R.C. Holt
"A Short Introduction to Concurrent Euclid" University of Toronto

(10)R.P. Cook
"*MOD - A Language for Disributed Programming" IEEE Trans. on Software Eng. Vol.SE-6, No.6, Nov. 1980