論文 97-22-1-18

네트워크 화일 시스템상에서 이동통신교환기 소프트웨어의 컴파일 관리

正會員 신 재 욱*, 박 광 로**

Compilation Management of the Mobile Switching Software on Network File System

Jae Wook Shin*, Kwang Roh Park** *Regular Members*

요 약

교환 소프트웨어의 규모가 커짐에 따라 이를 컴파일하고 관리하는 데에 많은 시간과 노력이 소요되고 있다. 본 논문에서는 네트워크 화일 시스템상에서 이동통신교환기 소프트웨어의 컴파일을 관리하는 시스템을 구현하고 그 구성 및 기능에 대해 기술하였다. 제안된 컴파일 관리 시스템은 네트워크 화일 시스템으로 구성된 각 시스템에 컴파일 작업을 균등하게 배분하여 분산적으로 수행하게 하고, 컴파일을 수행하는 시스템 및 컴파일 관리자와 소프트웨어 개발자간의 작업 흐름을 원활하게 관리함으로써 전체 컴파일 작업에 소요되는 시간을 크게 감소시켰다.

ABSTRACT

As the scale of the switching software gets large, it takes much time and is hard to compile and to manage it. We implement the compilation management system for the mobile switching software on network file system and describe its structure and functions in this paper. The compilation management system distributes its task of compilation to the each networked system equally to reduce the compilation time and also simplifies the management of the communications among the systems, compilation manager, and the software developers.

*한국전자통신연구원, 이동교환기기연구실
Jaewook Shin, Electronics and Telecommunications Research Institute, Mobile Switching Technology Section
**한국전자통신연구원, 초고속서비스연구실
Kwangroh Park, Electronics and Telecommunications Research Institute, ATM Services Section
論文番號:96075-0228
接受日字:1996年 2月 28日

Ⅰ. 서 론

교환 소프트웨어(switching software)의 규모가 방대해지고 구성이 점점 복잡해짐에 따라 이를 체계적으로 관리하고 종합(integration)하는 데에 많은 노력과 비용이 소요되고 있다. 이동통신교환기 소프트웨어(mobile switching software)의 종합 과정에서 컴파일은 가장 많은 시간을 필요로 하는 단계의 하나로서, 컴파일에 필요한 공통자료(common data) 화일 생성 단계와 블럭(block)의 컴파일 단계로 다시 나눌 수 있다. 블럭은 교환기에서 기능 구현의 기본 단위로서 여러 개의 화일로 구성되어 있으며 각 화일들을 컴파일하고 링크하여 하나의 실행 모듈을 생성한다[1].

이동통신교환기 소프트웨어의 컴파일은 네트워크 화일 시스템(Network File System:NFS)으로 구성된 다수의 유닉스 시스템상에서 이루어 지고 있으며, 각 시스템은 공유된 화일 시스템상에 구축된 동일한 컴파일 환경을 사용하고 있다. 따라서 이와 같은 컴파일 환경에서 백여 개가 넘는 블럭의 컴파일 수행 시간을 최소화하고 컴파일 수행 흐름을 얼마나 용이하게 관리하느냐 하는 것이 이동통신교환기 소프트웨어의 컴파일 작업에서 중요한 문제가 되고 있다.

네트워크 화일 시스템으로 구성된 컴파일 환경에서 컴파일 수행 시간을 최소화하기 위해서는 모든 호스트 시스템에 블럭을 할당하여 여러 블럭을 분산적으로 동시에 컴파일하는 방법이 있다. 이와 같은 방법이 효과적으로 동작하기 위해서는 각 시스템에 할당하는 블럭의 수를 잘 조정하여 컴파일 수행중에 유휴한(idle) 시스템이 없어야 하며, 모든 시스템에서 거의 동시에 컴파일이 완료되어야 한다.

지금까지는 컴파일 관리자가 컴파일을 수행하기 전에 각 블럭을 미리 각 시스템에 고정적으로 할당하는 방법을 사용하여 왔다. 그러나 컴파일을 수행하는 시스템의 성능과 블럭의 크기가 서로 다르고, 또한 컴파일을 수행하는 시스템의 부하(load)가 시간에 따라 동적으로 변하기 때문에 컴파일 수행전에 미리 블럭을 할당하는 방법은 주어진 시스템의 자원을 효율적으로 사용하지 못하는 단점이 있다. 뿐 만 아니라 어떤 시스템이 컴파일 수행중에 크래쉬(crash)되는 경우 그 시스템에 할당된 블럭은 컴파일이 수행될 수 없기 때문에 이들 블럭에 대한 컴파일을 별도로 수행하기 위한 추가적인 시간과 관리 작업이 필요하다.

이와 같은 문제점을 해결하기 위해서 본 논문에서는 네트워크 화일 시스템으로 이루어진 컴파일 환경에서 이동통신교환기 소프트웨어의 컴파일 작업을 자동적으로 관리하여 주는 컴파일 관리 시스템(Compilation Management System On NFS)을 구현하고 그 구성 및 기능에 대해 기술하였다. 제안된 컴파일 관리 시스템은 이동통신교환기 소프트웨어를 컴파일하는데 있어서 한 시스템이 각 시스템의 컴파일 진행 상황에 따라 블럭을 동적으로(dynamically) 할당하도록 하여 컴파일 수행 시간을 대폭 감소시켰으며, 한 시스템이 크래쉬되더라도 전체 컴파일 작업에 미치는 영향이 최소화되도록 하였다. 또한 컴파일 오류 메시지(error message)를 자동으로 추출하여 담당자에게 전송하고, 한 시스템에서 각 블럭에 대한 컴파일 수행 상태를 관리함으로써 컴파일 관리에 소요되는 부담을 크게 감소시켰다.

본 논문의 2 장에서는 이동통신교환기 소프트웨어의 구성과 분리 컴파일 방법에 대해 설명하고 3 장에서는 네트워크 화일 시스템상에서의 컴파일 수행제어 방법에 대해 기술한다. 4 장에서는 컴파일 관리 시스템의 구성과 기능에 대해 설명하며 5 장에서는 블럭 할당 방법의 성능을 평가한다. 그리고 결론을 맺은 후 앞으로의 연구 방향에 대해 기술한다.

Ⅱ. 이동통신교환기 소프트웨어에 대한 개요

2.1 이동통신교환기 소프트웨어 종합

이동통신교환기 소프트웨어 종합 작업은 화일 등록에 의한 소프트웨어 버젼 관리와 등록된 화일을 컴파일하여 목적 시스템인 교환기에 로딩할 수 있는 형태의 화일로 만드는 패키지(package) 제작으로 나눌 수 있다. 기존의 패키지에서 나타난 오류를 수정하거나 새로운 기능을 추가하기 위하여 새로운 버젼의 패키지 제작이 필요할 경우 각 소프트웨어 개발자는 화일 등록 도구를 이용하여 필요한 화일을 등록하며, 이 때 화일 등록 도구는 화일을 등록하는 사람 및 등록되는 화일에 대한 등록이 허가되었는지를 검사하여 허가된 경우에만 등록을 허용하고, 화일을 등록한 사람, 등록된 시간, 등록의 성공 여부 등의 정보를 지정된 화일에 기록한다[2].

패키지 제작은 크게 데이타 베이스 관련 작업과 컴파일 관련 작업으로 크게 나눌 수 있다. 데이타 베이스 관련 작업은 개발자가 등록한 데이타 베이스 자료 화일로부터 교환기에 로딩될 데이타 베이스 화일인 PLD(Processor Load Data)를 생성하는 것이며, 컴파일 작업은 개발자가 등록한 신호 및 입출력 메시지 정의화일로부터 컴파일에 필요한 공통자료 화일을 생성한 후, 각 블럭을 컴파일하여 실행 모듈을 생성하는 것을 말한다. 각 단계에서 소프트웨어 종합 담당자는 등록된 화일에 대한 검증을 수행하여 오류가 존재하는 경우 담당자에게 통보하여 재등록하도록 한다. 각 블럭에 대한 컴파일 작업이 모두 완료되면 앞에서 생성된 PLD 및 실행모듈을 교환기에 로딩하여 필요한 기능 시험을 수행할 수 있게 한다.

2.2 이동통신교환기 소프트웨어의 구성

이동통신교환기 소프트웨어의 구성은 그림 1과 같다. 이 중에서 소프트웨어 개발자가 제작하여 등록하는 화일에는 데이타베이스 초기자료 화일(*.dg), 데이타베이스 릴레이션 정의화일(*.db), 블럭별 프로세스(process) 선언 화일(B2P), 입력메시지 정의화일(*.imd), 출력메시지 정의화일(*.omd), 신호메시지 정의화일(*.md), 라이브러리 소스(libsrc/), 블럭 소스(src/) 및 사용자 정의 공통자료 화일(include/)이 있으며, 소프트웨어 종합 도구를 이용하여 이들 화일로 부터 생성한 후 소스의 컴파일에 사용하는 화일에는 데이타베이스 카탈로그 화일(catalog/), 시스템 공통자료 화일(include/), 입출력 메시지 관련 공통자료 화일(mms/), 신호메시지 식별(identity) 및 정의 화일(sig/), 그리고 라이브러리 화일(lib/)이 있으며, 최종적으로 교환기 시스템에 로딩(loading)되는 화일에는 데이타베이스 화일(PLD), 입출력 메시지 양식 자료 화일(*.DATA), 블럭별 실행모듈(a.out)이 있다. 여기서 '*'는 와일드 카드(wild card)의 의미로 사용하였다.

블럭은 교환기에서 기능 구현의 기본 단위로서 하드웨어 블럭과 소프트웨어 블럭으로 나눌 수 있다[1]. 각각의 소프트웨어 블럭은 교환기내 하나 이상의 지정된 프로세서에 로딩되어 수행되는 실행모듈을 하나씩 생성하며, 경우에 따라서 두 개 이상의 블럭이 하나의 실행모듈을 생성하기도 한다. 따라서 소프트웨어 블럭과 교환기 시스템과의 관계는 그림 2와 같이 나타낼 수 있다. 소프트웨어 블럭은 다시 여러 개의 화일로 구성되며 각 화일은 대부분이 CHILL (CCITT High Level Language) 언어로 작성되나 블럭에 따라서 C 또는 어셈블리 언어로 작성되기도 한다.

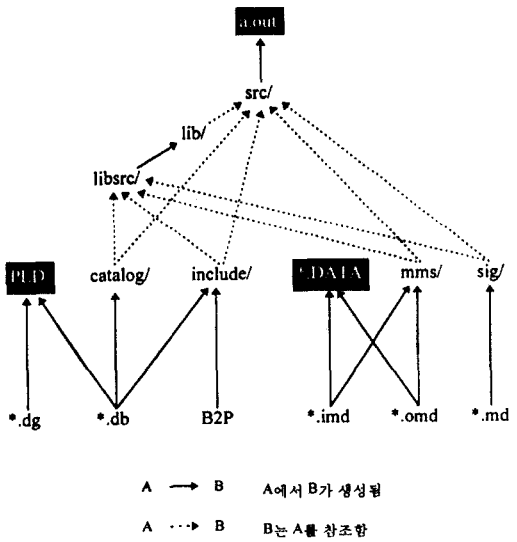


그림 1. 이동통신교환기 소프트웨어의 구성
Fig. 1 The structure of the mobile switching software

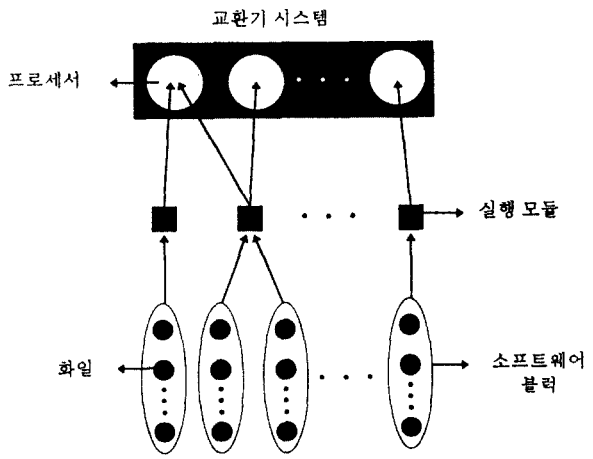


그림 2. 소프트웨어 블럭과 교환기 시스템과의 관계
Fig. 2 The relation between software block and exchange system

2.3 분리 컴파일(Separate Compilation)

CHILL은 강한 형 검사(strong type checking), 정보 은닉(information hiding), 구간적 프로그래밍(piece-wise programming), 병행 프로그래밍(concurrent programming), 예외 처리(exception handling) 등과 같은 대형의 실시간 처리 시스템을 개발하는데 필수적인 특성을 가진 표준화된 고급 프로그램밍 언어이다[3]. 이러한 CHILL로써 큰 프로그램을 개발하기 위해서는 하나의 프로그램을 여러 개의 화일로 나누어 개발하는 것이 바람직하며, 큰 프로그램을 여러 화일로 나누어 프로그래밍하고 이를 컴파일하기 위한 방법으로서 분리 컴파일을 사용한다[4, 5, 6, 7].

분리 컴파일의 과정은 크게 프로그램 명세(specification)의 자동 생성, 이름 바인딩, 그리고 컴파일의 세 단계로 나누어 진다[4]. 프로그램 명세 생성 단계에서는 화일간에 공유하는 정보를 저장한 명세화일을 생성한다. 이름 바인딩 단계에서는 앞 단계에서 생성된 명세 정보를 이용하여 전체 프로그램 수준에서의 공유된 정보의 일치성과 어의(semantics)를 분석하고 프로그램이 어떻게 수정되었는지를 분석한다. 컴파일 단계에서는 수정에 의한 재컴파일을 필요로 하는 화일을 컴파일하여 목적 화일을 생성하고 이들을 링크하여 하나의 실행모듈을 생성한다.

분리 컴파일은 화일들 사이에 교환되는 이름인 변수, 상수 및 자료형에 대한 바인딩(binding)과 일치성을 점검하여 항상 완전한 컴파일이 수행될 수 있게 하며, 컴파일 과정을 자동화하고 재컴파일을 최소화할 수 있도록 하여 준다. 또한 화일 단위로 컴파일할 경우에는 유닉스 시스템에서 제공하는 make 기능을 이용할 수 있어 컴파일 관리가 훨씬 용이해 진다.

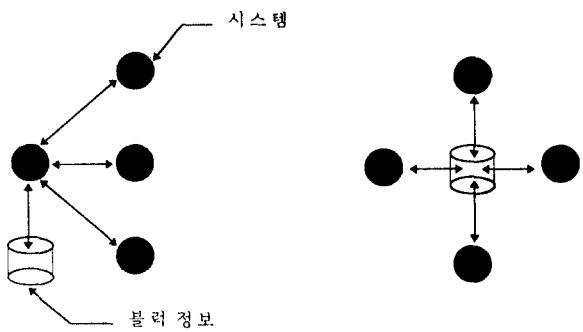
Ⅲ. 네트워크 화일 시스템상에서의 컴파일

3.1 시스템간의 블럭 할당

이동통신교환기 소프트웨어의 개발에는 네트워크 화일 시스템으로 구성된 다수의 유닉스 시스템이 사용되고 있으며 컴파일에 필요한 환경을 모든 시스템이 서로 공유하고 있다. 이와 같은 환경에서 각 시스템은 서로 독립적인 컴파일 수행이 가능하며, 모든 시스템에 컴파일 작업을 골고루 분산하여 수행시킴으로써 컴파일 수행 시간을 최소화할 수 있다. 그러나 이와 같은 방법이 효과적으로 동작하기 위해서는 모든 시스템에서 컴파일이 동시에 수행되고 거의 동시에 컴파일 작업이 완료될 수 있도록 컴파일 작업이 할당되어야 한다.

이동통신교환기 소프트웨어의 컴파일은 블럭 단위로 수행되기 때문에 각 시스템에 컴파일 작업을 블럭 단위로 할당한다. 블럭을 할당하는 방법에는 컴파일 수행전에 미리 고정적으로 하는 방법과 컴파일 수행중에 동적으로 하는 방법이 있다. 컴파일 수행전에 블럭을 고정적으로 할당하는 방법은 동적으로 할당하는 방법보다 제어가 훨씬 간단하여 구현이 용이하다. 그러나 블럭의 크기에 따른 블럭별 컴파일 수행 시간이 서로 다르고 시스템별 성능이 다를뿐 만 아니라, 시각에 따라 시스템의 부하가 계속 변하기 때문에 시스템의 자원을 최대한으로 이용할 수 없으며, 시스템이 크래쉬되는 경우에 쉽게 대처할 수 없는 단점이 있다.

블럭을 동적으로 시스템에 할당하는 방법에는 그림 3(a)와 같이 어느 한 시스템이 컴파일할 블럭에 관한 정보를 관리하면서 다른 시스템에 블럭을 할당하는 중앙 관리식과, 그림 3(b)와 같이 모든 시스템에서 블럭에 관한 정보를 공유하면서 컴파일할 블럭을 각자 스스로 할당하는 개별적 접근식의 두 가지로 나눌 수 있다. 중앙 관리식은 시스템간의 메시지 전송을



a. 중앙 관리식
a. Centrally managing method

b. 개별적 접근식
b. Individually accessing method

그림 3. 블럭의 동적 할당 방법
Fig. 3 Dynamic allocation methods of blocks

위한 통신이 추가로 요구되나 병행 접근을 위한 제어가 필요 없으며 다른 시스템의 컴파일 진행 상황에 관한 정보를 쉽게 관리할 수 있다. 또한 시스템간의 메시지로 블럭명외에 시스템의 동작을 제어하는 여러 메시지를 전달할 수 있기 때문에 좀 더 다양한 기능을 수행할 수 있다. 반면에 개별적 접근식은 중앙 관리식에 비해 시스템간의 제어가 훨씬 단순하여 구현이 용이하다.

중앙 관리식은 블럭을 할당하는 기능을 하는 시스템이 크래쉬될 경우 다른 모든 시스템에서도 컴파일 수행이 불가능해진다. 그러나 모든 시스템이 네트워크 화일 시스템으로 구성되어 있기 때문에 공유된 화일 시스템을 실제 소유하고 있는 시스템이 크래쉬된 경우에는 개별적 접근식에서도 더 이상 컴파일 수행이 어렵게 된다. 따라서 중앙 관리식의 경우 블럭을 할당하는 기능을 하는 시스템을 공유된 화일 시스템을 실제 소유하고 있는 시스템으로 할 경우 시스템 크래쉬에 의한 피해를 최소화할 수 있다.

3.2 시스템간 메시지 전달

각 프로세스들은 메시지 전달에 의해 서로 필요한 정보를 주고 받는다. 유닉스 시스템상에서 프로세스간에 서로 통신하는 방법에는 여러 가지가 있으나, 각 프로세스들이 서로 다른 시스템에 위치하기 때문에 시스템간에 통신을 가능하게 해주는 버클리 소켓(Berkeley Socket)을 사용하였다. 소켓으로 통신하는 프로세스는 서버(server) 프로세스와 클라이언트(client) 프로세스로 나눌 수 있으며, 클라이언트 프로세스는 서버 프로세스에서 소켓을 먼저 생성해야만 소켓에 접근할 수 있다. 서버 프로세스와 클라이언트 프로세스간에 일단 연결이 성립되면 두 프로세스는 소켓을 통하여 필요한 정보를 보내거나 받음으로써 서로간의 동작을 제어한다.

소켓을 이용한 프로세스간 통신에는 연결성(connection-oriented) 통신과 비연결성(connectionless) 통신으로 나눌 수 있다. 연결성 통신은 두 프로세스간에 논리적인 연결이 먼저 성립되어야만 통신이 가능한 경우이며, 비연결성 통신은 이와 같은 연결의 성립을 필요로 하지 않는 경우를 말한다. 연결성 통신의 대표적인 예로는 TCP(Transmission Control Protocol)를 들 수 있으며 비연결성 통신으로는 UDP(User Datagram Protocol)를 들 수 있다. 그림 4는 소켓을 이용한 서버 프로세스와 클라이언트 프로세스간의 연결성 통신의 흐름을 나타낸다[8, 9, 10].

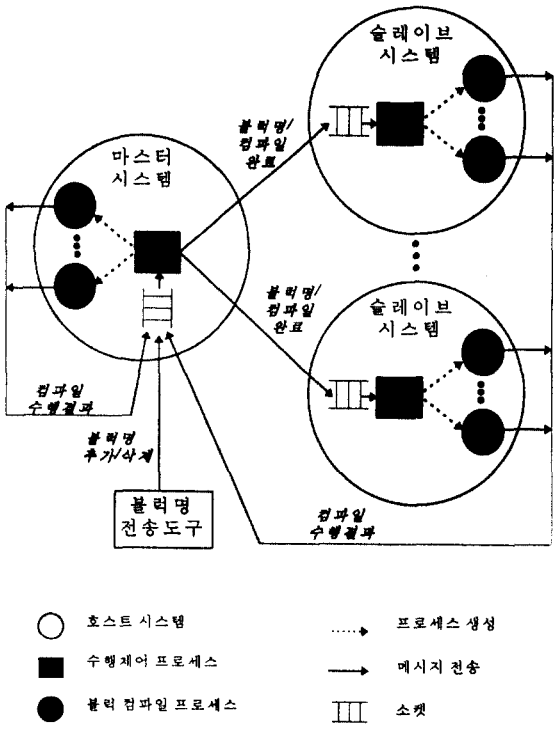


그림 4. 소켓을 이용한 연결형 통신의 동작 흐름
Fig. 4 The flow of the connection-oriented communication using socket

3.3 여러 블럭의 동시 컴파일

블럭을 컴파일하는 데 소요되는 시간은 주로 컴파일을 수행하는 호스트 시스템의 성능에 좌우된다. 호스트 시스템의 성능을 결정하는 가장 큰 요인은 프로세서(processor)의 속도 및 프로세서의 수이며 시스템에 따라 다양한 값을 가진다. 따라서 컴파일에 소요되는 시간을 최소화하기 위해서는 주어진 시스템의 계산 능력(computing capacity)을 얼마나 최대한으로 이용하느냐에 달려 있으며, 시스템 사용자의 입장에서는 동시에 컴파일되는 블럭의 수를 최적화하는 데에 그 목표를 둘 수 있다. 시스템의 성능과 더불어 컴파일에 소요되는 시간을 결정하는 큰 요인으로는 시스템의 부하를 들 수 있다. 시스템의 부하는 프로세

스의 수에 의해 결정되며 시각에 따라 동적으로 변하는 값을 가진다.

N 개의 유닉스 시스템이 컴파일에 필요한 모든 작업 영역을 서로 공유하고 있는 네트워크 화일 시스템으로 구성되어 있고, 각 시스템이 적어도 하나씩의 블럭을 컴파일한다고 가정할 때, 동시에 컴파일이 가능한 블럭의 최소수는 N이 된다. 그리고 각각의 시스템 H_1, H_2, ., H_n이 가지고 있는 프로세서의 수(Number of Processor)를 $NOP(H_1)$, $NOP(H_2)$, ..., $NOP(H_n)$라고 하고 각 시스템에서는 최소한 프로세서 수만큼의 블럭을 동시에 컴파일한다고 가정하면, 동시에 컴파일이 가능한 블럭의 최소 수는 $NOP(H_1)+NOP(H_2)+$, ..., $+NOP(H_n)$가 된다. 여기서 동시에 컴파일이 가능하다는 것은 시스템의 입장에서가 아니라 시스템 사용자의 입장에서 본 개념으로서, 각 시스템은 시분할(time sharing) 방식에 의해 여러 개의 프로세스를 병행적으로 수행하므로 시스템 또는 운영체제의 입장에서 본다면 어느 순간에 컴파일되고 있는 블럭의 최소수는 N보다 작은 수가 된다.

동시에 컴파일이 가능한 블럭의 최대 수는 그 시스템에서 생성할 수 있는 프로세스의 최대 수에 의해서 결정된다. 하지만 프로세스의 수가 많아 질수록 각 프로세스에 할당되는 시간은 감소하고 운영체제가 프로세스를 제어하는 데 소요되는 시간은 많아지므로 전체적인 컴파일 시간은 오히려 증가하게 된다. 또한 컴파일을 수행하는 호스트 시스템에는 컴파일 수행 관리자 소유의 프로세스뿐 만 아니라 다른 일반 사용자의 프로세스도 함께 존재하므로 동시에 컴파일이 가능한 블럭의 최대수는 시각에 따라 변하게 된다.

시스템상의 이와 같은 여건을 모두 고려하여 어느 순간에 컴파일 수행시간을 최소화하기위한 동시에 컴파일할 수 있는 최적의 블럭수를 결정하기는 무척 어려운 일이다. 그러나 주어진 시스템의 계산 자원을 최대한으로 활용하기 위해서는 프로세서당 최소한 하나 이상의 블럭이 컴파일되도록 할당하는 것이 바람직하다. 따라서 각 시스템에 할당되는 블럭의 수는 그 시스템의 프로세서의 수에 의존한다.

3.4 블럭 할당 알고리즘

호스트 시스템 H_i가 $NOP(H_i)$ 개의 프로세서로 구성되어 있으며, 각 시스템은 최대 $NOP(H_i)$ 개의 블럭을 동시에 컴파일한다고 가정할 때 각 시스템으로 블럭을 할당하는 알고리즘은 다음과 같다. 블럭의 할당은 초기 할당과 동적 할당의 두 단계로 구성된다. 초기 할당에서는 각 시스템으로 $NOP(H_i)$ 만큼의 블럭이 차례로 할당된다. 동적 할당은 한 블럭의 컴파일이 완료될 때마다 이루어 지며, 컴파일을 완료한 시스템으로 새로운 블럭이 할당한다.

```
〈Algorithm : BlockAllocation〉
Begin
  SYS:=(H1, H2, ..., HN);   /* Host systems */
  BQ:=(B1, B2, ..., BM);    /* Queue of Blocks to be compiled */

  For each system Hi
    ABN(Hi):=0;             /* Number of blocks currently allocated
                               to the system Hi */

  For each system Hi        /* Initial allocation of blocks */
  Begin
    if(BQ ≠ EMPTY AND ABN(Hi)< NOP(Hi))
    Begin
      Allocate a block in the BQ to the system Hi;
      Pop up the allocated block from the BQ;
      ABN(Hi):=ABN(Hi)+1;
    End;
  End;

  While(BQ ≠ EMPTY)         /* Run-time allocation of blocks */
  Begin
    if(Hi have completed the compilation of an allocated block)
    Begin
      ABN(Hi):=ABN(Hi)-1;
      Allocate a block in the BQ to the system Hi;
      Pop up the allocated block from the BQ;
      ABN(Hi):=ABN(Hi)+1;
    End;
  End;

End.
```

각 시스템으로 할당된 블럭은 다시 그 시스템내의 프로세서로의 할당이 이루어진다. 프로세서에서 수행되는 작업의 단위는 프로세스로서 각 블럭을 프로

세서에 할당한다는 것은 블럭의 컴파일을 수행하는 프로세스를 그 프로세서에서 수행되게 하는 것이다. 그러나 프로세스를 프로세서에 할당하는 일은 호스트 시스템의 운영체제가 담당하고 있기 때문에 응용 프로그램 단계에서 이를 제어하기는 어렵다. 따라서 블럭의 프로세서로의 할당은 운영체제에 의해 이루어지도록 한다.

Ⅳ. 컴파일 관리 시스템의 구성 및 기능

4.1 컴파일 관리 시스템의 구성

본 논문에서 제안된 컴파일 관리 시스템은 중앙 관리식 블럭 할당 방법을 사용하여 구현하였으며 시스템의 전체적인 구성은 그림 5와 같다. 네트워크 화일 시스템으로 구성된 다수의 유닉스 시스템중에서 하나의 시스템이 블럭을 할당하는 기능을 하는 마스터(master) 시스템이 되며 나머지 시스템은 슬레이브(slave) 시스템이 된다. 각 시스템은 컴파일 작업의 수행을 제어하는 하나의 수행제어 프로세스(Execution-control process)와 이 수행제어 프로세스에 의해 생성되는 하나 이상의 블럭컴파일 프로세스(Block-compile process)로 구성된다.

마스터 시스템에서의 수행제어 프로세스는 슬레이브 시스템으로 블럭을 할당하는 기능과 블럭컴파일 프로세스를 생성하는 기능 및 컴파일 수행 결과에 대한 정보를 관리하는 기능을 수행하며, 슬레이브 시스템에서의 수행제어 프로세스는 마스터 시스템의 수행제어 프로세스로 부터 블럭명을 할당받고, 이에 해당하는 블럭컴파일 프로세스를 생성하는 기능을 수행한다. 블럭컴파일 프로세스는 하나의 블럭에 대한 컴파일을 수행하고 컴파일의 성공 여부를 마스터 시스템의 수행제어 프로세스에게 전송하는 기능을 수행한다.

각 시스템내 프로세스들은 소켓을 이용한 메시지 전송에 의해 필요한 정보를 주고 받는다. 마스터 시스템에서의 수행제어 프로세스가 슬레이브 시스템의 수행제어 프로세스로 전송하는 메시지에는 '블럭명' 메시지와 '컴파일 완료' 메시지의 두 종류가 있으며, 블럭컴파일 프로세스가 마스터 시스템의 수행제어 프로세스에게 전송하는 메시지에는 '컴파일 수행 결과' 메시지가 있다. 그리고 컴파일 관리자가 마스터 시스템으로 전송하는 메시지에는 '블럭명 추가' 메시지와 '블럭명 삭제' 메시지의 두 종류가 있다. 컴파일 관리자는 마스터 시스템의 수행제어 프로세스에게 메시지를 보내기 위해서 별도의 메시지 전송 도구를 필요로 한다.

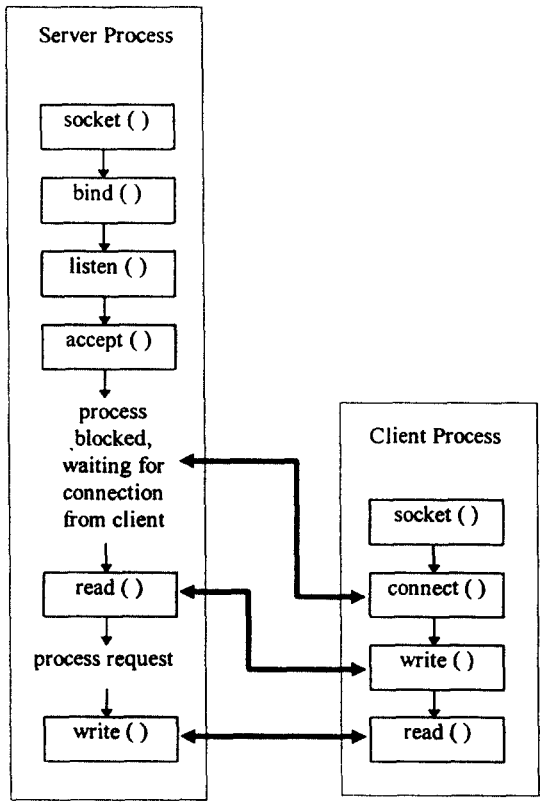


그림 5. 이동통신교환기 소프트웨어 컴파일 관리 시스템의 구성

Fig. 5 The structure of the Compilation Management System for the mobile switching software

4.2 컴파일 관리 시스템의 동작 흐름

마스터 시스템의 수행제어 프로세스는 컴파일을 수행할 시스템, 작업 디렉토리 및 블럭에 관한 정보를 저장한 화일을 읽어 들여 컴파일 수행 환경을 검사하고 초기화하는 작업을 수행한다. 먼저 자신의 시스템에서 수행될 일정 수의 블럭컴파일 프로세스를 생성하며 각 슬레이브 시스템으로 일정 수의 블럭명

메시지를 전송한다. 이 때 자신의 시스템에 생성하는 블럭컴파일 프로세스의 수 및 슬레이브 시스템으로 전송하는 블럭명 메시지의 수는 해당 시스템에서 동시에 컴파일할 블럭의 수에 따라 결정된다. 한 블럭에 대한 컴파일이 완료되어 블럭컴파일 프로세스로부터 컴파일 수행 결과 메시지가 전송되어 올 경우 이를 분석하여 시스템 로그화일에 기록하며, 현재까지 컴파일이 수행되지 않고 남아 있는 블럭의 수 및 메시지를 전송해온 시스템의 종류에 따라 블럭컴파일 프로세스를 생성하거나 블럭명 또는 컴파일 완료 메시지를 해당 시스템에 전송한다. 블럭명 추가 또는 삭제 메시지가 전송되어 온 경우는 해당 블럭명을 블럭명 리스트에 추가 또는 삭제하고 이를 시스템 로그화일에 기록한다. 마스터 시스템에서 수행제어 프로세스의 실행 흐름은 그림 6과 같다.

슬레이브 시스템에서의 수행제어 프로세스는 마스터 시스템의 수행제어 프로세스로부터 블럭명 메시지가 전송되어 올 경우 해당 블럭에 대한 블럭컴파일 프로세스를 생성하며, 컴파일 완료 메시지가 전송되어 올 경우에는 실행을 완료한다. 슬레이브 시스템에

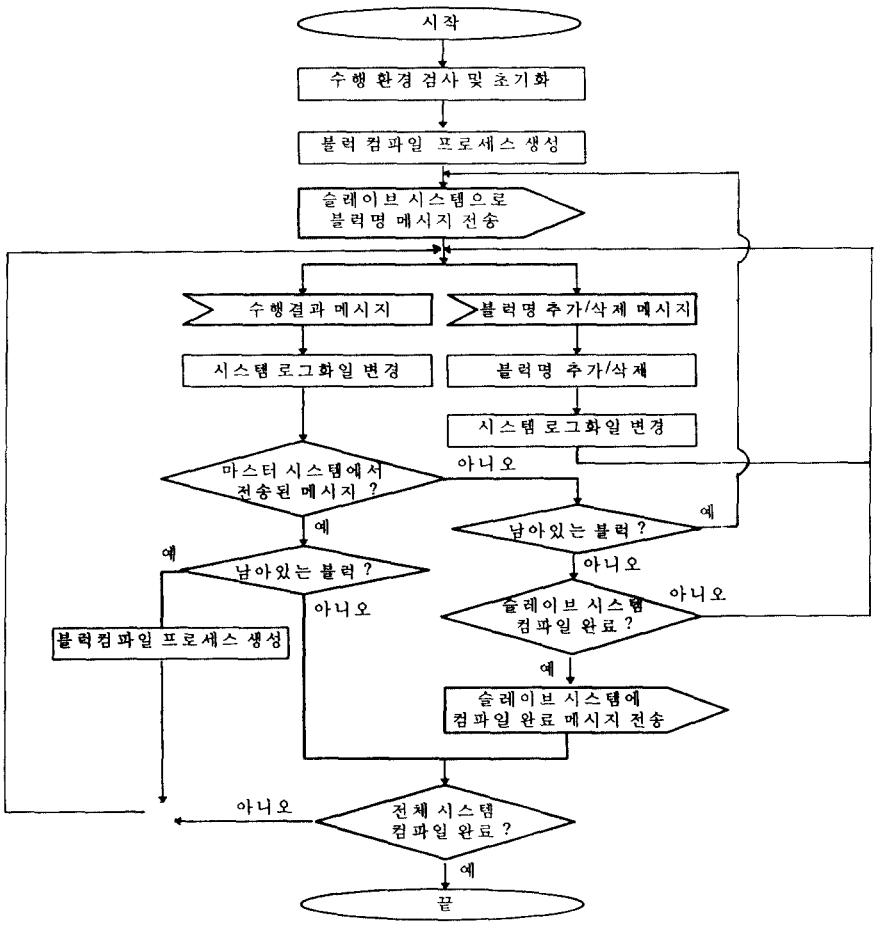


그림 6. 마스터 시스템에서 수행제어 프로세스의 실행 흐름
Fig. 6 Execution flow of the Execution-control process in master system

서 수행제어 프로세스의 실행 흐름은 그림 7과 같다.

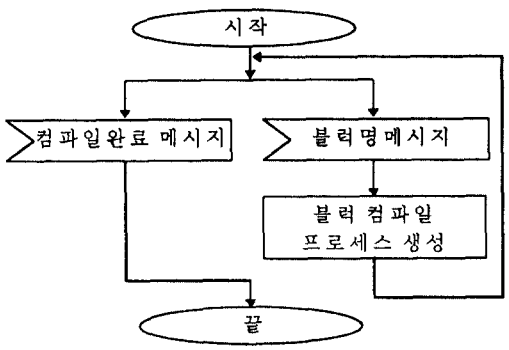


그림 7. 슬레이브 시스템에서 수행제어 프로세스의 실행 흐름

Fig. 7 Execution flow of the Execution-control process in slave system

수행제어 프로세스에 의해 생성된 블럭컴파일 프로세스는 한 블럭에 대한 분리 컴파일을 수행하며, 컴파일 오류 발생시 이를 블럭 담당자에게 통보해 주며, 컴파일 수행 결과 메시지를 마스터 시스템의 수행제어 프로세스에게 전송한다. 블럭컴파일 프로세스의 실행 흐름은 그림 8과 같다.

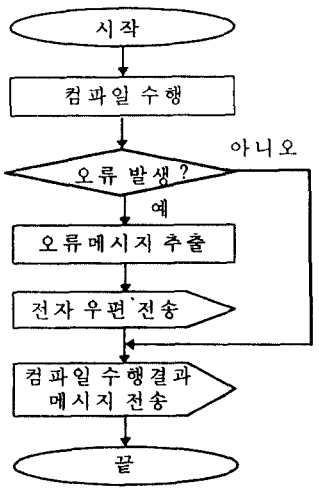


그림 8. 블럭 컴파일 프로세스의 실행 흐름

Fig. 8 Execution flow of the block complie process

4.3 컴파일 오류 메시지의 전송

블럭컴파일 프로세스는 분리 컴파일의 각 단계를 수행하면서 컴파일러로 부터 출력되는 메시지를 임시화일에 기록한다. 컴파일 수행중에 오류가 발생한 경우, 이 임시화일을 분석하여 오류 메시지를 추출하고 이를 블럭별 오류 메시지 화일에 기록한다. 오류 메시지는 오류 메시지에 사용되는 키워드(key word)를 검색함으로써 추출할 수 있으며, 이에 해당하는 키워드에는 'Error', 'Can't', 'not' 등이 있다.

오류가 발생된 블럭의 담당자에게 전송될 전자 우편(electronic mail)에는 컴파일중인 이동통신교환기 소프트웨어의 버젼, 전자 우편 전송 시간, 오류 발생 블럭명, 재등록 시간 등을 포함할 수 있으며, 그 외 컴파일 관리자가 블럭 담당자에게 알리고자 하는 메시지를 추가로 실을 수 있다. 이와 같은 메시지는 전자 우편 헤더(header) 화일로 구성한 후, 전자 우편을 보낼 때에 이 헤더 화일과 오류 메시지를 결합하여 보낸다. 시간, 블럭명, 재등록 기간 등은 오류가 발생한 블럭과 컴파일을 수행하는 시간에 따라 달라지므로 헤더 화일에 전자 우편 전송 시간, 블럭명, 재등록하는 기간등이 위치할 곳에 이를 나타내는 지시어(directive)를 적고, 전자 우편을 보낼 때에 헤더 화일의 지시어를 이에 해당하는 실제의 값으로 대체하는 방법을 사용한다.

전자 우편을 전송할 주소는 형상화일(configuration file)을 참조하여 얻을 수 있으며, 형상화일에는 블럭 정보화일과 블럭 담당자 정보화일이 있다. 블럭 정보화일에는 블럭의 이름, 블럭의 담당자 및 기타 블럭에 관한 정보를 가지고 있으며, 블럭 담당자 정보화일에는 블럭 담담자명, 전자우편 주소, 전화번호, 소속 연구실명 등이 있다.

4.4 컴파일 우선 순위

블럭은 컴파일 우선순위에 의해 컴파일 수행 순서를 서로 다르게 할 수 있다. 각 블럭에 컴파일 우선 순위를 할당하는 문제는 프로세서에 프로세스를 어떤 순서로 할당하느냐 하는 것과 유사하다고 볼 수 있으며, 블럭의 컴파일 우선 순위는 블럭내 전체 화일의 수, 새로 등록된 화일의 수, 블럭의 수행 기능에 따라 결정할 수 있다.

하나의 블럭은 여러 개의 화일로 구성되어 있고, 각

화일은 서로 크기가 다르기 때문에 블럭당 컴파일 시간은 서로 다르다. 따라서 이와 같은 블럭의 컴파일 수행 시간을 정확히 예측하기는 어렵지만 블럭간의 상대적인 컴파일 수행시간은 각 블럭을 구성하고 있는 화일의 수로 예측할 수 있다. 즉 블럭내 화일의 수가 많은 블럭이 적은 블럭보다 컴파일 수행 시간이 많이 걸린다고 할 수 있다. 이와 같은 방법에 의해 블럭간에 상대적인 컴파일 수행시간을 계산한다고 할 때, 일정한 시간내에 컴파일되는 블럭의 수를 최대화하기 위해서는 컴파일 수행 시간이 적게 걸리는 블럭에 우선 순위를 두면 되고, 모든 시스템에서 거의 동시에 컴파일이 완료되도록 하려면 컴파일 수행 시간이 많이 걸리는 블럭에 우선 순위를 두면 된다.

블럭내 어느 소스 화일이 수정되었거나, 또는 이 블럭내의 어느 소스 화일이 참조하는 공통자료 화일의 내용이 수정되었을 때 이 블럭은 새로 컴파일을 해야 한다. 새로 컴파일하는 블럭은 당연히 오류가 발생할 가능성이 있으며, 그 수정된 내용이 많을수록 그 가능성이 크다고 볼 수 있다. 따라서 이와 같은 블럭은 다른 블럭보다 먼저 컴파일하여 오류가 발생한 경우 신속한 재등록이 이루어 지도록 한다. 그러나 공통자료 화일을 수정했을 경우 이 화일의 내용을 실제로 참조하는 블럭 또는 소스 화일을 일일이 찾는 데는 많은 시간이 필요하므로 블럭내에 새로 등록된 소스 화일의 수만을 가지고 블럭의 컴파일 우선 순위를 간단히 결정할 수 있다. 즉 새로 등록된 소스 화일의 수가 많은 블럭일수록 높은 컴파일 우선 순위를 갖는다.

각 블럭은 교환기의 가장 기본적인 기능을 수행하는 블럭과 부수적인 기능을 수행하는 블럭으로 나눌 수 있다. 부수적인 기능은 기본적인 기능이 정상적으로 수행되고 있을 때에만 동작이 가능하다. 예를 들어 호처리(call processing) 관련 블럭은 교환기의 가장 기본적인 기능을 수행하는 블럭으로서 그 어떤 블럭들보다 먼저 컴파일이 수행되어 시험이 이루어 져야 한다. 따라서 이와 같이 교환기에서 가장 기본적인 기능을 수행하는 블럭을 먼저 컴파일함으로써 전체적인 개발 일정을 앞당길 수 있다.

이상과 같은 세가지 컴파일 우선 순위 기준에 의해 블럭별로 컴파일 순서를 다르게 할당함으로써 컴파일 수행 시간 또는 시스템 개발 일정을 단축할 수 있다.

4.5 블럭명의 추가와 삭제

화일의 등록은 화일 등록 도구에 의해서 이루어 지며, 화일 등록자와 등록하고자 하는 화일이 모두 허가되었을 때 실제 등록이 가능하다. 마스터 시스템의 수행제어 프로세스는 컴파일 중에 오류가 발생한 블럭에 대해 재등록이 가능하도록 화일 등록 도구를 호출하여 그 블럭에 대한 등록을 허가해 준다. 블럭에 대한 재등록이 이루어 지는 시점에서 다른 블럭에 대한 컴파일이 모두 완료되었다면 컴파일 작업을 다시 시작하면 되지만, 다른 블럭에 대한 컴파일이 계속 진행중에 있다면 새로 등록된 블럭 이름을 마스터 시스템의 수행제어 프로세스에게 알려 주어 재컴파일 하도록 하여야 한다.

컴파일 관리자 또는 화일 등록 도구는 소켓을 사용하여 마스터 시스템의 수행제어 프로세스에게 새로 등록된 블럭의 이름을 전송하며, 이를 위해서는 블럭명 메시지를 전송하는 별도의 도구가 필요하다. 이 도구는 컴파일 관리자 또는 화일 등록 도구로부터 필요한 인자들을 넘겨받아 이를 블럭명 추가 또는 삭제 메시지의 형태로 마스터 시스템의 수행제어 프로세스에게 전송한다. 블럭명의 추가와 삭제 메시지 전송에 사용되는 소켓은 블럭컴파일 프로세스가 마스터 시스템의 수행제어 프로세스에게 수행 결과 메시지를 전송하는 데 사용되는 소켓을 함께 사용한다. 마스터 시스템의 수행제어 프로세스는 이 소켓으로 새로운 메시지가 전송되었는 지를 검사하고 해당되는 메시지에 대한 동작을 수행한다.

4.6 컴파일 수행 상태의 기록

컴파일 관리자는 현재 컴파일이 수행되고 있는 블럭이 무엇이며, 현재까지 컴파일 완료된 블럭의 수가 얼마나 되는지 수시로 확인할 필요가 있다. 그러나 컴파일이 수행되는 블럭의 수가 백 여 개에 달하기 때문에 이를 컴파일 관리자가 일일이 기록하고 관리하기는 무척 어려운 일이다. 이를 해결하기 위해서 마스터 시스템의 수행제어 프로세스는 블럭컴파일 프로세스로 부터 전송되어온 컴파일 수행 결과 메시지를 분석하고, 이를 시스템 로그 화일에 기록한다. 시스템 로그 화일에 포함되는 내용에는 다음과 같은 것이 있다.

- 컴파일 작업을 시작한 시각과 끝낸 시각
- 컴파일이 수행된 블럭의 블럭명, 컴파일이 수행된 시스템명, 컴파일이 시작된 시각과 끝난 시각, 컴파일 완료 상태
- 전체 블럭의 수
- 현재까지 컴파일이 완료된 블럭의 수
- 컴파일 수행중에 새로 추가 또는 삭제된 블럭명과 그 수행 시각

V. 성능 평가

블럭의 컴파일 소요 시간에 영향을 주는 요소에는 시스템의 성능과 부하를 들 수 있다. 그러나 시스템간에 성능을 정확하게 비교하기는 어려우며, 또한 시스템의 부하를 일정하게 함으로서 동일한 시험 환경을 계속 유지하기는 더욱 어렵다. 그러나 여기서는 시스템의 성능 및 부하를 수치로 일정하게 나타낼 수 있다는 것을 가정하여 몇 가지 블럭 할당 알고리즘에 대한 성능 평가를 수행하였다. 다음은 성능 평가에 사용될 표기들이다.

$P(H_i)$, $1 \le i \le N$: 호스트 시스템 H_i의 성능

$T(H_i, B_j)$, $1 \le i \le N$, $1 \le j \le M$: 시스템 H_i에서 블럭 B_j의 컴파일에 소요되는 시간

$T(B_1, B_2, ..., B_M)$: 블럭 $B_1, B_2, ..., B_M$의 컴파일에 소요되는 시간

$NOP(H_i)$: 시스템 H_i의 프로세서 수

먼저, N 개의 호스트 시스템의 성능 및 부하가 동일하고 각각의 블럭의 컴파일에 소요되는 시간이 T_c로서 모두 동일하다고 가정할 때, M 개의 블럭을 하나의 시스템에 모두 할당하여 컴파일하는 경우의 컴파일 소요 시간은 식 (1)과 같이 계산되며, N 개의 시스템에 균등하게 할당하여 컴파일 하는 경우의 컴파일 소요 시간은 식 (2)와 같이 계산된다.

$$T(B_1, B_2, ..., B_M) = T(H_1, B_1) + T(H_1, B_2) + ... + T(H_1, B_M) = M * T_c \quad (1)$$

$$T(B_1, B_2, ..., B_M) = T(H_1, B_1) + T(H_1, B_2) + ... + T(H_1, B_{M/N}) = M/N * T_c \quad (2)$$

N 개의 호스트 시스템의 성능이 식 (3)과 같은 관계를 가지고 각각의 블럭의 컴파일에 소요되는 시간이 식 (4)와 같이 시스템의 성능에 비례한다고 가정할 때, M 개의 블럭을 N 개의 시스템에 균등하게 정적으로 할당하는 경우의 컴파일 소요 시간은 식 (5)와 같이 계산된다. 여기서 T_d는 H_1 시스템에서 하나의 블럭을 컴파일하는 데 소요되는 시간이다.

$$P(H_2) = 2 * P(H_1), P(H_3) = 3 * P(H_1) ..., P(H_N) = N * P(H_1) \quad (3)$$

$$T(H_1, B_1) = 2 * T(H_2, B_1) = , ..., = N * T(H_N, B_1) \quad (4)$$

$$\begin{aligned} T(B_1, B_2, ..., B_M) &= MAX\{T(H_1, B_1) + T(H_1, B_2) + ... \\ &\quad + T(H_1, B_{M/N}), T(H_2, B_{M/N+1}) \\ &\quad + T(H_2, B_{M/N+2}) + ... + T(H_2, B_{2*M/N}), \\ &\quad ..., T(H_N, B_{M-M/N+1}) \\ &\quad + T(H_N, B_{M-M/N+2}) + ... + T(H_N, B_M)\} \\ &= MAX\{M/N * T_d, M/N * T_d/2, ..., \\ &\quad M/N * T_d/N\} \\ &= M/N * T_d \end{aligned} \quad (5)$$

시스템 H_1이 하나의 블럭을 컴파일하는 동안 H_i는 i 개의 블럭을 컴파일할 수 있는 성능이므로 M 개의 블럭을 N 개의 시스템에 동적으로 할당하는 경우, 전체 블럭의 컴파일에 소요되는 시간은 H_1과 동일한 성능을 가지는 $(1 + 2 + ... + N)$ 개의 시스템에 블럭을 정적으로 균등하게 할당하는 경우로 생각할 수 있으며 식 (6)과 같은 결과를 얻을 수 있다.

$$T(B_1, B_2, ..., B_M) = (2 * M)/(N * (N + 1)) * T_d \quad (6)$$

시스템의 성능은 프로세서의 수에 비례한다. 따라서, 각 시스템에서의 프로세서의 수가 식 (7)과 같은 관계를 가지고 각각의 블럭의 컴파일에 소요되는 시간이 식 (4)와 같이 시스템의 성능에 비례한다고 가정할 때, M 개의 블럭을 N 개의 시스템에 균등하게 정적으로 할당하는 경우와 동적으로 할당하는 경우의 컴파일 소요 시간은 식 (5), (6)과 동일한 것으로 생각할 수 있다.

$NOP(H_2) = 2 * NOP(H_1)$, $NOP(H_3) = 3 * NOP(H_1)$, ...,
$NOP(H_N) = N * NOP(H_1)$ (7)

따라서, 식 (8)에 의해 블럭을 동적으로 할당하는 경우는 정적으로 균등하게 할당하는 경우보다 항상 컴파일에 소요되는 시간이 적음을 알 수 있다.

$$(M/N * T_d)/((2 * M)/(N * (N+1)) * T_d) = (N+1)/2 \geq 1 \quad (8)$$

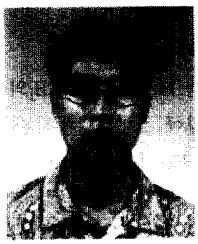
Ⅵ. 결 론

네트워크 화일 시스템상에서 이동통신교환기 소프트웨어의 컴파일 작업을 관리해주는 시스템의 구성과 기능에 대해 기술하였다. 제안된 시스템은 네트워크 화일 시스템상에서의 분산적인 컴파일, 한 시스템에서 여러 블럭의 동시 컴파일, 컴파일 오류 메시지 추출 및 전자 우편전송, 컴파일 상태 기록 등의 기능을 가지며, 시스템 계산 자원의 효율적인 이용, 컴파일 관리자와 블럭 담당자간의 효과적인 인터페이스, 컴파일 진행 상황의 용이한 파악등의 장점을 가지고 있다. 나아가서 이동통신교환기 소프트웨어의 컴파일 및 종합 작업에 소요되는 시간을 크게 단축함으로써 전체 시스템 개발의 일정을 앞당길 수 있었다.

앞으로의 연구 내용으로는 시스템의 부하 계산에 의한 동시에 컴파일할 블럭의 수 결정 및 네트워크 화일 시스템으로 구성되어 있지 않은 다른 시스템의 계산 자원도 함께 이용함으로써 컴파일에 소요되는 시간을 최소화하는 것이다.

참 고 문 헌

1. 김대식, 김성희, 안지환, 이충근, "이동통신교환기 소프트웨어", 한국통신학회지, 제 11 권 3 호, pp. 34-49, 1994.
2. 박광로, 이남준, 채용우, 이영호, "이동통신교환기에서의 소스 및 공통화일 관리 시스템의 구현", 대한전자공학회 하계종합학술대회 논문집, 제 17 권 1 호, pp. 169-172, 1994.
3. CCITT, "CCITT High Level Language(CHILL)", Recommendation Z.200, 1992.
4. 松尾勇二, 丸山勝己, 교환용 프로그래밍 언어 CHILL, 도서출판 기다리, 1988.
5. 최완, 송영기, 김영시, "CCITT 언어들에 기반을 둔 소프트웨어 개발 환경", JCCI'92 학술논문집, 제 2 권, pp. 180-184, 1992.
6. 최완, 최고봉, 이충근, "통신 시스템 개발을 위한 소프트웨어 개발 환경", 소프트웨어 공학회지, 제 6 권 2 호, pp. 54-69, 1993.
7. 김상국, 이동길, 최완, 이병선, "통신 시스템용 대용량 소프트웨어 개발을 위한 CHILL 프로그래밍 환경", 한국정보과학회 '95 가을 학술발표논문집, 제 22 권 2 호, pp. 1267-1269, 1995.
8. W. R. Stevens, Advanced Programming in the UNIX Environment, Addison-Wesley Publishing Company, 1992.
9. J. Bloomer, Power Programming with RPC, O'Reilly & Associates, Inc, 1992.
10. W. R. Stevens, Unix Network Programming, Prentice-Hall International, 1991.



신 재 욱(Jae Wook Shin) 정회원

1992년 2월 : 경북대학교 컴퓨터과학과 졸업(학사)
1994년 2월 : 경북대학교 대학원 컴퓨터과학과 졸업(이학석사)
1994년 3월~현재 : 한국전자통신연구원, 이동교환기 기연구실

박 광 로(Kwang Roh Park) 정회원

1982년 2월 : 경북대학교 공과대학 전자공학과(학사)
1985년 2월 : 경북대학교 대학원(석사)
1984년 3월~현재 : 한국전자통신연구원, 선임연구원 초고속서비스연구실