SPAX를 위한 OSF/1 AD3 기반의 마이크로 커널 초기화 설계 및 구현

김 정 녀[†] · 조 일 연[†] · 이 재 경[†] · 김 해 진[††]

요 약

마이크로 커널 기반 운영체제는 종래의 통합커널에 비해 비교적 속도가 늦지만 운영체제 모듈성, 이식성 측면에서 장점이 있어 다중컴퓨터 시스템에 적합하다. 다중컴퓨터 시스템용 운영체제가 시스템의 기능을 원활하게 수행할 수 있도록 하기 위해서는 부트 시의 정보를 이용하여 처리기의 각 장치 및 메모리를 시스템에 알맞게 초기화하여야 할 것이다.

본 논문에서는 OSF/1 AD3를 기반으로 한 운영체제인 고속병렬컴퓨터의 OSF/1 AD3 MISIX 마이크로 커널 초기화에 대해 기술한다. 클러스터링 기반 고속병렬처리 시스템인 고속병렬컴퓨터의 초기화를 부트, 하드웨어 관련 초기화, 메모리 주소공간 구축 등의 관점에서 기능을 소개하고, 이를 시험한 내용을 시험 환경을 바탕으로 기술한다. 구현된 마이크로 커널은 운영체제 이식 작업의 일부로 4개의 처리기를 갖는 단일노드 시스템에서 시험이 이루어졌다.

The Design and Implementation of OSF/1 AD3 Based-Microkernel Initialization for SPAX

Jeong-nyeo Kim[†] · Il-yeon Cho[†] · Jaekyung Lee[†] · Haejin Kim[††]

ABSTRACT

In comparison to traditional monolithic kernel, the microkernel based operating system has slower speed. But Microkernel based OS suites for multi-computer system, because It has benefits in the modularity and portability point of view. Each unit and memory of a processor must be initialized by using the boot information so that the multi-computer system OS can actively run the function of the system.

This paper describes the microkernel initialization of OSF/1 AD3 MISIX that is based on OSF/1 AD3 for SPAX. It will introduce the initialization of microkernel for the SPAX which is High-speed Parallel Processing system in terms of Boot, Initialization related hardware and memory address space construction. This paper will also state the test result based on test environments. Microkernel tested in single node system that has 4 processors.

1. 서 론

마이크로 커널 기반의 운영체제 설계[1]는 요즈음 들어서 그리 선호되고 있지 않다. 그러나 마이크로 커널 구조는 운영체제 모듈성 및 이식성의 관점에서 많은 장점들을 가지고 있어 클러스터링 기반의 고속병렬처리 시스템에 적합하다.

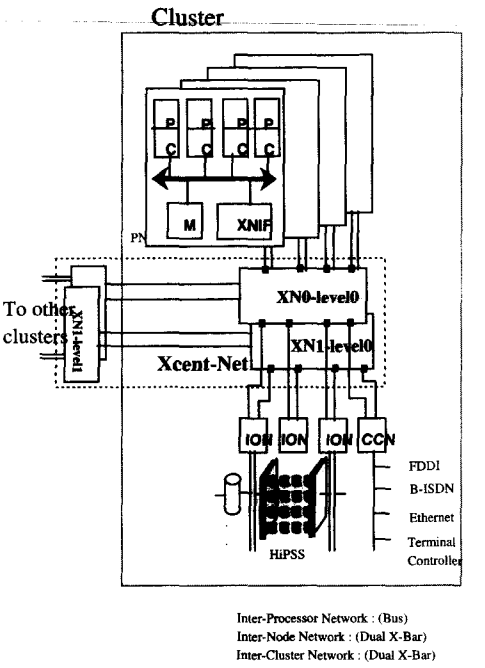
고속병렬컴퓨터(Scalable Parallel Architecture computer based-on X-bar network, 이하 SPAX)는 클러스터링 기반의 고속병렬처리 시스템으로 노드, 클러스터의 계층적 구조를 갖는다[2]. SPAX 시스템의

† 정 회 원 : 한국전자통신연구원 시스템S/W 연구실
†† 종신회원 : 한국전자통신연구원 시스템S/W 연구실
논문접수 : 1997년 5월 30일, 심사완료 : 1998년 2월 25일

구조는 아래 (그림 1)과 같다. 즉 최대 8개의 노드가 클러스터를 구성하며 최대 16개의 클러스터가 전체 시스템을 구성한다. 노드 및 클러스터를 연결하는 내부 연결망은 10x10 크로스바(Cross-Bar) 구조를 갖는 X-cent Net을 사용한다. 하나의 노드는 SMP (Symmetric Multi processor) 구조를 가지며 최대 1 기가 바이트의 메모리 및 4개의 인텔 P6 처리기가 장착된다. 노드간 통신은 IPC (Inter-Process Communication)에 의해 수행되며, 이 IPC 기능은 쓰레드 수행 위치에 관계없이 동일한 메커니즘을 제공한다. 각 노드마다 OPB(Orion PCI Bridge), APIC(Advanced programmable Interrupt Controller), XNI(Send Network Interface), RNI(Receive Network Interface), OMC(Orion Memory Controller) 등은 공통되며 지역 자원 및 통신 장치, 입출력 장치 등은 OPB0, OPB1을 통해 연결된다.

OSF/1 AD3 MISIX(Microkernel based Single system Image uniX)는 인텔 파라곤과 같은 수퍼컴퓨터에서 부터 SCSI 또는 ATM, Ethernet 등에 연결된 워크스테이션 클러스터까지의 범주에 속하는 Massively Parallel Processing(이하 MPP) 환경을 위한 유닉스 단일 시스템 이미지(Single System Image) 기능을 제공하는 운영체제인 OSF/1 AD3 운영체제를 기반으로 SPAX의 하드웨어와 시스템 형상에 맞게 개발되었다[2]. 또한 OSF/1 AD3 MISIX는 SPAX의 부운영체제로 각 노드마다 운영체제가 탑재되나 사용자에게는 유닉스 단일 시스템 이미지를 제공한다. OSF/1 AD3 MISIX는 Mach 마이크로 커널 기반으로 마이크로 커널과 운영체제 서버로 구성된다. 마이크로 커널은 SPAX 하드웨어 자원을 효율적으로 관리하며, 응용 프로그램 인터페이스를 제공하는 운영체제 서버들이 시스템의 자원들을 쓸 수 있도록 시스템 서비스를 제공한다. 운영체제 서버는 운영체제의 기본 특성인 프로세스 관리, 파일 관리, 네트워크 관리 등의 기본 기능을 갖는 유닉스 서버로 사용자 API를 제공한다. OSF/1 AD3 1980년대 후반부터 선호되었던 MISIX는 시스템 종속적인 모듈과 플랫폼 관련 하드웨어 종속적인 부분이 모두 마이크로 커널에 있으므로, 마이크로 커널이 SPAX 하드웨어 구조 및 SPAX 시스템 분산 구조에 적합하게 설계 및 구현되었다.

본 논문에서는 이러한 OSF/1 AD3 MISIX의 마이크로 커널 초기화 설계 및 구현 내용을 기술한다.



(그림 25) SPAX 시스템의 구조
(Fig. 1) Architecture of SPAX

2절에서는 MPP 환경을 위한 UNIX 단일 시스템 이미지 기능을 제공하는 운영체제인 OSF/1 AD3 MISIX를 소개한다. 3절에서는 OSF/1 AD3 MISIX의 마이크로 커널 초기화의 설계 및 구현된 내용을 기술하고, 4 절에서는 이러한 OSF/1 AD3 MISIX를 위하여 설정한 시험 환경과 이러한 환경에서 시험한 결과를 나타낸다. 마지막으로 5절에서는 앞으로의 연구 방향을 간단하게 설명한다.

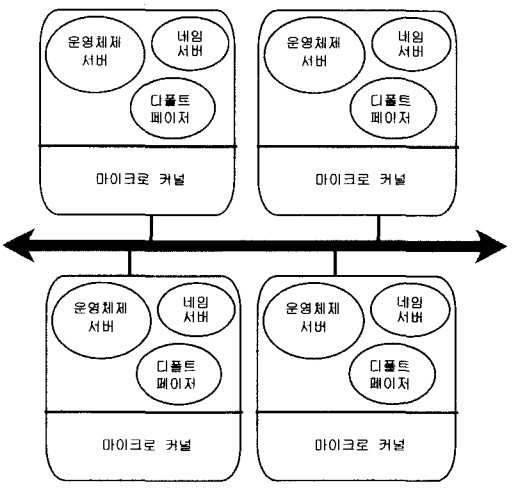
2. OSF/1 AD3 MISIX의 기본구조

OSF/1 AD3 MISIX는 MPP환경을 위하여 확장 가능하고, 고 성능인 유닉스 단일 시스템 이미지를 구현하였다. 즉 X-cent Net으로 연결된 다중 컴퓨팅 노드들이 비록 메모리를 서로 공유하지는 않지만 모든 노드들 사이의 단일 시스템 유닉스 이미지를 만든다. 또한 OSF/1 AD3 MISIX는 성능(Performance),확장성(Scalability), 견고성(Robustness) 등의 기능을 갖

는다. 전형적인 운영체제의 동작은 응용 프로그램의 각 지역 노드에 의해 수행된다.

OSF/1 AD3 MISIX 환경은 두 개의 주요 부분을 포함한다. 하나는 Mach 기반의 마이크로 커널로 기존의 운영체제에서 사용되는 기본적인 기능을 제공한다. OSF/1 AD3 MISIX에 포함된 마이크로 커널은 다중 노드들을 통한 분산된 동작을 강화시켰다. OSF/1 AD3 MISIX 환경의 다른 하나는 NORMA domain을 통해 기존의 유닉스 단일 시스템 이미지를 구현한 운영체제 서버이다. 보통 마이크로 커널 위에 전형적인 운영체제의 기능으로 구현된 서버는 사용자 API를 제공하며 시스템 호출 수준에서 단일 시스템 이미지를 지원한다.

시스템이 4개의 노드로 구성된 경우에 OSF/1 AD3 MISIX의 구성 모습은 (그림 2)와 같다. 각 노드에 탑재되는 마이크로 커널 및 운영체제 서버는 모두 동일하다.



(그림 26) OSF/1 AD3 MISIX 마이크로 커널 구조
(Fig. 2) Structure of OSF/1 AD3 MISIX Micro-kernel

2.1 마이크로 커널

마이크로 커널은 노드의 가장 낮은 수준의 물리적인 자원 관리에서부터 분산 프로세스간 통신(Distributed IPC)과 같은 상위 수준의 인터페이스까지 제공한다. 마이크로 커널은 5가지 주요 모듈로 분류할 수 있으며, 이를 통하여 시스템 전반에 걸친 서비스를 제공한다.

- 타스크 및 쓰레드 관리 : 타스크 및 쓰레드의 생성에서 소멸까지에 관련한 제어를 담당한다. 즉 타스크 및 쓰레드의 생성, 소멸과 쓰레드에 대한 동기화, 스케쥴링, 처리기 할당 등의 기능을 제공한다.
- 프로세스간 통신 : 쓰레드 상호간의 통신에 대하여 같은 노드 내에 존재하는 지 또는 서로 다른 노드에 존재하는지에 관계없이 동일한 프로그램 인터페이스를 제공한다. 프로세스간 통신의 기본 기능인 다양한 포트 기능을 제공한다.
- 메모리 관리 : 노드 내의 지역 메모리를 페이지로 관리하며, 주소를 메모리 옵셋으로 매핑하는 가상 주소 변환 등을 담당한다. 또한 다중 노드 상에서 접근하는 메모리 데이터의 일관성을 유지하여 주는 XMM(eXternal Memory Management)이 있어 메모리 일관성을 유지시킨다.
- 시스템 호출 및 인터럽트 처리 : 처리기간 인터럽트 및 XNIF 다양한 인터럽트 처리, 트랩 처리, 예외 처리, RTC 읽기/쓰기 및 동기화 처리와 같은 주로 하드웨어 종속적인 부분의 처리를 담당한다.
- 장치 관리 : 하위 수준의 장치를 지원하고, 데이터 이전이나 장치를 제어 관리한다. 또한 비동기식 및 동기식 입출력 기능을 제공한다.

2.2. AD3 MISIX 운영체제 서버

AD3 MISIX 운영체제 서버는 Mach 마이크로 커널 위에서 실행하는 분산 운영체제로 몇 가지 서버가 존재하지만 보통은 유닉스 API를 제공하는 osf1_server를 운영체제 서버라 한다. 몇 가지 서버란 마이크로 커널 실행 과정에서 운영체제 서버를 메모리에 적재하고 실행시켜주는 부트스트랩(bootstrap) 서버, 네이밍을 관리하는 네임 서버(name server), 교체 공간을 관리하여 주는 디폴트 페이저(default pager) 등을 말한다. 운영체제 서버는 두종류의 서버로 구성된다. 각 노드에는 그 노드 상에서 실행중인 타스크에 대한 시스템 호출 실행을 제공하는 프로세스 관리 에이전트(Process Management Agent: 이하 PMA)가 있다. 덧붙여서 중앙 프로세스 관리 서버인 프로세스 관리 마스터(Process Management Master: 이하 PMM)가 있다. 이 서버는 프로세스간 관계 관리를 포함하여 모든 노드들에 대해서 다양한 형태의 전역적인 조정을 한다. PMM 서버는 항상 부트 노드에서 동작한다. 이러

한 확장성 있는 다중 서버 구조로 프로세스 관리, 네트워킹, 데이터 캐싱, 그리고 다른 유닉스 기능들을 제공한다.

3. 마이크로 커널 초기화 설계 및 구현

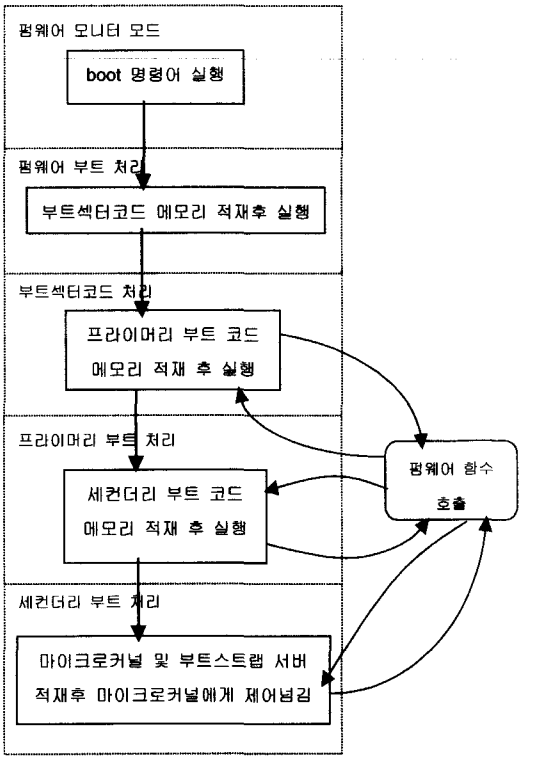
SPAX 시스템의 노드를 구성하는 각 처리기는 인텔 P6 칩으로 그 실행되는 주소 모드는 실제 모드(real mode)와 보호 모드(protected mode)로 구분한다. 두 모드의 차이점은 주소 번역 시 실제 모드에서는 20 비트를 이용하며 보호 모드에서는 32 비트를 이용한다[12]. 시스템의 리셋 상태에는 각 처리기는 실제 모드의 상태이나 펌웨어 모니터 모드에서는 보호 모드로 변경되어 실행된다. 그러므로 부트 코드 및 마이크로 커널 실행 코드 모두 보호 모드에서 실행된다.

마이크로 커널 초기화는 SPAX 펌웨어 모드에서 부팅을 수행하여 서버가 구동되기 전에 마이크로 커널 수행 환경을 만들고 서버와의 인터페이스를 구축하기 까지를 말한다. 이러한 초기화 과정이 종료된 후 마이크로 커널 및 커널 타스크들이 구동된다. 초기화 과정의 설계는 크게 두단계로 첫째는 시스템 구성 및 전반적인 부트 처리를 소개하고 둘째로는 부트 시에 넘어온 정보를 이용하여 처리기의 각 장치 및 메모리를 초기화 하는 과정을 크게 세부분으로 나누어서 기술한다. 또한 마지막 절에서는 다른 운영체제와 비교한 초기화 기법을 기술한다.

3.1 부트 단계

SPAX 시스템의 부트는 펌웨어 모니터 모드에서 시스템 관리자의 boot 명령어에 의해 수행된다. 부트는 노드 내에서 주처리기가 메모리에 실행될 운영체제 커널인 마이크로 커널과 부트스트랩 서버를 적재시키고, 시스템 환경에 맞는 부트 정보를 스택에 쌓아 실행될 주처리기에게 마이크로 커널의 시작 주소로 제어를 옮겨 시작시키는 것이다.

부트 과정은 위의 (그림 3)과 같으며 그 설계는 크게 네단계로 나누어서 기술한다. 첫번째는 부트를 실행시키는 단계로 시스템 구성 및 전반적인 부트 처리를 소개하고 두번째 단계는 프라이머리 부트 단계로 세컨더리 부트 프로그램을 메모리에 적재하고 세컨더리 부트 프로그램을 실행시키는 단계이다. 세번째 단계인 세컨더리 부트 단계에서는 마이크로 커널과 부트스트랩 서



(그림 3) SPAX 부트 과정
(Fig. 3) Boot procedure of SPAX

버를 메모리에 적재하고 부트 시에 각종 정보를 마이크로커널에게 스택을 통해 넘겨주고 실행시킨다. 실행을 시작한 마이크로 커널은 부트 정보를 이용하여 처리기의 각 장치 및 메모리를 초기화한다. 마지막으로 프라이머리와 세컨더리 부트 프로그램이 부팅 과정에서 터미널 입출력, 디스크 입출력, 노드간 메시지 송수신 등을 하여야 하는데, 간접 호출에 의한 펌웨어 함수 호출을 사용하였다.

3.1.1 마이크로 커널 부트

펌웨어 모니터 모드에서의 boot 명령어는 부트 하드 디스크의 부트 섹터로부터 부트 코드를 읽어서 메모리에 적재한다. 메모리에 적재된 부트 섹터 코드는 프라이머리 부트 코드를 메모리의 정해진 번지에 적재한 후에 프라이머리 부트 코드에게 제어를 넘긴다. 부트 섹터 코드는 부트 하드 디스크의 0번째 블록에 있는 프로그램으로 프라이머리 부트 프로그램을 메모리의 정해진 번지에 적재하고 실행시키는 기능을 한다. 부트 섹터

프로그램은 보통 X86 계열의 PC에서 표준화된 것으로 단순히 프라이머리 부트 프로그램을 메모리의 0x7c00 번지에 적재하고 실행시키는 프로그램으로 실제 모드에서 실행된다. 그러나 SPAX 시스템에서는 보호 모드에서 실행되어야 하므로 보호 모드의 실행 코드이다.

3.1.2 프라이머리 부트 단계

프라이머리 부트 코드는 하드 디스크의 파티션 테이블을 읽어서 유닉스 파티션을 찾아내고, 찾아낸 유닉스 파티션 중에 루트 파일 시스템의 정보를 가져온다. 그 루트 파일 시스템의 유효성을 확인 한 후에 디폴트 세컨더리 부트 프로그램을 메모리 정해진 번지에 적재한다.

부트 섹터 코드에 의해 0x7c00에 적재된 프라이머리 부트 코드는 실행 시에 SCSI 하드 디스크의 부트 섹터 블록인 0번째 블록을 읽어서 유닉스 파티션을 찾는다. 그리고 0x1be에는 파티션 테이블이 있어 네 개의 파티션을 가질 수 있도록 하며, 그 구조는 부트 파티션 식별자, 시작 헤드, 시작 섹터, 파티션의 종류(DOS or UNIX) 등의 필드로 구성된다. 그 중 디폴트로는 네 번째 파티션을 유닉스 파티션으로 한다. 프라이머리 부트 코드는 유닉스 파티션 중 루트 파일 시스템을 찾기 위해 VTOC(Volume Table Of Contents) 정보가 들어있는 하드 디스크의 30번째 블록을 읽는다. 그 블록으로 읽게 되는 16개의 파일 시스템 정보를 이용하여 루트 파일 시스템의 시작 섹터 주소를 얻는다.

프라이머리 부트 프로그램은 먼저 부트 섹터 블록인 디스크의 첫 번째 블록을 메모리에 읽는다. 그 중 네 개의 파티션 테이블을 검색하여, 부트 파티션을 찾는다. 파티션 테이블 중에 부트 파티션을 찾으면, 부트 파티션의 시작 실린더와 섹터를 가져온다. 그런 후에 그 것을 이용하여 프라이머리 부트의 전체 프로그램을 메모리 정해진 번지에 적재한다. 참고로 OSF/1 AD3 MISIX 운영체제의 프라이머리 부트 프로그램의 크기는 16 개의 블록(약 8K) 이어서 부트 섹터 코드가 적재한 512바이트의 프라이머리 부트 코드가 프라이머리 부트 코드 전체를 다시 적재해야 한다. 전체 프라이머리 부트 코드가 적재되어 새로 시작된 부트 코드는 먼저 펌웨어 함수 호출을 이용하여 기본 메모리와 확장 메모리 크기를 구한다. 그런 후에 하드 디스크의 부트 파티션 정보와 VTOC 블록에 있는 루트 파일 시스템의 정보를 읽는다. 디폴트로는 /mach_boot 프로그램을 메모리의 커널 적재 주소(0x1000000)에 임시로 적재한다. 참고로 여기에서 적재된 세컨더리 부트 프로그램은 그 프로그램이 시작되면서 부트 코드 주소인 0x1000번지로 복사되어 실행된다. 적재에 실패한 경우에는, 시스템 관리자로 부터 세컨더리 부트 프로그램 정보를 다시 얻는다. 그러나 적재에 성공하면 프로그램 적재 시에 얻은 프로그램 시작 주소 , 부트 디바이스 정보 등을 스택에 쌓아 세컨더리 부트 프로그램에게 제어를 넘긴다.

3.1.3 세컨더리 부트 단계

세컨더리 부트 프로그램의 시작으로 제어가 이양되면 세컨더리 부트 프로그램은 먼저 부트 코드 주소로 세컨더리 부트 프로그램을 복사 한 후에 그시작으로 제어를 옮긴다. 그런 후에 시스템 관리자의 입력을 받아들인다. 만약 시스템 관리자의 입력이 있을 경우에는 그 입력에 따라 마이크로 커널과 부트스트랩 서버의 경로를 찾아 메모리에 적재해주고, 입력이 없는 경우에는 디폴트 마이크로 커널과 부트스트랩 서버 프로그램을 메모리의 커널 적재 주소(0x100000)에 차례로 적재한다. 적재한 후에 마이크로 커널로 제어를 넘겨주는데 그 때 메모리 정보, 적재된 프로그램 및 인수 정보 등을 스택에 쌓아서 마이크로 커널에게 넘겨준다. 세컨더리 부트 프로그램이 마이크로 커널에게 넘겨주기 위해 스택에 쌓는 부트 정보는 적재된 메모리 top 주소, 확장 메모리 크기, 기본 메모리 크기, 커널 심볼 정보, 커널 인수 리스트 정보, 부트스트랩 심볼 정보, 부트스트랩 인수 정보, 부트 쓰레드 정보, 환경변수 리스트 정보 등이 있다.

SPAX에서는 콘솔 입출력에 COM1 포트를 이용하므로, 이것 또한 펌웨어 함수 호출을 이용한다. 마이크로 커널과 부트스트랩 서버 프로그램의 적재를 위해 디폴트 마이크로 커널로는 /mach_kernel을, 디폴트 부트스트랩 서버로는 /mach_servers/bootstrap을 사용한다. 만약 시스템 관리자로 부터 입력 있으면, 해당 마이크로 커널과 부트스트랩 서버를 적재한다. 마이크로 커널 또는 부트스트랩 서버 적재를 실패하면 시스템 관리자로 부터의 입력이 있는가 다시 확인한다. 성공한 경우에는 각종 부트 정보를 스택에 쌓아 마이크로 커널 시작으로 제어를 옮겨 마이크로 커널을 실행시킨다. 스택에 쌓아 넘기는 부트 정보는 〈표 1〉과 같다.

〈표 1〉 스택 자료구조
〈Table 1〉 stack structure

-1 : 새로운 부트 방식을 나타내는 플래그
long extmem : 확장 메모리 크기
long cnvmem : 기본 메모리 크기
long kern_sym_start : 커널 심볼 시작 주소
long kern_sym_size : 커널 심볼 크기
long kern_args_start : 커널 인수 리스트시작
long kern_args_size : 커널 인수 리스트크기
long boot_sym_start : 부트스트랩 심볼시작
long boot_sym_size : 부트스트랩 심볼크기
long boot_args_start : 부트스트랩 인수 시작
long boot_args_size : 부트스트랩 인수 크기
long boot_start : 부트스트랩 서버 시작
long boot_size : 부트스트랩 서버 크기
long boot_region_desc : 부트스트랩 reg-desc
long boot_region_count : 부트스트랩 reg 개수
long boot_thread_state_flavor : 부트 쓰레드 상태
long boot_thread_state : 부트쓰레드컨텍스트
long boot_thread_state_count : 부트컨텍스트 크기
long env_start : 환경변수 리스트 시작
long env_size : 환경변수 리스트 크기
long top_addr : 커널/부트스트랩 적재된 주소

이렇게 시작한 주처리기는 먼저 부트 정보를 읽어 각 변수들을 세트하고 그에 따라 시스템 및 메모리를 초기화 하고 난 후에 부처리기들의 수행을 시작시키는 것이다. 여기에서 노드는 최대 4개까지의 처리기가 장착되어 있으며, 시스템 콘솔이 장착되어 있어 통신 접속노드 기능을 하며 PCI 슬롯에 디스크를 접속하여 입출력 노드의 기능까지를 모두 한다.

노드 내에 장착된 처리기들 중 주처리기는 시스템 하드웨어에 의해 임의로 선정되게 되며 물리적인 위치와는 관계가 없다. 커널 적재를 위한 시스템 부트 수행 이후 주처리기는 본격적인 마이크로 커널 초기화 과정을 수행하게 되며 다른 처리기들(이하 부처리기들)을 깨우는 작업도 포함된다.

3.1.4 펌웨어 함수 호출

SPAX 펌웨어는 Power on 및 리셋 이후에 초기화 과정에서 모든 자원의 메모리 접근 방식을 위해 보호 모드로 동작하도록 되어 있다. 그러므로 SPAX 펌웨어에서는 부팅 과정에서 사용하는 터미널 입출력, 디스크 입출력, 노드간 메시지 송수신 등을 위한 기능을 간접 호출에 의해 제공하도록 하였다. 그래서 OSF/1 AD3 MISIX의 프라이머리 부트와 세컨더리 부트 코드는 이러한 간접 호출 처리를 하는 펌웨어 함수 호출을 사용한다. 펌웨어 함수 호출을 사용할 때는 메모리 0x200 번지에 있는 펌웨어 함 수 호출 엔트리 포인트들을 call 하여 그 기능을 수행하도록 하는데 펌웨어 제공하는 기능 중 기본적인 몇 가지 기능만 사용한다. 제공하는 함수 호출의 배열 중에 먼저 0x208 번지에 있는 기본 메모리 크기를 얻는 함수, 0x20c 번지에 있는 디스크 읽기/쓰기를 위한 함수, 0x210 번지에 있는 콘솔(COM1 포트) 터미널 동작을 위한 함수, 0x214 번지에 있는 확장 메모리 크기를 얻는 함수, 0x228 번지에 있는 클락 카운트 세트 또는 읽기를 위한 함수 등을 사용한다.

부트 프로그램이 펌웨어 함수 호출을 사용할 때는 다음과 같이 한다. 먼저 함수 호출에 사용될 인수를 차례로 스택에 저장(push)한다. 각 함수 호출에 따라 사용되는 인수의 수가 다르기 때문에 스택에 쌓는 개수도 달라진다. 해당되는 함수 호출 엔트리 포인트를 호출한다. 예를 들면 기본 메모리 크기를 구하는 함수인 경우에는 "call *0x208"을 한다. 처리가 끝났으면 스택에 저장한 크기만큼 클리어하기 위해 스택 포인터를 조절한다.

3.2 초기화 단계

SPAX 시스템의 초기화 단계는 크게 커널 주소 공간 구축과 커널 타스크 실행 환경 구축을 위한 처리기 초기화, 디바이스 초기화 그리고 다른 부처리기들의 실행 관리 등으로 나눈다.

3.2.1 커널 주소 공간 구축

(1) 지역 메모리 초기화

SPAX 시스템은 각 노드당 최대 1기가 바이트 크기의 메모리가 장착된다. 모든 노드의 메모리가 합쳐져서 시스템 주기억장치를 구성하며 이러한 관점에서 한 노드의 물리 메모리를 지역 메모리라 하며 관리한다.

지역 메모리는 페이지 단위로 나뉘어 관리되며 메모리가 요구될 때 페이지를 할당하고 반환하는 동작에 의해 관리된다. 마이크로 커널 초기화 시 시스템 부트로부터 지역 메모리 사용에 관련한 데이터를 넘겨 받아

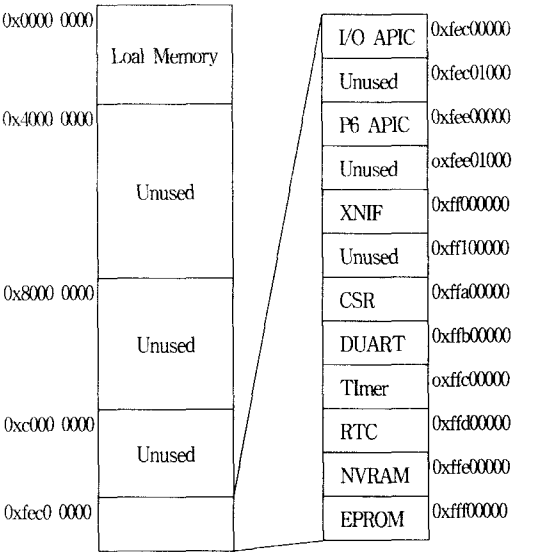
이를 이용, 지역 메모리들의 가용 풀을 구성한다. 메모리 관리와 관련하여 부트로 부터 마이크로 커널에 전달되는 정보는 지역 메모리 중 기본 메모리 크기, 확장 메모리 크기, 마이크로 커널과 부트스트랩 서버가 적재된 메모리의 top 주소 등이 있다. 여기에서 MMU가 동작되기 전에 마이크로 커널과 부트스트랩 서버가 적재되어 사용되는 물리 메모리에 대응되는 가상 주소를 매핑시키기 위해 페이지 디렉토리 테이블과 페이지 테이블 엔트리 테이블을 초기화시킨다. 이 때 중요한 것은 SPAX에서 사용하는 특정 하드웨어는 메모리 지정 방식(Memory mapped Access)을 사용하므로 이러한 주소 영역을 커널 가상 주소 공간에 페이지 디렉토리와 테이블을 설정하여 물리 주소-대-가상주소가 동일하게 사상이 되도록 초기화하였다.

커널 주소 공간 구축은 커널 내에서 크게 두 번 이루어진다. 한번은 커널이 MMU를 실행시키기 전에 커널이 사용하는 텍스트와 데이터 영역만을 실제 주소와 가상 주소로 일대일 매핑시키는 것이다. 이때는 MMU 동작시키고 나서 커널의 각종 데이터구조를 초기화 할 때 MMU가 동작하는 가상 주소 모드에서 초기화하기 위해 커널 텍스트와 데이터만 초기화 하는 것이다. 그렇다 하더라도 초기화를 하려면 각종 인터럽트 및 콘솔을 사용하기 위한 하드웨어 레지스터들을 엑세스 하여야 하므로 하드웨어들도 가상 주소와 매핑을 하여야 한다. 다만 가상 주소와 물리 주소가 동일하게 하도록 한다. 두 번째는 현재 사용한 메모리 이후 사용 가능한 메모리까지 물리 주소와 가상 주소의 매핑을 하여 가용한 메모리 모두의 커널 주소 공간을 구축한다. 이 때도 하드웨어 레지스터들의 주소 매핑이 이루어진다.

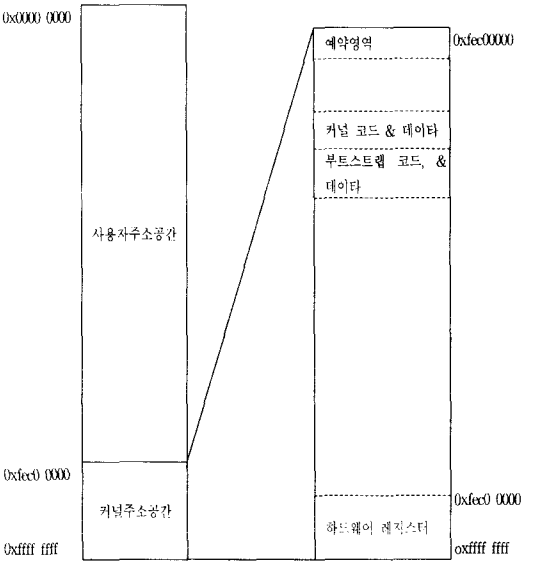
(2) 커널 가상 주소 공간 관리

SPAX의 물리 메모리 공간은 하드웨어에 의해 결정되며, 아래 (그림 4)와 같다. 물리 메모리 공간은 각 노드가 가질 수 있는 최대 지역 메모리 크기인 1기가 만큼을 사용하며, 실제 하드웨어 레지스터들을 참조할 때 사용하는 하드웨어 주소를 0xfec00000 이후에 매핑시킨다. 하드웨어 레지스터들은 I/O APIC, P6 Local APIC, XNIF, CSR, DUART, Timer, RTC, NVRAM, EPROM을 위한 것들이 사용된다.

커널 주소 공간 구축에 의해 만들어진 결과로 (그림 5)의 가상 주소 공간을 엑세스 하면 (그림 4)의 물리 주소 공간에 내용을 엑세스할 수 있다. MMU가 동작



(그림 4) SPAX의 물리 주소 공간
(Fig. 4) Physical address space of SPAX



(그림 5) 커널 가상 주소 공간
(Fig. 5) Kernel Virtual Address space

되어 실행되면 지역 메모리 초기화에서 구축된 가상 주소 공간에 물리 주소 공간이 매핑이 되게 한다. 예를 들자면 MMU는 CR3 레지스터에 저장된 페이지 디렉토리 테이블의 기본 주소(base address)를 통해 커널 가

상 주소 영역인 0xc0000000을 엑세스 하면 지역 메모리 초기화 시에 구축된 페이지 디렉토리 테이블과 페이지 테이블의 값을 이용해 물리 주소 0x0을 액세스할 수 있도록 한다. 즉 MMU를 통한 2단계 주소 변환으로 0x0를 엑세스할 수 있도록 초기화된 각 페이지 디렉토리 테이블과 페이지 테이블 엔트리들을 이용한다는 것이다.

SPAX 시스템에서 사용되는 각 하드웨어 들은 각 레지스터 별로 초기화가 이루어져서 실제로 사용된다. I/O APIC의 경우에는 SCSI 디바이스에 종류에 따라 인터럽트 및 입.출력 주소를 초기화하여 사용하고, P6 Local APIC의 경우에는 각 처리기의 지역 타이머 초기화 및 부처리기 활성화 시에 사용된다. XNIF는 다른 노드로 나가는 외부 인터페이스로, CSR은 SPAX 시스템 형상 관련한 정보 레지스터로, DUART의 경우는 콘솔 디바이스 관련 레지스터로, 타이머는 전역 타이머 레지스터로, RTC는 실시간 클락 관련 레지스터로, NVRAM은 비휘발성 메모리 레지스터로, EPROM은 프로그램 가능한 ROM 메모리 레지스터로 사용하도록 각각 초기화한다.

위의 물리 메모리 공간에 대한 가상 주소 공간의 맵은 운영체제 수준에서 결정되며 아래 (그림 5)와 같이 사용된다. 4 기가 바이트의 가상 주소 공간 중 처음 3 기가 바이트가 사용자 가상 주소 공간으로, 나머지 1 기가 바이트가 커널 주소 공간으로 할당된다.

3.2.2 처리기 초기화

(1) 커널 타스크 실행 환경 구축

커널 타스크가 실행되기 위해서는 커널이 시스템 전체에 걸쳐 사용하는 전역 디스크립터 테이블(Global Descriptor Table)과 인터럽트 처리를 위한 인터럽트 디스크립터 테이블(Interrupt Descriptor Table), 사용자가 사용할 지역 디스크립터 테이블(Local Descriptor Table)이 필요하다. 전역 디스크립터 테이블의 내용은 커널이 사용할 커널 코드에 대한 정보가 있는 코드 세그먼트, 데이터에 대한 정보가 있는 데이터 세그먼트, 지역 사용자가 사용할 지역 디스크립터 테이블, 프로세서의 타스크 상태 정보가 있는 타스크 상태 세그먼트(Task State Segment) 등이 있어 커널이 실행하면서 사용하도록 한다. 전역 디스크립터 테이블에 있는 지역 디스크립터 테이블은 사용자 프로세스가 사용자 모드에서 사용하는 것으로 시스템 호출이나 마이크로 커널 내에서의 RPC를 위한 정보와 사용자 코드 세그먼트, 사용자 데이터 세그먼트 등으로 구성된다. 커널 코드 세그먼트와 커널 데이터 세그먼트의 영역도 하드웨어 레지스터 사용에 따라 SPAX 시스템에 알맞게 그 범위가 설정되었다. 인터럽트 디스크립터 테이블은 커널에서 발생되는 모든 예외(Exception)나 인터럽트 처리에 사용되는 테이블로 예를 들자면 버스 에러나 더블 폴트 에러에 처리하여 줄 루틴 정보를 가지고 있어 그러한 상황이 발생하였을 때 그것을 처리하도록 한다. 운영체제가 실행되기 위해 작성된 테이블(GDT와 IDT, LDT)의 내용은 운영체제 작성자가 만든 포맷에서 주처리기가 실행 시에 쓰는 포맷으로 변경하는 과정이 필요하다. 변경하는 루틴 내부에서 각 테이블의 주소를 참조할 때 보호모드의 참조로 변경한 후에 GDT와 IDT를 각각 GDTR과 IDTR에 적재한다.

페이지 관련 모든 테이블을 SPAX에 알맞게 모두 초기화 한 다음에 제어 레지스터인 CR0내의 PG(paging) 비트를 세트한다. PG 비트가 세트 된 후에는 발생되는 모든 주소가 MMU를 거쳐서 물리적 메모리에 접근되므로 페이지 관련 테이블 중 어떤 것 하나라도 제대로 되어 있지 않으면 시스템에 문제가 발생된다. MMU는 세그먼트/페이징 수준의 주소 번역을 사용하며 논리적 주소에서의 물리적 주소로의 주소 번역은 논리적 주소에서 선형 주소로, 선형 주소에서 물리적 주소로 변환하는 두 단계로 이루어진다. 논리적 주소는 세그먼트의 인덱스인 셀렉터와 옵셋으로 구성된 주소를 말하며 선형 주소는 디렉토리, 테이블, 옵셋으로 구성된 주소이다. PG 비트가 세트 된 후에는 MMU가 동작되기 때문에 이미 가져왔거나(fetch), 해독된(decode) 다음 인스트럭션은 플러쉬되어야 한다.

(2) CSR 초기화

각 처리기의 정보를 machine_slot이라는 자료구조에 저장하여 사용한다. SPAX 구성의 특성 상 처리기는 노드 번호, 노드 타입, 처리기 논리 번호의 정보를 담은 변수를 처리기별로 관리한다. getCSR() 함수는 CSR_ADDR (0xff900000)의 CSR 레지스터를 읽어 노드 ID, 노드 타입 등을 반환하도록 한다. 초기화 시에 버스 종류(busType) 및 시스템 종류(machine type)는 SPAX에서 고정된 값을 가지므로 직접 지정하며 처리기 개수(cpu_number), 노드 ID (nodeId),

노드 종류(nodeType) 만 수행 처리기에 해당하는 값이 지정되도록 한다.

(3) 주처리기 타입 및 FPU 초기화

주처리기의 타입과 FPU 존재 유무를 결정해서 주처리기 변수와 FPU 관련 변수 및 제어 레지스터를 초기화한다. 원천코드에서는 EFLAGS 레지스터의 AC(alignment check) 플래그 비트를 이용하여 386인지 486인지를 구별한다. 486 이후에는 EFLAGS 레지스터의 ID (IDentification) 플래그 비트에 의해 cpuid 인스트럭션이 사용가능 한지를 알 수 있다. cpuid 인스트럭션을 사용하여 486, 586, 686 임을 확인한 후에 CPU 관련 family 정보, vendor id, feature 플래그, Stepping, model 변수에 그 값들을 세트한다. SPAX는 P6 처리기를 사용하므로 SPAX 처리기에 알맞게 CPU 타입(CPUID_FAMILY_PRO), 모델, frequency, 처리기에서 제공하는 각종 feature 등을 설정한다. 또한 P6의 경우에는 처리기에 내장된 FPU를 사용하면 되므로 제어 레지스터인 CR0의 TS 비트와 EM 비트를 클리어한다.

(4) Cache 초기화

처리기가 486 이상인 경우에는 캐쉬를 초기화 하는 함수가 필요하다. SPAX의 처리기는 P6 처리기이므로 그에 알맞게 초기화 한다. 그 함수를 실행하는 시점 또한 주처리기와 부처리기에 따라 달라지므로 각각 이 함수를 호출하여 캐쉬를 실행(enable) 시킨다. 주처리기의 경우에는 MP 관련한 자료구조 및 인터럽트 등을 초기화 한 후에 이 함수를 호출하며, 부처리기인 경우에는 부처리기의 기계 종속부분을 초기화 한 부분에서 캐쉬를 초기화한다. P6 처리기는 기계 종속적인 메모리 타입 영역 레지스터(memory-type range registers)를 사용하여, 메모리 용량 및 영역을 지정할 수 있으므로 SPAX 주소 공간에 알맞게 메모리 베이스 주소 및 가변 영역을 설정한다. 또한 P6 처리기에 알맞은 인스트럭션(wbinvd, rdmsr, wrmsr)과 CR0 및 CR4 레지스터를 이용하여 캐쉬를 실행시킨다.

3.2.3 디바이스 초기화

OSF/1 AD3 MISIX는 각종 디바이스 관련한 구동기들을 마이크로 커널 내에 가지고 있다. 입출력 버스 또한 ISA, EISA 및 PCI 버스를 지원하며, 커널 생성시에 옵션을 주어 각종 디바이스를 구동하게 한다. 지원하는 구동기는 여러 가지 이지만, SPAX 시스템의 입출력 버스인 PCI 버스에 관련된 설계 대상의 디바이스만을 기술한다. 노드가 가지는 PCI 슬롯은 네 개로 보통은 디스크를 연결하는 SCSI 보드와 외부 네트워크를 연결하는 네트워크 보드를 접속하여 사용한다. 지원하는 SCSI 보드는 On-board 인 NCR/Symbios PCI Adaptors 53C810, NCR/ Symbios PCI Adaptors 53C825 이며, 지원하는 네트워크 보드는 DEC Etherworks PCI Ethernet card(DE435-AA) 이다.

SPAX 시스템은 네트워크와 디스크 디바이스를 부착할 수 있도록 4개의 PCI 슬롯을 제공하고 있다. 먼저 PCI용 입출력 디바이스를 사용하기 위해 4개의 슬롯에 각각 하나씩 인터럽트를 할당한다. PCI용 입출력 디바이스를 사용하기 위한 구동기는 마이크로 커널 내에 구현되어 있으나 이러한 입출력 구동기들이 제대로 동작할 수 있도록 각 구동기가 초기화되기 전에 PCI 소자들을 SPAX에 맞게 초기화하여야 한다. SPAX 하드웨어는 PCI 형상 구성 메커니즘 중에서 0xcf8번지를 엑세스 하는 형상 메커니즘만을 제공하고 있다. 따라서 커널에서도 그 메커니즘에 따라서 PCI 소자를 초기화한다. 참고로 PCI 형상 구성 메커니즘은 초기화 시에는 PCI 형상 레지스터에 각 슬롯별로 사용할 주소 공간과 인터럽트 번호를 설정하여 놓는다. 각 입출력 구동기들의 초기화 모듈에서는 PCI 초기화 시에 슬롯에 설정된 값을 읽어 입출력 디바이스를 커널이 사용할 수 있도록 하여 준다.

3.2.4 부처리기 활성 관리

SPAX 시스템에서는 한 노드에 4개의 처리기가 장착되어서 마이크로 커널 실행 시에는 주처리기인 한 개의 부트 처리기만이 실행된다. 이 부트 처리기는 노드 내 초기화를 실행한 후에 나머지 부처리기들을 깨운다. SPAX 시스템에 알맞게 P6 Local APIC 하드웨어 레지스터들을 이용하여 직접 깨워주는 방식을 사용한다. 부처리기가 실행을 시작하는 주소를 물리적 주소 MP_BOOT(0x1000)로 하고, 그 곳에 부처리기 실행 코드를 복사한 후에 부처리기를 깨운다. 이 때 부처리기의 실제 시작 인 pstart를 설정해주고, 주처리기의 페이지 디렉토리 및 페이지 테이블 엔트리들을 복사하여 부처리기들이 사용할 수 있도록 한다. 운영체제 목적 코드의 구성 시 부처리기 코드는 slave_boot_ base

라는 심볼로 시작되어 물리 주소 0x1000에 복사된다. 이 부처리기 실행 코드는 실제 모드의 코드이다.

부처리기들은 펌웨어 모드에서 대기 상태로 있다가, P6 Local APIC을 통해 처리기 활성(startup) 인터럽트를 받아 해당 주소로 제어를 옮겨 해당 주소에 복사된 부처리기 코드를 수행한다.

부처리기 실행 코드의 내용은 먼저 실제 모드의 상태에서 데이터 세그먼트, 스택 세그먼트 등을 초기화하여 플랫 모드의 보호 모드로 바꾼다. 그런 후에 그 보호 모드에서 가상 주소 pstart로 제어를 옮겨 실행한다. 여기에서 커널 시작 주소가 같아 주처리기와 부처리기가 수행하는 코드이지만, 주처리기인 경우에 실행하면서 master_is_up 이라는 플래그를 세트하여서 주처리기만 수행해야 하는 코드를 구분하여 실행하도록 한다. 그러므로 이 과정에서 부처리기의 지역 메모리 초기화와 커널 가상 주소 구축이 일어난다. 즉 부처리기 또한 SPAX의 하드웨어의 기능을 사용하여야 하므로 이러한 초기화 과정이 필요하다.

3.3 초기화 기법

먼저 부트 단계에서의 다른 기법 중에 하나는 SPAX의 부트 섹터 프로그램으로 다른 운영체제와는 다르게 실제 모드에서 실행되지 않고 보호 모드에서 실행된다. SPAX 펌웨어 실행 모드가 보호 모드이기 때문에 실제 모드의 부트 섹터 코드를 실행하려면, 모드 변환의 오버헤드가 따르므로 보호 모드의 실행 코드로 구현하였다. 다른 기법으로는 기존의 운영체제와는 다르게 새로 구현된 펌웨어 함수 호출을 사용하므로, 보호 모드에서 실제 모드로, 실제 모드에서 보호 모드로의 변환에 대한 오버헤드를 줄이는 장점이 있다. 또한 각 함수 별로 인수의 레지스터 변수들을 사용하는데 주의를 기울이지 않고 프로그램을 작성할 수 있어, 프로그래밍 하기가 쉬운 장점도 있다. 그에 따라 프라이머리 부트와 세컨더리 부트 과정에서 필요한 티미널 입.출력, 디스크 입.출력, 메모리 관련 처리 등을 위한 모든 코드들이 펌웨어 함수 호출 방식으로 SPAX에 알맞게 구현되었다. 그 이외에도 부트 과정에서 주소 영역을 보호 모드의 가상 주소를 쓰지 않고 보호 모드이지만 실제 주소를 사용하는데, SPAX 시스템에 알맞게 펌웨어 예약 영역을 사용할 수 있도록 주소 공간을 새로 설정하였다.

초기화 단계에서의 다른 기법은 커널 주소 공간 구축 중 지역 메모리 초기화의 경우이다. 기존의 운영체제 초기화와는 다르게 초기화 이후에 하드웨어의 커널 가상 주소를 엑세스 하였을 때, 메모리가 아닌 하드웨어 레지스터를 엑세스 할 수 있도록 매핑을 시켜주는 것이 그 특징이다. 만약 이러한 지역 메모리 초기화가 없으면, 각 하드웨어 주소를 엑세스 하여 그 기능을 사용할 수 가 없다. 그러므로 SPAX에 알맞게 지역 메모리 초기화를 설계 및 구현하여, 메모리 지정 방식의 하드웨어 레지스터 입.출력으로 각 하드웨어의 각종 기능을 쉽게 이용할 수 있도록 하였다. 또한 커널 주소 공간 구축 중 커널 가상 주소 공간에서는 각 하드웨어 별로 해당 메모리를 입.출력 하는 것으로 하드웨어의 기능을 이용한다. 이 때의 입.출력도 SPAX 하드웨어 레지스터의 구성에 따라 초기화 하고 사용하도록 설계 및 구현하였다. 예를 들자면 SCSI 장치의 입출력은 I/O APIC 레지스터의 주소 접근에 의해, 처리기 지역 타이머 및 부처리기 활성은 P6 Local APIC 레지스터의 주소 접근에 의해, 콘솔 드라이버 처리는 DUART 레지스터 접근에 의해 해당 하드웨어 처리를 할 수 있다. 그 이외에도 CSR, Timer, NVRAM 등도 SPAX 시스템에 알맞게 처리하도록 구현하였다. 마지막으로 초기화 단계의 처리기 초기화, 디바이스 초기화, 부처리기 활성화의 거의 모든 초기화 부분들이 SPAX 시스템에 알맞게 새로 설계 및 구현되었다.

4. 테스트 환경 구성

여기에서 테스트 환경은 현재 가용한 고속병렬컴퓨터 시스템인 4개의 P6 처리기가 장착되고 32 메가 바이트의 메모리를 갖는 단일 노드 시스템 환경으로 부트 및 마이크로 커널 초기화 테스트를 완료하였다.

테스트 결과, 부트 시에 마이크로 커널 및 서버들이 메모리에 알맞게 적재되었음을 확인할 수 있었으며, 마이크로 커널의 초기화에 의해 OSF/1 AD3 MISIX가 정상적으로 동작함을 확인할 수가 있었다. 부팅 하는데 소요되는 시간은 단일 사용자 모드로의 부팅은 약 10초 정도이며, 다중 사용자 모드인 경우에는 약 1 분 이내에 이루어진다. 이 소요 시간은 사용자가 디폴트 세컨더리 부트 프로그램과 마이크로 커널 그리고 부트스트랩 서버를 사용하였을 때의 경우이며, 만약 그렇지 않은 경우에는 입력을 기다리는 시간인 30 초와 입력시간 등에 따라 달라진다. 또한 다중 사용자 모드의 경우에

는 파일 시스템을 마운트하기 전에 fsck 명령어를 이용하여 파일 시스템을 확인하여야 하므로 다중 사용자 모드에서 마운트 되는 파일 시스템의 크기에 따라 시간이 걸린다.

SPAX 시스템의 구조상으로는 컴퓨팅 작업 및 유닉스 환경을 제공하는 프로세싱 노드와 네트워크를 위한 통신 접속 노드, 입출력 작업을 주로 하는 입.출력 노드가 있어서 각각 그 기능을 테스트하여야 하지만 테스트 환경 상의 이유로 단일 노드에서 그 모든 기능을 테스트하여야 하였다.

또한 시스템의 시험을 위하여 상용 작업 부하 벤치마크 프로그램 중에 하나인 AIM III를 사용하였다. 이 벤치 마크 프로그램은 AIM사에서 공급한 suite로 전형적인 유닉스 시스템 호출과 사용자 모드 동작을 섞어 골고루 실행되게 한다. AIM III는 다중 사용자의 작업 부하를 simulate하여 결과를 얻는다. 각 사용자는 고정된 양의 일을 하며, 동시에 simulate되는 사용자의 수도 지정할 수 있다. 성능 분석을 위하여 작업부하를 끝마친 모든 simulate 사용자들에게 요구되는 실제 시간과 그 처리율에 관련된 벤치마크 출력을 알 수 있다. 현재는 약 80 사용자까지 simulation이 가능하였다.

5. 결론 및 향후 연구 방향

본 논문에서는 고속병렬컴퓨터에 알맞게 설계 및 구현된 OSF/1 AD3 MISIX의 마이크로커널의 초기화 기능을 기술하고 그 시험 내용을 기술하였다. 먼저 부트처리기가 속한 노드의 부트를 실행하고 각 노드 내에서 마이크로 커널 초기화를 실행한다. 부트는 부트 전반적인 처리 및 프라이머리 부트, 세컨더리 부트 단계 등이다. OSF/1 AD3 MISIX의 부트는 다른 시스템 부트와는 다르게 펌웨어 모니터 모드부터 보호 모드의 실행이므로 모두 보호 모드로 동작한다. 또한 UFS 파일의 세컨더리 부트 프로그램 적재로 여러 목적 코드를 지원하거나 프로그램 위치에 무관하게 지정이 가능하여 부트 파일의 높은 유연성(Flexibility)을 제공하며, 펌웨어 함수 호출로 인한 모드 변환의 오버헤드를 줄이므로 부트의 효율성을 제공한다. 마이크로 커널 초기화는 커널 주소 공간 구축, 처리기 초기화, 디바이스 초기화, 부처리기 활성 관리 등이다. 마이크로 커널 초기화는 SPAX 시스템에 알맞게 하드웨어 레지스터 주소 공간을 초기화 하여 메모리 지정 방식의 입.출력으로 그 기능을 이용하기가 용이하도록 하였다. 또한 각종 초기화에서 SPAX 시스템에 형상 및 처리기에 알맞게 초기화를 하여 시스템이 원활하게 동작하도록 하였다.

초기화 시험은 현재 가용한 고속병렬컴퓨터 시스템인 4개의 P6 처리기가 장착되고 32 메가 바이트의 메모리를 갖는 단일 노드 시스템에서 하였으므로 노드간 부트 및 분산 환경에서의 초기화 테스트는 하지 못했다. 가능하다면 향후에 여러 개의 노드를 연결하여 부트 및 초기화를 시험하겠다. 그리고 비록 노드간 테스트는 하지 못했지만, 분산 IPC를 테스트하기 위해 현재 가용한 두 개의 노드를 X-cent 대신 SCSI로 클러스터 하여 테스트 할 수 있었다. 이 테스트는 노드간 부트 및 초기화 테스트 시에 많은 도움이 될 것이다. 또한 가능하다면 OSF에서 만들어진 마이크로 커널 성능 측정 유틸리티를 이용하여 성능을 측정 할 것이다.

참 고 문 헌

[1] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A New Kernel Foundation for Unix Development", *Proceedings of the USENIX Summer'86 Conference*, July 1986.

[2] Y. W. Kim, S. W. Oh and J. W. Park, "Design Issues and System Architecture of TICOM IV, A highly parallel commercial computer," The 3rd Euromicro Workshop on Parallel & Distributed Processing, 1995.

[3] Brian Bershad, "The Increasing Irrelevance o f IPC Performance for Micro-kernel-based Operating Systems", Proceedings of *USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pp. 205-212, April 1992.

[4] Michael Condict, Don Bolinger, Dave Mitch -ell, and Eamonn McManus, "Microkernel Modularity with Integrated Kernel Performa -nce", *Mach/Chorus Workshop at the First USENIX OSDI Symposium*, October 1994.

[5] Jochen Liedtke, "Improving IPC by Kernel Design", *Proceedings of the 14th ACM Sym -posium on Operating Systems Principles*,

pp. 175-188, December 1993.

[6] 김해진, 임기욱, "MISIX: 고속병렬컴퓨터(주전산기 VI) 병렬 운영체제", UNIEXPO'95 논문지, pp.45-49, 1995년 11월.

[7] Randall W. Dean, "Using Continuations to Build a User-Level Threads Library", *Proceedings of the USENIX Mach III Symposium*, pp. 137-152, April 1993.

[8] Bryan Ford, Mike Hibler, and Jay Lepreau, *"Notes on Thread Models in Mach 3.0, University of Utah Computer Science Dept."*, TR UUCS-93- 012, April 1993.

[9] Bryan Ford and Jay Lepreau, *"Evolving Mach 3.0 to Use Migrating Threads"*, University of Utah Computer Science Dept., TR UUCS-93-022, November 1993.

[10] Jay Lepreau, Mike Hibler, Bryan Ford, and Jeff Law, "In-Kernel Servers on Mach 3.0: Implementation and Performance", *Proceedings of the USENIX Mach III Symposium*, pp. 39-55, April 1993.

[11] Simon Patience, "Redirecting System Calls in Mach3.0, an Alternative to the Emulator", *Proceedings of the USENIX Mach III Symp -osium*, pp. 57-74, 1993.

[12] Intel Corp., "Pentium Pro Family Developer's Manual(Vol. 2)", 1996.

김 정 녀

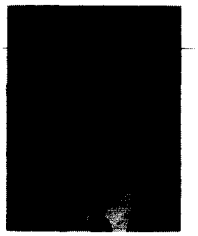
1987년 전남대학교 계산통계 학과 졸업(학사)

1995년~1996년 Open Software Foundation Research Institute 공동 연구 파견(미국)

1998년~현재 충남대학교 컴퓨터공학과 석사과정

1988년~현재 한국전자통신연구원 시스템 S/W 연구실(선임연구원)

관심분야 : 운영체제, 분산 처리, 고장 감내



조 일 연

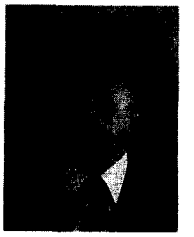
1991년 성균관대학교 산업공학과 (공학사)

1993년 성균관대학교 산업공학과 (공학석사)

1993년~현재 한국전자통신연구원 연구원(컴퓨터연구단 시스템연구부)

1995년~1996년 미국, OSF RI(Open Software Foundation Research Institute)파견 근무

관심분야 : 병렬운영체제, 성능평가



이 재 경

1981년 경북대학교 전자공학과 (공학사)

1997년 경북대학교 전자공학과 (공학사)

1997년~현재 충남대학교 컴퓨터공학과 석사과정

1983년~1984년 (주)금성반도체 사내 전산과 근무

1985년~현재 한국전자통신연구원 책임연구원(컴퓨터연구단 시스템연구부)

관심분야 : 병렬운영체제, 펌웨어, 병렬프로그래밍

김 해 진

1983년 경북대학교 컴퓨터공학과 (학사)

1995년 충남대학교 대학원 전산학과 (석사)

1992년 정보처리기술사 (전자계산조직응용)

1983년~현재 한국전자통신연구원 시스템 S/W 연구실장(책임연구원)

관심분야 : 운영체제, 병렬 처리, 실시간 처리, 고장 감내