실시간 분산처리를 제공하는 CHILL 실행시간 지원 시스템의 설계 및 구현

백 의 현[†] · 장 종 현[††] · 이 동 길[†††]

요 약

본 논문은 ATM 교환 시스템과 같은 실시간 분산 소프트웨어를 범용 컴퓨터 시스템에서 실행할 수 있는 환경을 제공하는 CHILL 실행시간 지원 시스템(CHILL Run-time System)의 설계 및 구현 기술에 관하여 기술한다. 이를 위하여 실시간 분산 CHILL 프로그램을 위한 실행 모델을 제시하고 해당 실행 모델을 갖는 소프트웨어를 범용 컴퓨터에서 실행하는 CRS를 개발하였다. CRS는 목적 시스템 운영체제와 동일한 실행환경을 호스트 컴퓨터에서 제공함으로 프로그래머는 목적 시스템이 없이도 소프트웨어 개발이 가능하다. 또한 다수의 개발자가 동시에 소프트웨어를 실행할 수 있는 환경을 제공하여 프로그램 생산성 향상에 도움이 된다. 이 실행환경은 SROS (Scalable Real-time Operating System)를 위하여 개발되었지만 다른 내장형 운영체제의 실행환경으로 확장이 용이하다.

A Design and Implementation of CHILL Run-time System for Distributed Real-time Processing

Eui-Hyun Paik[†] · Jong-Hyun Chang[††] · Dong-Gill Lee[†††]

ABSTRACT

This paper describes the design and implementation of a CHILL Run-time System(CRS) that provides a running environments of real-time distributed software like as ATM switching software on host computers. To do this, we have proposed an execution model of real-time distributed CHILL programs and developed a CRS executing the software of the execution model on host computers. Because CRS provides transparent running environments of target operating system, programmers are able to develop target system software without target system. As it provides an environment for multiple users running programs concurrently, it guarantees the improvement of program productivity. Originally, CRS has developed as a simulator of SROS. But it can be easily modified for other operating systems.

1. 서 론

CHILL은 ITU-T에서 제안한 범용 프로그래밍 언어로 교환기 프로그램과 같은 통신용 소프트웨어의 개발에 주로 이용된다[1]. 이 언어는 기존의 Module-2, PASCAL, C 등의 프로그래밍 언어가 가지는 특성을 대부분 포함하고 있으며 모듈 프로그래밍, 병렬 처리, 강한 형 검사, 데이터 추상화 등의 특징을 가지고 있어 교환기용 소프트웨어의 개발에 아주 적합한 언어로 평가되고 있다.

차세대 정보 통신망의 핵심이 될 ATM (Asynchronous Transfer Mode) 교환기 시스템과 같은 소프트웨어들은 실시간 특성을 가지며 각 기능별로 그룹화 된

† 정 회 원 : 한국전자통신연구원 교환전송기술연구소 연구원
†† 준 회 원 : 한국전자통신연구원 교환전송기술연구소 연구원
††† 정 회 원 : 한국전자통신연구원 연구원
논문접수 : 2000년 3월 22일, 심사완료 : 2000년 9월 18일

여러 프로세서들로 구성된 분산 환경에서 실행된다[2]. 이러한 소프트웨어들은 계산의 논리적 결과가 정확해야 할 뿐 아니라 처리되는 결과가 주어진 시간 내에 도출되어야 하기 때문에 시스템 특성에 부합되는 특수한 운영체계를 요구한다. 그러나 상용 실시간 운영체제들은 내장형 시스템에 탑재되어 시스템이 목적하는 고유의 업무를 지원하는 반면에 범용 운영체제가 제공하는 것과 같은 다양한 프로그래밍 환경을 제공하지 않는다. 따라서 이와 같은 소프트웨어를 개발하기 위해서는 호스트 컴퓨터에서 소프트웨어를 개발하고 목적 시스템에 로딩한 후 호스트 컴퓨터에서 원격으로 시험하는 방법을 사용하여야 하는데 이러한 교차 개발 방법은 다음과 같은 몇 가지 제약점을 갖는다.

첫째로, 목적하는 내장형 시스템의 하드웨어 및 운영체제가 개발되기 전에는 프로그램을 실행할 수 없어 프로그램 개발 초기에는 효과적인 개발이 어렵다.

둘째로, 목적 시스템들은 제한된 자원을 가지므로 다수의 사용자가 한 시스템에서 동시에 프로그램을 시험할 수 없다. 따라서 효과적인 시험을 위해서는 다수의 테스트 베드가 요구된다.

세째로, 프로그램 실행 환경이 하드웨어 의존적이기 때문에 목적 시스템의 사양이 변경될 경우 하드웨어 및 운영체제가 완전히 구축되기 전까지는 해당 시스템에서 프로그램을 시험할 수 없다.

이러한 제약점을 해결하고자 범용 컴퓨터 시스템에서 교환기 소프트웨어를 실행 및 시험하려는 노력이 계속되어 왔다. 조철희[3]는 TDX-10 교환기(Time-Division Exchange-10)[4]의 운영체제 기능을 Ultrix와 SunOS에서 시뮬레이션할 수 있는 CHILL 실행시간 지원 시스템(CHO-CRS : CHILL Run-time System)을 개발하였다. 그러나 CHO-CRS는 단일 실행 블록에서 생성되는 CHILL 프로세스의 병렬처리 기능만을 제공하므로 프로그램의 블록 단위 시험에는 유용하지만 실시간, 다중, 분산 처리 기능을 제공하지 않아 기능 시험 및 통합 시험에는 적용할 수가 없었다.

박경숙[5]은 SROS(Scalable Real-time Operating System)[6]의 실행 환경을 UNIX[1]를 사용하는 범용 컴퓨터에서 시뮬레이션 할 수 있는 CHILL 시뮬레이션 환경(CSE : CHILL Simulation Environment)을 송후봉[7]은 SROS의 실행 환경을 UNIX를 사용하는 범용 컴퓨터에서 시뮬레이션 하고 재실행하는 기능을 제공하는 병행 프로그램 디버깅을 위한 결정적 재실행 시스템(CHILL Run-time System for Replay)을 각각 개발하였다. 이들은 모두 단일 프로세서에서 다중 실행 블록의 실행을 지원함으로 여러 실행 블록으로 구성되는 교환기 프로그램의 기능 시험에는 유용하나, 이들 역시 분산 처리 및 실시간 처리 기능을 지원하지 않아 이러한 특성을 포함한 프로그램 시험에는 적용할 수 없다.

Trond Borsting[8]은 단일 프로세서 상에서 CHIPSY_CHILL이 요구하는 병렬처리 기능을 지원하는 CHIPSY_CRS를 개발하였다. CHIPSY_CRS는 병렬 처리 기능은 제공하지만 교환기 소프트웨어가 요구하는 분산처리 기능은 지원하지 않으며 실시간 처리 기능도 제한적으로 제공한다. 또한 프로세스 간에 대량의 메시지 전송 시 메세지를 여러 개로 나누어 보내야 하는 제약을 갖고있다.

본 논문에서는 교환기 소프트웨어와 같은 실시간 분산 처리 소프트웨어를 범용 컴퓨터 상에서 효율적으로 개발하고 시험하기 위하여 실시간 처리 기능, 분산 프로세서 및 다중 실행 블록간의 동기화 및 메세지 전송 기능을 제공하는 CRS를 Kvatro Telecom사와 공동으로 개발하였다. CRS는 ATM 교환기의 운영체제로 사용되고 있는 SROS가 요구하는 실시간 분산 병렬처리 기능을 모두 수용하며 UNIX 상에서 실행되도록 구현되었으며 ETRI가 개발하고 있는 대형 ATM 교환기 개발에 적용되어 사용되고 있다.

본 논문의 구성은 다음과 같다. 2장에서는 분산 병렬 CHILL 프로그램을 위한 실행 모델을 정의하고, 3장에서는 CRS의 설계 및 구현에 대하여, 4장에서 CRS에 대한 시험 및 평가를 한 후 5장에서 결론을 내린다.

2. 분산 병렬 CHILL 프로그램을 위한 실행 모델 정의

본 논문에서는 분산 시스템 상에서 메세지 송/수신을 이용하여 프로세스간에 통신을 하는 메세지 기반 병행 프로그램을 기본 실행 모델로 하였다. <표 1>은 본 논문에 적용되는 병렬 CHILL 프로그램의 특성을 보여주고 있다. 병렬 수행의 기본 단위는 프로세스이

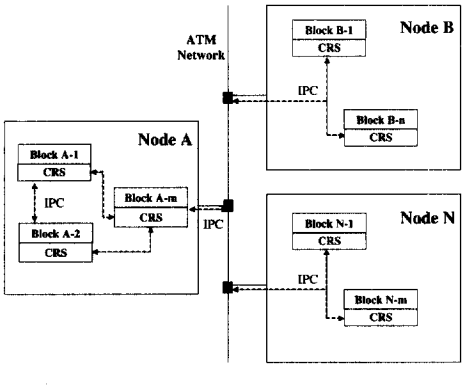
1) UNIX is a registered trademark of AT&T Bell Laboratories.

며 하나의 프로세스 내에는 하나의 쓰레드만 존재한다. 프로세스는 하나의 구조체로서 명시적인 표현 방법으로 기술되며 프로세스 내에 프로세스를 중첩하여 기술할 수 없다. 동일한 실행 블록에서 생성된 프로세스들은 전역 변수(global variables)를 공유할 수 있으나 다른 실행 블록에서 생성된 프로세스들 간에는 전역 변수를 공유할 수 없고 메세지를 이용한 통신만이 가능하다. 본 CHILL 프로그램 실행 모델에서는 전역 변수의 공유를 허용하였기 때문에 전역 변수에 대한 데이터 경쟁과 접근 순서 경쟁이 발생하며 메세지 경쟁에 의한 비결정성이 발생한다.

〈표 1〉 분산 병렬 CHILL 프로그램을 위한 실행 모델

		지원 방법	비 고
병행성 표현 방법		• 명시적 표현 방법	• 프로세서 선언
병행처리 단위		• 프로세스	• 프로세스내에는 하나의 쓰레드만 존재
정보교환 및 동기화 표현 방법	프로세서 내에서의 정보전달	• 동기적 메세지 전달 • 비동기적 메세지 전달	• 버퍼 송/수신 (버퍼크기 0) • 시그널, 버퍼 송/수신 (버퍼크기 1이상)
	프로세서 간의 정보전달	• 비동기적 메세지 전달	• 시그널
상호 배제적 실행방법		• 위험 지역 설정	• 위험지역 진입/탈출
실시간 처리		• 타임아웃 • 절대시간 획득 • 프로세스 실행 지연 • 주기적/반복적 문장 수행	• alarm 타이머 설정/해제 • 절대시간 획득 • time_sleep 설정 • 주기적/반복적 문장 실행 타이머 설정

실행 블록들은 (그림 1)과 같이 여러 개의 프로세서에 분산되어 실행되며 동일한 프로세서상의 실행 블록에서 생성된 프로세스 간에는 동기적/비동기적 통신

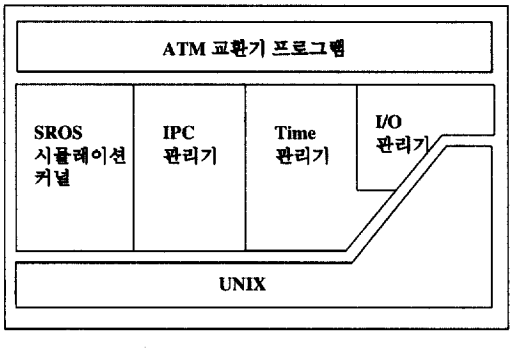


(그림 1) 분산 병렬 CHILL 프로그램의 실행 구조

방법을 모두 사용하고 분산 프로세서에서 실행되는 프로세스간에는 비동기적인 통신 방법인 시그널만 사용한다. 실시간 처리를 위하여 프로세스에 일정 시간의 경과를 알려주는 타임아웃 기능, 현재 시간 조회 기능, 일정 시간동안 프로세스 실행을 지연하는 기능 등을 사용하며 주기적인 문장의 실행이나 횟수 및 시간 간격이 지정된 반복된 실행 기능을 사용한다. 병행 CHILL 언어가 제공하는 병렬처리 문장은 컴파일러에 의해서 시스템 라이브러리 호출 형태의 코드로 생성되며, 프로세스가 실행 중 이 문장들을 만나면 라이브러리 형태로 구성된 CHILL 실행시간 지원 시스템이 이를 지원한다.

3. CRS의 설계 및 구현

CRS는 ATM 교환기와 같은 실시간, 분산 병렬처리 기능을 요구하는 내장형 소프트웨어를 범용 컴퓨터 상에서 투명하게 실행하기 위하여 개발되었다. CRS는 (그림 2)와 같이 네 부분으로 구성되어있다. SROS 시뮬레이션 커널은 병렬 CHILL 프로세스의 생성, 소멸, 실행, 동기화 및 상호배제 등을 담당하고, IPC 관리기는 분산 프로세서 상에서 실행되는 병렬 프로세스 간의 메세지 송수신 기능을 담당한다. Time 관리기는 병렬 프로세스가 UNIX 환경 위에서 실시간 처리가 가능하도록 절대, 상대 및 시간 처리 기능 제공하며 I/O 관리기는 프로세스들의 입출력을 담당한다. CRS는 라이브러리 형태로 구성되며 실행 블록마다 링크되어 수행된다.



(그림 2) CHILL 실행시간 지원 시스템의 구성도

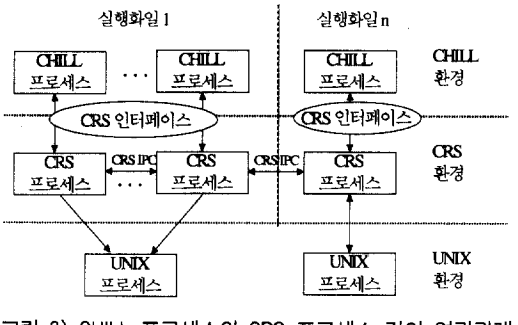
3.1 SROS 시뮬레이션 커널

CHILL 프로그램에서 병렬수행의 기본 단위는 프로

세스이며 하나의 프로세스 안에는 하나의 쓰래드만 존재한다. CHILL 프로세스들은 UNIX 프로세스처럼 다른 프로세스와는 독립적으로 수행되며 하나의 실행 블록에서 생성된 프로세스들은 UNIX 프로세스와는 달리 공유 변수들을 공유한다.

SROS 시뮬레이션 커널은 SROS 커널이 제공하는 기능을 UNIX 상에서 제공하며 실행화일 단위로 링크되어 동작한다. 시뮬레이션 커널은 CHILL 프로세스가 생성될 때 PCB 블록을 할당하고 프로세스 인스턴스 번호(PID), 프로세스 이름, 우선순위를 비롯한 프로세스의 상태 정보를 해당 PCB에 저장한다. SROS 시뮬레이션 커널은 PCB와 시그널/버퍼/리젼 큐, 메모리 풀 등을 이용하여 CHILL 프로세스들의 실행을 제어한다. (그림 3)과 같이 CHILL 프로세스와 CRS 프로세스는 1 : 1로 매핑되며, 같은 시뮬레이션 커널의 제어를 받는 CRS 프로세스들은 하나의 UNIX 프로세스로 매핑된다.

SROS 시뮬레이션 커널은 우선순위 방식으로 프로세스를 스케쥴링하며 하나의 UNIX 타임 퀀텀 안에서 자신이 생성한 프로세스들을 제어한다. 프로세스 생성시 프로그램에서 지정한 우선순위를 가지고 생성하는데 프로그램에서 지정하지 않는 경우에는 가장 낮은 우선순위를 부여한다. 우선순위가 높은 프로세스는 실행 시 프로세스 지연, 도착하지 않은 메세지 수신 등과 같은 문장에 의하여 블록킹되기 전 까지는 프로세스 문맥 교환(context switching)이 일어나지 않는다.



(그림 3) CHILL 프로세스와 CRS 프로세스 간의 연관관계

SROS 커널은 <표 2>와 같이 프로세스 생성, 소멸, 지연, 재실행, 세마포 획득 및 프로세스 스케쥴링 과 같은 병렬 프로세스 실행 및 제어에 필요한 기본 기능들을 제공하고 자신이 제어하는 실행 블록에서 생성된 프로세스의 로컬 메모리 및 히프(heap) 영역을 관리하는 등 SROS가 제공하는 기본 기능을 제공한다.

3.2 IPC 관리기

SROS와 같는 실시간 병렬처리 전용 운영체제들은 실행 중 병렬 프로세스 들이 메시지를 주고 받으며 정의된 동기화 규칙에 따라서 실행될 수 있도록 프로세스 간의 통신 및 동기화 방법을 운영체제의 시스템 호출 수준에서 제공한다. 이러한 기능을 범용 운영체제 상에서 지원하기 위해서는 메시지 및 동기화 기능을 제공하는 런타임 라이브러리가 필수적으로 요구된다. 본 논문에서는 IPC 관리기를 이용하여 이를 해결하였다.

IPC 관리기는 분산 시스템 상에서 실행되는 병렬 프로세스들 간에 메시지 송수신 및 동기가 가능하도록 시그널 송신 및 수신, 버퍼 송신 및 수신 그리고 이벤트 지연 및 재실행 기능을 제공한다. <알고리즘 1>과 <알고리즘 2>는 프로세서나 프로세스로 메시지를 직접 전송하는 시그널 송신 및 수신 방법을, <알고리즘 3>과 <알고리즘 4>는 동일한 프로세서 내에서 메일박스를 이용하여 메시지를 전송하는 버퍼 송신 및 수신 방법을 기술하고 있으며, <알고리즘 5>와 <알고리즘 6>은 프로세스 간의 동기화를 위한 프로세스 지연 및 재실행 방법을 기술하고 있다.

● 시그널 송수신

병렬 프로세스 간에 시그널 메세지를 송수신하는 방법은 동일한 프로세서 내에서 생성된 프로세스 간에 송수신하는 경우와 분산 프로세서 상에서 실행되는 프로세스 간에 송수신의 경우로 구분이 된다.

프로세스가 시그널을 송신하면 <알고리즘 1>의 sendsig() 프리미티브가 호출되고 시그널 번호(sigId), 시그널을 수신할 프로세서 번호(processor), 시그널을 수신할 프로세스 인스턴스 번호(desPid), 시그널 메세지 포인터(sigMsg)가 매개변수로 넘어온다. 호출된 프리미티브는 라인 3과 같이 desPid 값을 검사하여 송신 시그널이 initial 시그널인지 normal 시그널인지를 검사한다. desPid가 0인 경우는 initial 송신의 경우이다. 이 경우 라인 4와 같이 동일한 프로세서 내에서 생성된 프로세스 중 해당 시그널을 요청한 프로세스가 있는지 검색하여 시그널을 요청한 프로세스 인스턴스 번호를 찾는다. desPid가 0이 아닌 경우는 normal 시그널이므로 매개변수에 할당된 인스턴스 번호가 수신 프로세스

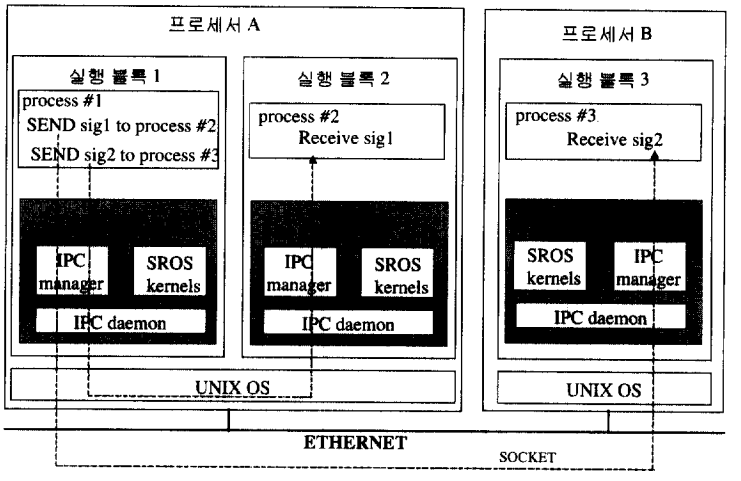
인스턴스 번호이다. 그 다음은 라인 8,9와 같이 시그널을 수신할 프로세서 번호 및 프로그램 번호를 검색한다. 동일한 프로세서 내에서의 시그널 송신인 경우는 dest_pg_id와 this_pg_id가 동일하므로 라인 11-14와 같이 수신할 프로세스의 PCB 블록의 시그널 지연 큐에 시그널을 등록하고 수신 프로세스를 실행한다. 다른 프로세서로 시그널이 송신되는 경우 IPC 매니저는 ipc_daemon을 통하여 해당 프로세서로 시그널을 송신한다. Ipc_daemon은 프로세서들의 인터넷 주소 및 실행 블록들이 사용하는 포트 번호 정보를 외부 파일로부터 읽어 메시지 송수신 테이블을 작성한 후 자신이 사용할 소켓을 열고 자신에게 할당된 포트 번호로 소켓을 바인딩한다. Ipc_daemon은 시그널 번호(sigId), 수신 프로세서 인터넷 주소, 수신 프로그램이 사용하는 포트 번호를 이용하여 시그널을 송신한다.

시그널이 전달될 프로세스 인스턴스를 찾은 후에는 라인 8,9와 같이 송수신 실행 블럭, 송수신 프로세스가 실행되는 프로세서에 대한 정보를 찾는다. 동일한 블록 내에서 송수신이 이루어지는 경우에는 라인 11과 같이 CRS가 스케쥴링 하지 못하게 로크(lock)를 설정하고 수신 프로세스의 상태를 검사한다. 수신 프로세스가 시그널을 대기하고 있는 "waiting" 상태이면 시그널 메세지를 해당 프로세스의 PCB 블록의 시그널 지연 큐(desPCB->signalDelayQueue)에 복사하고 해당 프로세스를 실행시킴으로 시그널을 전송한다. 만약 수신 프로세스가 시그널을 대기하는 "waiting" 상태가 아니면 시그널 메세지를 해당 PCB 블록의 시그널 지연 큐에 등록시킨 후 스케쥴링 로크를 해제한다. 시그널 지연 큐에 등록된 메세지는 해당 프로세스 인스턴스가 시그널을 수신하는 문장을 호출하는 경우 해당 프로세스에 전달이 된다.

```
1  int sendsig(int sigId,int processor,instance desPid, message
               sigMSG);
2  begin
3     if(desPid = 0) then
4        pid←get_link_pid(sigId,processor);
5     else
6        pid←desPid;
7     fi;
8     dest_pg_id←get_pgid(pid); /* get program id */
9     dest_or_id←get_orid(pid); /* get processor id */
10    if(dest_pg_id = this_pg_id) then /*signal send/receive is
                                       occurred in the same
                                       block */
11       lock_process_scheduling();
12       if(status(pid) is waiting)
13          Attach sigMsg into desPCB->signalDelayQueue;
14          reactivate_Process(pid);
15       else
16          Attach sigMsg into desPCB->signalDelayQueue;
17       fi;
18    else if (this_or_id = dest_or_id) /* the destination process is
                                       executing in another
                                       program/same processor */
19             send signal and MSG by Message Queue;
20       else /* the destination process is executing in another
                program/another processor */
21          send signal and MSG to ipc_daemon by socket;
22       fi;
23    fi;
24    unlock_process_scheduling();
25 end;
```

〈알고리즘 1〉 시그널 송신

다른 블록 간에 시그널 전송은 동일한 프로세서 내에서의 전송과 다른 프로세서 간의 전송으로 구분이 된다. (그림 4)와 같이 실행 블록 1에서 생성된 프로세스 #1이 실행 블록 2에서 생성된 프로세스#2에게 시그널 sig1을, 실행 블록 3에서 생성된 프로세스 #3에게 시그널 sig2를 송신하는 경우를 보자. <알고리즘 1>의 라인 18에서는 송신 프로세스와 수신 프로세스가 동일한 프로세서에서 동작하는지 검사한다. Sig1과 같이 동일 프로세서 내의 다른 블록으로 전달되는 경우에는 수신 프로세스를 생성한 실행 블록 정보, 수신 프로세스 인스턴스 번호 등의 정보를 가지고 메세지 큐와 공유 메모리를 이용하여 시그널을 전송한다. Sig2와 같이 서로 다른 프로세서 간의 시그널 전송인 경우에는 수신 프로세스를 생성한 실행 블록 정보, 네트워크 노드 정보, 수신 프로세스 인스턴스 번호를 이용하여 두 프로세서에서 실행되는 ipc_daemon간 입출력 소켓 채널을 형성하고 시그널을 프로세서 B로 송신한다. 프로세서 B의 ipc_daemon이 sig1을 수신하게 되면 실행 블록 3에 링크된 CRS에게 시그널 수신을 알리는 사용자 정의 시그널(SIGUSR2)을 이용하여 인터럽트를 전달하며 CRS는 수신될 프로세스의 상태 정보에 따라서 수신 대기 상태이면 해당 프로세스의 PCB 블록의 시그널 메세지 참조 영역에 송신 프로세스 식별자 및 메세지 관련 정보를 복사하고 대기중인 프로세스를 재활성화 시켜 시그널을 전달하게 된다. 그러나 해당 시그널을 수신할 프로세서가 존재하지 않는 경우에는 시그널 대기 큐에 해당 메시기를 등록하여 시그널을 전달할 수 있게 한다.



(그림 4) 블록간의 시그널 송수신

<알고리즘 2>는 프로세스가 시그널을 수신하는 절차이다. (그림 4)와 같이 프로세서 B에서 프로세스 #2가 sig2 시그널 수신을 요청하면 프로그램 제어는 IPC 관리기의 rcvsigcase() 프로미티브로 넘어온다. 호출된 프리미티브는 CRS가 스케쥴링하지 못하게 로크를 설정하고 프로세스 #2의 PCB 블럭의 시그널 지연 큐(thisPCB->signalDelayQueue)에 sig2 메세지가 있는지를 검사한다. 존재하면 지연되어 있던 시그널 중 가장 먼저 지연된 것을 큐에서 삭제하고 해당 메세지 내용을 수신한 후 CRS가 다시 스케쥴링할 수 있도록 스케쥴링 로크를 해제한다.

만약 프로세스 #2의 PCB 블럭의 시그널 메세지 지연 큐(thisPCB->signalDelayQueue)에 sig2 메세지가 들어있지 않으면 라인 22와 같이 시그널 수신을 호출한 프로세스를 지연시킨다. 이때 CRS가 다시 스케쥴링할 수 있도록 스케쥴링 lock 해제되며 프로세스 #2의 상태는 "waiting" 상태가 된다.

```
1  rcvsigcase(signal_list, msgLoc)
2  begin
3      L1 : do while(1)
4      begin
5          lock_process_schduling();
6          sigPtr←signal_list;
7          sigDelayQptr←thisPCB->signalDelayQueue;
8          L2: do while (sigDelayQptr /= NULL)
9          begin
10             L3 : do while(sigPtr /=NULL)
11             begin
12                 if(sigDelayQptr->sigId = sigPtr) then
13                     Delete the sigDelayQptr from the thisPCB
                       ->sigDelayQueue;
14                     msgLoc← sigDelayQptr->sigMsg;
15                     unlock_process_scheduling();
16                     return;
17                 fi;
18                 sigPtr++;
19             end L3;
20             sigDelayQptr++;
21         end L2;
22         delay_unlock_this_process();
23     end L1;
24 end;
```

〈알고리즘 2〉 시그널 수신

● 버퍼 송수신

버퍼는 프로세스간의 동기화 및 메세지 전송 기능을 제공하며, 동기화를 위하여 프로세스간의 공유 영역을 할당하고 메세지를 저장하기 위하여 해당 영역의 크기를 검사하여 저장 가능한 상태가 아니면 프로세스의 실행을 지연하는 기법을 이용하여 동기화 통신 기능을 제공한다. 버퍼를 이용한 메세지 전송은 시그널 전송과 달리 동일한 프로세서 내에서만 가능하다. <알고리즘 3>은 IPC 관리기가 병렬 프로세스들 간에 버퍼를 이용하여 메세지를 전송하는 방법이다.

(그림 5)와 같이 프로세스1이 송신 위치에 먼저 도착하여 메세지를 송신하면 IPC 관리기는 <알고리즘 3>의 라인 3과 같이 CRS가 스케쥴링하지 못하도록 로크를 설정한 후 해당 버퍼가 empty 인가 검사한다.

검사 결과 empty 이면 버퍼 수신 지연 큐(receive_delay_queue)에 지연된 프로세스중 가장 먼저 지연된 프로세스를 찾아 메세지를 전달하고 버퍼 큐의 빈 슬롯의 크기(slot_left)를 1 증가한 후 큐에서 삭제한다. 그리고 지연되었던 프로세스를 재실행시킨다.

라인 10과 같이 버퍼에 빈 공간이 있으면 송신 메세지를 버퍼에 넣고 송신 프로세스는 실행을 계속한다. 만약 버퍼가 full이면 자신을 버퍼 송신 지연 큐에 지연시킨다. 지연된 버퍼 메세지는 다른 프로세서에 재실행된다.

```
1  sendbuf(Buffer *buffer, int priority, char *msg);
2  begin
3     lock_process_scheduling()
4     if (check_buffer_empty(buffer)) then
5     if select_receive_delayed_pid(buffer->name)/=NULL then
                                  /* a receiver is present in the
                                     receive_delay_queue */
6              receiver←select_delayed_process(buffer->name);
7           delete_receiver(receiver); /* remove a receiver from
                                     the receive_delay_queue */
8           reactivate_process(receiver); /* re-activate the
                                        receiver */
9        fi;
10    else if slots_left /= 0     /* No receivers present and room
                                     in buffer */
11             allocate_kernel_loc(sizeof(msg));
12             enter_buffer_msg_list(msg);
13             slot_left slot_left - 1;
14       else /* No receivers present and no room in buffer */
15             delay_unlock_this_process();
16             lock_process_scheduling()
17             insert_send_delay_queue(->buffer,priority,->msg);
18       fi;
19    fi;
20    unlock_process_scheduling();
21 end sendbuf;
```

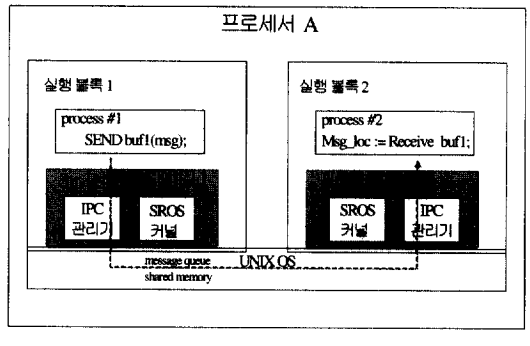
〈알고리즘 3〉 버퍼 송신

<알고리즘 4>는 병렬 프로세스가 버퍼를 이용하여 메세지를 수신하는 방법이다. (그림 5)와 같이 실행블록 B에서 생성된 프로세스 #2가 버퍼 buf1을 통하여 메세지를 수신할 경우 IPC 관리기의 rcvbuf()는 CRS가 프로세스 스케쥴링하지 못하도록 로크를 설정한다.

Buf1 버퍼에 도착한 메세지가 없으면 라인 5와 같이 프로세스 #2는 buf1 버퍼 수신으로 지연되며 버퍼 수신 지연 큐(receive_delayed_queue)에 등록한다. 만약 buf1 버퍼에 메세지가 있으면 라인 9와 같이 버퍼 큐에서 메세지를 하나 삭제하고 삭제한 메세지를 수신한 후 버퍼에 빈 공간이 하나 생겼음을 표시하고 버퍼 송신 지연 큐에 buf1 버퍼 송신으로 지연된 프로세스가 있는지 검사하여 있으면 프로세스를 재실행시킨 후 CRS가 다시 스케쥴링하도록 로크를 해제한 후 실행을 계속한다.

```
1  rcvbuf(Buffer *buffer, char *buf);
2  begin
3     lock_process_scheduling();
4     if buffer_queue(buffer->name) = NULL then
5        delay_unlock_this_process();
6        lock_process_scheduling()
7        insert_buffer_queue(->buffer, ->buf, this,
                              ->receive_delay_queue);
8     else /* a message exist in buffer queue */
9        delete_buffer_queue(buffer->name);
10       read_buffer_msg(buf);
11       slot_left slot_left +1;
12       if (sender select_send_delayed_pid(buffer->name)) != NULL then
13          reactivate_process(sender);
14       fi;
15    end;
16    unlock_process_scheduling();
17 end rcvbuf;
```

〈알고리즘 4〉 버퍼 수신



(그림 5) 버퍼를 이용한 메시지 송수신

버퍼를 이용한 메세지 전송의 경우 선언한 버퍼의 크기에 따라 전송시 프로세스들의 동기 방식이 달라진다. 버퍼 크기가 0인 경우는 송수신 프로세스는 Ada 랑데부와 같은 동기식 전송 방식으로 메세지가 전송된다. 즉 송신 프로세스가 송신 지점에 먼저 도착하여도 수신 프로세스가 수신지점에 도착할 때 까지 송신이 지연된다. 반대로 수신 프로세스가 먼저 수신 지점에 도착할 경우도 송신 프로세스가 송신 지점에 도착할

때 까지 지연된다. 버퍼 크기가 주어지는 경우 버퍼 full/empty에 의하여 송수신이 지연된다. 송신 프로세스는 버퍼가 full이 아닌 경우에는 비동기식 전송을 하고, full인 경우 버퍼가 not full 이 될 때 까지 송신을 지연한다. 수신 프로세스는 버퍼가 not empty 상태가 될 때 까지 수신을 지연하다가 버퍼로부터 메세지를 수신한다.

● 프로세스 동기화

IPC 관리기는 프로세스 인스턴스 간의 동기화를 위하여 이벤트를 제공한다. IPC 관리기는 동기화를 위하여 고유한 영역(location)을 할당하고 해당 영역에 대한 접근 허용 조건을 검사하여 프로세스 인스턴스들의 상태를 변경하는 기법을 이용한다. 프로그램에서 프로세스의 실행을 지연하기 위한 DELAY 명령과 지연된 프로세스를 재활성화(reactivation) 시키는 CONTINUE 명령을 사용하는 경우 해당 프로세스는 지연 및 재활성화 된다. <알고리즘 5> 및 <알고리즘 6>은 IPC 관리기가 제공하는 프로세스 지연 및 재활성 과정을 보여주고 있다.

프로세스가 프로세스 실행 지연(delay_event()) 프리미티브를 호출하면 <알고리즘 5>의 라인 3과 같이 CRS가 스케쥴링하지 못하도록 로크를 설정한 후 해당 이벤트 지연 큐에 빈 슬롯이 있는가 검사한다. 빈 슬롯이 존재하면 프로세스를 지연시키고 슬롯의 개수를 하나 감소시킨다. 그리고 프로세스의 상태를 "process_delayed"로 변경 시키고 delay_event()를 호출한 프로세스를 이벤트 지연 큐(event_delay_queue)에 등록시킨 후 설정된 스케쥴링 로크를 해제한다. 만약 빈 슬롯이 없는 경우에는 이벤트 지연 호출이 실패된다.

```
1   delay_event(Event *event, int priority)
2   begin
3       lock_process_scheduling();
4       if slot_left > 0 then
5           slot_left --;
6           process_state←process_delayed;
7           insert_event_delay_queue(->event, priority, this_process,
                                     ->event_delay_queue);\
8           delay_unlock_this_process();
9       else
10          CAUSE delay_event_fail;
11          exit;
12      fi;
13  end delay_event;
```

〈알고리즘 5 〉 프로세스 지연

프로세스가 프로세스 실행(continue_event()) 프리미티브를 호출하면 <알고리즘 6>의 라인 3과 같이 CRS가 스케쥴링하지 못하도록 로크를 설정한 후 해당 이벤트 지연 큐가 지연되어 있는 프로세스가 있는가 검사한다. 만약 지연된 프로세스가 존재하면 가장 먼저 지연된 프로세스를 선택하여 프로세스의 상태가 "process_delayed" 상태이면 슬롯의 개수를 하나 증가시킨 후 이벤트 지연 큐에서 해당 프로세스를 제거한다. 그리고 선택된 프로세스 인스턴스를 재실행 시킨 후 설정된 스케쥴링 로크를 해제한다.

```
1   continue_event(Event *event)
2   begin
3       lock_process_scheduling();
4       if event_queue != NULL then
5           instance←select_delayed_process();
6           if instance->process_state == process_delayed then
7               slot_left ++;
8               delete_event_delay_queue(event,instance);
9           fi;
10      else
11          CAUSE continue_event_fail;
12          Exit;
13      fi;
14      reactivate_process(instance);
15      unlock_process_scheduling();
16  end continue_event;
```

〈알고리즘 6〉 프로세스 재활성

3.3 Time 관리기

Time 관리기는 병렬 프로세스의 실시간 제약 조건을 관리하는 기능을 제공하기 위하여 각 실행 블럭 마다 별도의 시간 제어기를 할당하고 시간에 관련된 자료 구조를 관리, 제어한다. 타이머 관리 기능으로는 크게 유닉스의 경보 시그널(SIGALARM)을 이용한 경보 타이머 처리기, 프로세스 단위의 시간 특성을 제어하기 위하여 타임 테이블을 이용한 관리 기능을 제공한다. 또한 실시간 실행시스템에서 해당 프로세서의 어셈블리 코드를 이용하여 시간 간격을 측정함으로써 나노 초 단위의 정확성을 제공한다.

경보타이머 서비스는 CHILL 특성 중 특정 시간이 경과된 후에 프로세스의 시작 또는 시그널 송신 기능을 제공하기 위하여 상호 배제된 타이머 관리 영역과 알람 식별자를 이용하여 특정 시간동안 프로세스의 실행을 지연시키는 기법을 이용한다. 프로세스 단위의 시

간 특성을 제어하기 위하여 해당 프로세스 객체의 동기화 및 제어(synchronization and control object)를 위하여 할당된 영역의 우선순위, 프로세스 상태 정보를 변경한다. 만약 해당 프로세스가 지연 상태에 있으면 프로세스 상태와 우선 순위를 실행 모드로 변경하고 스케쥴링을 요청하고 타임 테이블 정보를 설정한다.

Time 관리기가 지원하는 기능은 <표 2>와 같다. 프로세스의 실행을 주기적으로 실행시키기 위한 time_cycle, 현재 시스템 시간을 초, 밀리초 값으로 가져오는 get_time, 현재 시스템 시간을 년, 월, 일 시,분, 초 형태로 가져오는 time_tod_get, 요구한 프로세스의 시간을 원하는 시간 만큼 지연 또는 일정 시간까지 지연시키는 time_sleep, 일정 시간 후에 요구한 프로세스에게 시그널 송신하는 time_sig, 지정한 시간에 요구한 프로세스에게 시그널 송신하는 time_tod_sig, 등록한 타이머를 취소하는 time_cancel, time_tod_cancel 등을 지원한다.

3.4 I/O 관리기

I/O 관리기는 병렬 CHILL 프로세스들의 입출력/화일 관리 기능을 효과적으로 제공하기 위하여 UNIX가 제공하는 I/O 시스템 호출을 이용하여 제공한다. I/O 관리기는 sros_open, sros_close, sros_read, sros_write, sros_create, sros_delete, sros_lseek, sros_ioctl 등을 제공한다. 상기 프리미티브들은 UNIX 시스템 호출과 이름의 충돌을 막기 위하여 각 프리미티브 이름 앞에 "sros_"를 붙였다.

3.5 지원 기능

CRS는 실시간 분산 병렬 처리를 위하여 <표 2>와 같은 기능을 제공한다. 각 기능들은 시스템 호출 들로 구성된 라이브러리 형태로 구성되어 있으며 사용자 프로그램에서 CRS 프리미티브를 직접 호출하여 사용할 경우는 언어에 관계 없이 사용할 수 있다. 그러나 CHILL 언어처럼 분산 병렬 처리 기능을 언어 수준에서 지원하는 경우에는 해당 컴파일러가 분산 병렬처리 문장을 CRS 라이브러리 호출 형태의 코드로 생성하여 호출한다.

SROS의 병렬처리 기능은 SROS 시뮬레이션 관리기가 지원하며 그 기능으로는 프로세스 제어를 위한 프로세스 생성(process_create), 프로세스 지연(process_delay), 프로세스 시작 (process_start), 프로세스 재실행(process_resume), 프로세스 이름 획득(process_name), 프로세스 상태 획득(process_status) 등이 있으며, 프로세스 간에 통신을 위한 시그널 송신(sendsig), 버퍼 메세지 송신(sendbuf)와 시그널 수신(rcvsigcase), 버퍼 메세지 수신(rcvbufcase) 등이 있다. 이 이외에도 실시간 지원 기능과 프로세스의 입출력을 지원하는 I/O 처리 기능 및 UNIX가 제공하는 라이브러리를 제공한다.

4. 시험 및 평가

CHILL 실행시간 지원 시스템은 ATM 교환기의 운영체제인 SROS가 제공하는 실시간 분산 처리 기능을

〈표 2〉 CRS의 지원 기능

	SROS 시뮬레이션 커널	IPC 관리기	Time 관리기	IO 관리기	라이브러리
지원 프리미티브	process_create, process_delay process_delete, process_exit process_isexist,process_ismain process_myid, process_name process_resume, process_start process_status process_suspend process_all_list, process_errno process_priority_get sema_create,sema_delete sema_give, sema_take process_priority_set message_create message_delete message_receive message_send, malloc, free	sendsig sendbuf rcvsigcase rcvbuffcase rcvbuf ipc_link ipc_rcvcase ipc_send delay continue	time_cycle time_get time_tod_get time_sleep time_sig time_cancel time_cycle_sig time_tod_cancel time_tod_sig time_toy_sig	sros_open sros_close sros_read sros_write sros_creat sros_delete sros_lseek sros_ioctl	*atoi, bcmp* *bcopy, bfill* *bzero, gets* *memchr, pritnf* *put, qsort* *scanf, sprintf* *sscanf, strcat* *strchr, strcmp* *strcpy, strlen* *strncat, strncmp* *strncpy, strch* *strstr, strtol*

범용 컴퓨터 시스템에서 실행할 수 있는 환경을 제공하기 위하여 개발되었으므로 SROS와 성능을 비교하였으며 기존의 다른 실행시간 지원 시스템과 특성 및 지원 기능을 비교하였다.

4.1 시 험

<표 3>은 CRS와 CHILL 시뮬레이션 환경 및 SROS의 성능 측정치이다. 각 시스템의 성능을 비교하기 위하여 동일한 하드웨어 상에서 시험을 하였다. 시그널 및 버퍼 전송은 전송 메세지 크기를 100 bytes로 제한하였다. 시그널 전송의 경우 분산 프로세서/SROS에서는 1초에 약 3200개 전송되어 1개의 시그널 전송에 소요되는 시간은 313 micro seconds가 소요된 반면 CRS에서는 516 micro seconds 기 소요되어 분산 프로세서 간에 시그널 송신의 경우 CRS가 1.65배 정도 더 소요된다. 버퍼를 이용한 메세지 전송시 SROS에서는 메세지당 26 micro seconds가 소요된 반면 CRS에서는 594 micro seconds 정도 소요되어 프로세서 내에서 내부 메세지 전송은 CRS가 SROS에 비해 22.85배의 오버헤드를 가진 것으로 측정되었다. 프로세스 생성, 소멸 및 문맥 교환에 소요되는 시간도 CRS가 SROS에 비해 약 2.95배 느린 것으로 나타났다. 이 결과로부터 ATM 교환기 프로그램을 범용 호스트 컴퓨터에서 시뮬레이션 하는데 무리가 없음을 볼 수 있었다.

〈표 3〉 실행시간 비교

		CRS on SPARC micro seconds (SROS:CRS)	CSE on SPARC micro seconds (SROS:CSE)	SROS for SPARC (micro seconds)
프로세스 트랜잭션 시간(프로세스생성, 소멸, 문맥 교환)		1,580 (2.95 : 1)	1,641 (3.06 : 1)	536
메세지 송/수신	시그널 송수신	516 (1.65 : 1)	601 (1.92 : 1)	313
	버퍼 송/수신	594 (22.85 : 1)	6,605 (254.14 : 1)	26
세마포 획득/양도		162 (13.5 : 1)	660 (55 : 1)	12

4.2 비교 평가

<표 4>는 기존의 CHILL 실행시간 지원 시스템들의 특성 및 지원기능을 비교한 것이다. 본 논문에서 제안한 CRS는 ATM 교환기 시스템의 운영체제로 사용되고 있는 SROS가 요구하는 분산 실시간 병렬처리 기능을 모두 수용하였다. CRS는 분산 병렬 처리 기능을 호스트 컴퓨터에서 제공한다는 점과 프로세스 인스턴스를 효과적으로 제어하기 위하여 단일 실행 블록에서 생성되는 프로세스를 하나의 UNIX 프로세스로 매핑시킨 것은 CHIPSY CRS와 유사하다. 그러나 다중 블록간에 시그널 송/수신하는 메커니즘 및 프로세스 스케쥴링 방법, 실시간 처리기능 등에서 차이가 있다.

CHIPSY CRS는 프로세스 스케쥴링시 CHO_CRS나 CSE가 사용하는 방식인 라운드-로빈을 사용하는 반면에 CRS는 우선순위 기반 스케쥴링 방식을 사용한다. 이는 우선순위를 기반으로 하여 스케쥴링을 하는 실시간 운영체제들의 요구사항과 부합된다. CHIPSY CRS는 다중 블록간에 전역 시그널을 전송하기 위하여 수신할 프로세스 인스턴스를 반드시 지정하여야 하지만, CRS는 수신 프로세스의 인스턴스 번호를 모를 경우에도 시그널을 전송할 수 있는 방법을 제공한다. ATM 교환기 시스템[1]의 운영체제인 SROS는 실행 블록 간에 다량의 데이터 처리를 효율적으로 지원하기 위하여 메세지 크기를 1,008 bytes까지 지원한다. 그러나 CHIPSY CRS는 메세지 크기를 최대 256 bytes로 제한하므로 목적 시스템용으로 개발된 프로그램을 호스트 시뮬레이션하기 위해서는 256 bytes가 넘는 전송 데이터의 경우 전송 메세지의 크기를 작게 분할하여 여러 번 전송해야 하는 등 프로그램의 변경이 불가피하다. CRS에서는 메세지 전송의 최대 크기를 1,008 bytes까지 제공하여 시그널 전송 메세지 크기 제약에

〈표 4〉 실행시간 지원 시스템 비교

	CHO-CRS	CSE	CHIPSY CRS	CRS
프로세스 생성 (프로세스 생성 방법)	UNIX 쓰레드 (n cp : 1up)	UNIX 쓰레드 (1 cp : 1ut)	CHIPSY CRS 쓰레드 (1 ccp : 1ut)	CRS 쓰레드 (1 cp : 1ut)
스케쥴링 방법	우선순위	라운드-로빈	라운드-로빈	우선순위
IPC 지원	단일 프로세서/단일 블럭/다중프로세스 통신	단일 프로세서/다중 블록/다중프로세스 통신	단일 프로세서/다중 블록/다중프로세스 통신	분산 프로세서/다중 블록/다중프로세스 통신
프로세서간 IPC 메세지 크기(최대)	지원 안함	지원 안함	256 bytes	1,008 bytes
실시간 기능 지원	지원 안함	stop watch 타이머만 제공	alarm, stop watch 타이머만 제공	다양한 기능 제공<표 2>
I/O 지원	UNIX i/o	UNIX i/o	CHIPSY CHILL i/o	UNIX i/o
DBMS와 연동	연동 불가	연동 불가	DBMS와 연동상 제약	분산DBMS와 연동

cp : chill process, ccp : chipsy chill process, ut: unix thread, up: unix process

따른 프로그램의 수정이 발생하지 않도록 하였다.

<표 4>에서 보는 바와 같이 CHO_CRS, CSE 및 CHIPSY_CRS는 단일 프로세서 상에서 실행되는 프로세스 간의 IPC 전송을 지원하므로 단일 프로세서 상에서 실행되는 블록의 경우에는 적용이 가능하나 분산처리 기능을 요구하는 응용에는 적용이 불가능하다. CHIPSY_CRS가 프로세스 간에 주고받는 IPC 메시지의 크기가 256 bytes로 제한되어 있고 제한된 실시간 기능만을 제공함으로 실시간 분산 병렬 특성을 갖는 목적 시스템용 프로그램을 범용 컴퓨터에서 실행하기 위해서는 프로그램의 기능을 제한하거나 변경해야만 하였다. 그러나 CRS는 <표 2>와 같이 분산 병렬처리 기능과 다양한 실시간 처리 기능을 제공함으로 목적 시스템용으로 개발된 교환기 소프트웨어를 수정하지 않고 호스트 시스템에서 투명하게 실행할 수 있게 하였다. 또한 다수의 사용자가 동일한 호스트 시스템에서 동시에 블록 실행을 가능하게 하여 교환기 소프트웨어 개발의 생산성을 향상시키므로 총 개발 기간을 단축 시키는 효과를 제공하였다.

5. 결 론

내장형 시스템들은 제한된 개발환경을 갖기 때문에 소프트웨어는 범용 컴퓨터에서 개발하고 교차 컴파일한 후 목적 시스템에 탑재하여 운영 및 시험하는 방법을 사용하고 있다. 이 경우 소프트웨어를 개발하는 시스템과 운용 및 시험하는 시스템이 달라 시험하는데 많은 시간과 노력이 소모되며, 다수의 개발자가 하나의 목적 시스템을 동시에 사용할 수 없어 소프트웨어 개발 시 많은 어려움이 따른다.

본 논문에서는 이러한 문제점을 해결하기 위하여 내장형 시스템 운영체제가 제공하는 실시간, 분산, 병렬처리, 동기화와 같은 기능을 범용 컴퓨터 상에서 완벽하게 제공하는 CHILL 실행시간 지원 시스템을 개발하였다. CRS는 병렬처리를 지원하는 SROS 시뮬레이션 커널과 분산 프로세스간의 메세지 전송 및 동기화를 지원하는 IPC 관리기, 실시간 지원을 위한 Time 관리기, 입출력을 지원하는 I/O 관리기로 구성된다.

CRS는 목적 시스템의 실행환경을 범용 컴퓨터 상에서 투명하게 제공하므로 다음과 같은 효과를 얻을 수 있다. 첫째로 목적 시스템용으로 개발된 소프트웨어를 범용 컴퓨터에서 수정 없이 재 컴파일하여 실행할 수 있는 환경을 제공함으로 목적 시스템이 개발되지 않는 시스템 개발 초기에 프로그램 개발 및 시험이 가능하다. 둘째로, 프로그램의 개발, 실행 및 시험으로 이루어지는 프로그램 개발 주기가 한 시스템에서 이루어지므로 개발 노력 및 기간을 단축할 수 있다. 세째로, 하나의 컴퓨터상에서 동시에 여러 명이 프로그램을 개발할 수 있는 효과적인 실행환경을 제공하므로 프로그램 개발에 소요되는 비용을 절감시킬 수 있다. 넷째로, 목적 시스템의 하드웨어와 독립적으로 프로그램을 개발할 수 있어서 목적 시스템 변경에 따른 프로그램 개발기간 손실을 차단한다. 다섯째로, 목적 시스템 프로그램의 호스트 시험을 위한 기반 실행환경을 제공한다. 이와 같이 CRS는 실시간 분산 병렬 특성을 갖는 내장형 시스템 소프트웨어 개발에 필요한 호스트 실행환경을 제공함으로 소프트웨어 개발의 생산성을 향상시켰을 뿐만 아니라 호스트에서 다양한 시험을 가능케 하여 고 신뢰도를 갖는 소프트웨어를 개발할 수 있도록 하였다.

CRS는 ATM 교환기와 같은 대형 내장형 시스템의 운영체제인 SROS가 제공하는 실시간 병렬 분산 처리 기능을 범용 컴퓨터 상에서 제공하기 위하여 개발되었지만 UNIX 시스템 호출을 이용하여 하드웨어 독립적으로 개발하였기 때문에 분산 병렬 처리를 요구하는 내장형 시스템 개발의 호스트 개발 도구로 사용될 수 있다.

참 고 문 헌

[1] ITU-T, CHILL Recommendation Z.200, ISO/IEC 9496, Geneva, Swiss, 1996.

[2] Young-Boo Kim, Soon Seok Lee, Chang Hwan Oh, Young Sun Kim, Chimoon Han and Chu Hwan Yim, "An Architecture of Scalable ATM Switching System and Its Call Processing Capacity Estimation," ETRI Journal, Vol.18, No.3, pp.107-125, 1996.

[3] 조칠희, 김성희, 홍진표 "CHILL 런타임 시스템 설계 및 구현", 한국정보과학회 춘계학술 발표회 논문집, pp.441-444, 1988.

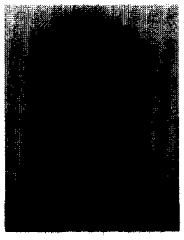
[4] Hang-Gu Park, "Narrowband and broadband ISDN technology implementation by TDX Switching System," Proceedings of International Symposium on Telecommunications, Slovenia, pp.119-122, 1992.

[5] Kyung-Suk Park et al., "Simulation Environment for Telecommunication Systems," Proceedings of

International Conference on Communication Technology, pp.1387-1391, 1994.
[6] Sung-Ik Jun et. al., "SROS : A Dynamically-Scalable Distributed Real-time Operating System for ATM Switching Network," Proceedings of IEEE Globecom98, pp.2918-2923, 1998.
[7] 송후봉, 유재우, 백의현, "병행 프로그램 디버깅을 위한 결정적 재실행 시스템", 정보과학회논문지(B), 제24권, 제2호, pp.218-230, 1997.
[8] Trond Borsting, "A CHILL Run-time System support for Teletesting," Proceedings of the 5th CHILL Conference, pp.96-103, 1990.

백 의 현

e-mail : ehpaik@etri.re.kr
1984년 숭실대학교 전자계산학과 졸업
1987년 숭실대학교 전자계산학과 석사
1997년 숭실대학교 전자계산학과 박사
1987년~현재 한국전자통신연구원 교환전송기술연구소 책임연구원
관심분야 : 병렬처리, 프로그래밍 언어론, 미들웨어 등



장 종 현

e-mail : jangjh@etri.re.kr
1988년 경북대학교 전자공학과 (공학사)
2000년 충남대학교 전자공학과 (공학석사)
현재 한국외국어대학교 전자공학과 박사과정
1988년~1994 대우통신㈜ 종합연구소
1994년~현재 한국전자통신연구원 교환전송기술연구소 선임연구원
관심분야 : 네트워크 보안, 이동통신, 미들웨어 등

이 동 길

e-mail : dglee@etri.re.kr
1983년 경북대학교 전자공학과 (공학사)
1985년 한국과학기술원 전산학석사
1994년 한국과학기술원 전산학박사
1985년~현재 한국전자통신연구원 책임연구원
관심분야 : 컴파일러 구성론, 프로그래밍 언어론, 미들웨어 등