2001 년 한국정보처리학회 춘계 학술발표논문집 제 8 권 제 1 호

# M3K 와 Fiasco 의 IPC 성능 비교 및 분석

아재용*, 고영웅*, 유 혁*
*고려대학교 컴퓨터학과
e-mail : jyah@os.korea.ac.kr

# Comparison and Analysis of IPC Performance between M3K and Fiasco

Jae-Yong Ah*, Young-Woong Ko*, Hyuck Yoo*
*Dept. of Computer Science, Korea University

**요 약**

마이크로커널 구조에 있어서 IPC 의 오버헤드는 전반적인 운영체제의 성능을 결정짓는 주요한 요소가 된다. 본 논문에서는 M3K 마이크로 커널에서의 IPC 지연 시간을 명확히 측정하기 위해서 IPC 수행 경로를 몇 단계 구간으로 나누었으며, 각 구간에서 사용되는 코드들의 사용빈도와 수행시간을 정확히 측정함으로써 지연이 발생되는 부분을 세밀하게 분석하였다. 또한 완전히 구현된 Fiasco 마이크로커널의 IPC 지연 시간을 M3K 에서 적용한 구간별로 측정하고 상호 비교하였다. 연구 결과 IPC 의 성능 향상을 위해서 Fiasco 에서 적용하고 있는 하드웨어의 특성을 활용한 구현이 실제로는 M3K 마이크로커널의 하드웨어 독립적인 IPC 구현에 비해서 큰 장점이 없음을 보여주고 있다. 오히려, 소프트웨어적인 최적화가 더많은 성능상의 이점을 줄 수 있다는 것을 실험을 통해서 보여줄 수 있었다

## 1. 서론

마이크로커널은 운영체제를 경량화 함으로써 커널의 복잡성을 줄이고 다양한 운영체제의 기능성을 사용자 영역에서 서버로 제공하는 아이디어를 제시하고 있다. 따라서 마이크로커널은 실행 객체들간의 통신 프리미티브 그리고 주소 공간 객체등과 같은 핵심적인 기능만으로 구성되며, 그 위에서 운영체제가 필요로 하는 대부분의 서비스들, 예를 들어, 메모리 관리, 파일 시스템, 네트워크 서비스, 기타 디바이스 드라이버 등이 수행된다.

기존의 모노리틱 커널 구조에 있어서는 운영체제 서비스를 받기 위해서 간단한 트랩 메커니즘을 이용했지만, 마이크로커널 구조에서는 서버에게 IPC(Inter Process Communication) 메커니즘을 통해서 서비스를 요청하는 구조로 바뀌게 된다. 따라서 빈번하게 발생되는 IPC 의 성능은 마이크로 커널을 기반으로 하는 운영체제의 성능에 결정적인 영향을 끼치게 되며, 이러한 오버헤드를 줄이는 문제는 마이크로커널 연구에 있어서 핵심이 되어 왔다.

본 논문에서는 제 2 세대 마이크로커널 구조로 설계된 독일의 Fiasco[1] 마이크로 커널의 IPC 수행 경로를 몇 단계의 구간으로 나누어 세밀하게 수행시간을 측정하고 있다. 측정 결과 Fiasco 의 IPC 수행시간의 대부분은 IPC 메시지의 전달과 처리, 그리고 문맥전환에 의해서 사용되고 있음을 알 수 있었다.

또, 이러한 Fiasco IPC 수행시간 측정을 통해서 본 연구실에서 구현중인 M3K(MultiMedia Microkernel) [2][3][4] 커널에서의 IPC 수행시간과 비교 분석한다. 그리고 이를 통하여 IPC 성능에 큰 영향을 미치는 요소를 파악하고 개선하기 위한 분석을 하고 있다.

본 논문은 다음과 같은 구성으로 되어 있다. 2 장에서는 기존의 마이크로 커널에서 제공하고 있는 IPC 의 성능 향상 관련 연구들을 살펴본다. 3 장에서는 M3K 의

[1] 본 연구는 한국과학재단의 특정기초 연구과제 연구비 지원에 의한 결과임 (과제번호 97-01-00-09-01-3)

Architecture 에 대해서 설명하겠고, 4 장에서는 Fiasco 의 IPC 의 개요를 설명하고, 수행 시간을 몇 단계 구간별로 측정한다. 5 장에서는 M3K 와 Fiasco IPC 와의 성능 비교를 통해 수행시간에 영향을 미치는 요소들을 분석한다. 6 장에서 결론을 맺고, 향후 계획을 언급한다.

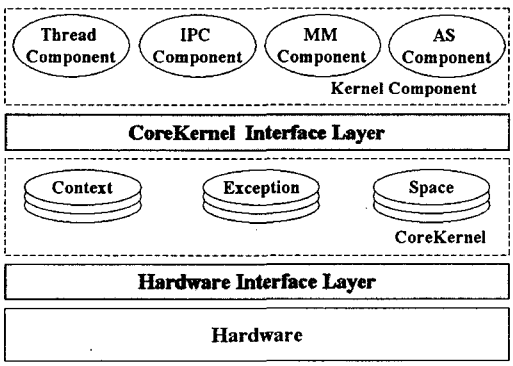
## 2. 관련연구

기존의 마이크로커널 구조의 운영체제에서 IPC 성능 향상을 위해서 많은 연구가 진행되어 왔다. 첫째는 빈번한 IPC 를 요구하는 서버를 커널공간으로 들여오는 방법이다. Mach 3.0[5]과 Amoeba[6]와 같은 마이크로 커널은 서버를 커널 공간 안으로 들여와서 실행시키는 구조이며, IPC 호출 중간에서 발생하게 되는 문맥전환(Context Switching) 오버헤드를 줄여줌으로써, 성능을 향상시키고자 하는 방법이다. 하지만, 이러한 접근 방법은 커널의 크기를 최소화 하고자 하는 마이크로 커널의 기본 개념을 위반하게 된다.

둘째는 IPC 메커니즘을 구현하는 데 있어서 가능한 모든 최적화를 적용하는 것이다. 예를 들면, L4/L3 에서의 알고리즘 및 코딩 단계에서의 최적화, 그리고 하드웨어 아키텍처의 특성을 활용하여 IPC 의 성능을 향상시키는 방법[7]과 같은 접근을 예로 들 수 있겠다.

## 3. M3K Microkernel

M3K 는 멀티미디어를 지원하기 위해 제작된 마이크로 커널로, 그림 1 과 같이 하드웨어를 추상화한 코어커널과 이들이 제공하는 인터페이스를 이용하여 실제 커널 서비스를 제공하는 커널 컴포넌트로 이루어 진다. 이중에 본 논문에서 언급할 IPC 컴포넌트(IPC component)는 커널 컴포넌트에 속해 있으며 다른 커널 컴포넌트와 마찬가지로 하드웨어 독립적인 특성을 지닌다.

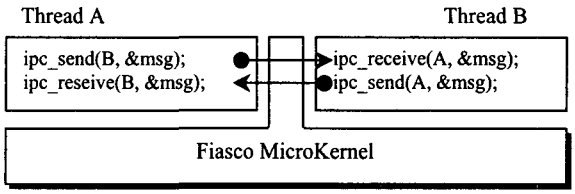



[그림 1] M3K 커널 모델

## 4. Fiasco 의 IPC 구간별 측정 및 분석

### 4.1. Fiasco IPC

Fiasco 에서의 IPC 는 메시지 패싱(message passing) 방식으로 수행되는 IPC 이며, IPC 에 참여하는 쓰레드간에 버퍼를 사용하지 않는 동기식(synchronous) IPC 이다. 또한, 우선 순위에 기반하여 IPC 메시지를 핸들링하여 실시간성을 지원하고자 디자인되었다.




[그림 2] 유저 쓰레드간의 IPC 수행 모델

Fiasco 에서 제공하는 IPC 는 두 가지 타입으로, 전달하고자 하는 메시지의 크기에 따라서 8 바이트 이하면 short IPC, 8 바이트를 초과하면 long IPC 로 구분하고 있다. 측정에 사용되는 IPC 타입은 short IPC 로 두개의 유저 쓰레드를 이용하여 IPC 의 구간별 수행 시간을 측정하다. 그림 2 에서처럼 쓰레드 A 가 IPC send 를 호출하게 되면, 쓰레드 B 는 IPC receive 를 호출하여 메시지를 받고, 바로 쓰레드 B 는 IPC send 를 호출하여 쓰레드 A 가 IPC receive 를 호출하여 다시 메시지를 받을 수 있도록 하였다. 각각의 IPC send 와 receive 는 커널 모드에서의 수행하게 된다.

### 4.2. Fiasco short IPC 의 측정

Fiasco 에서의 short IPC 는 다음과 같은 경로로 수행이 된다.

(1) IPC 인터페이스를 호출한다.
(2) IPC 송신 쓰레드는 IPC 메시지 객체를 생성한다.
(3) IPC 송신 쓰레드는 IPC 수신 쓰레드 객체에게 IPC 메시지를 전달한다.
(4) IPC 수신 쓰레드로 문맥 전환을 한다.
(5) IPC 수신 쓰레드는 이미 도착한 IPC 메시지 객체를 찾는다.
(6) IPC 수신 쓰레드는 IPC 메시지를 핸들링한다.
(7) 송신 쓰레드로 문맥전환한다.
(8) 수신 쓰레드로 문맥이 돌아온 후, 제대로 IPC 가 수행되었는지를 조사한다.

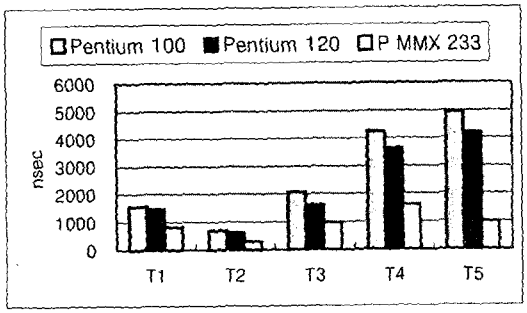
본 논문에서는 위에서 언급한 8 개의 경로를 비교의 편이를 위해서 5 개의 구간으로 나누었다. 구간 1 은 IPC 준비구간이고 (1)+(2)의 수행 시간이다. 구간 2 는 IPC 객체를 수신 쓰레드에게 전달하는 데 걸리는 시간이고 (3)의 수행 시간이다. 구간 3 은 수신 쓰레드에서 메시지 객체를 찾는데 걸리는 시간이고, (5)의 수행 시간이다. 구간 4 는 메시지

객체를 처리하는 시간이고, (6)의 수행하는 시간이다. 구간 5 는 두 번의 문맥전환에 걸리는 시간이고, (4)+(7)+(8)을 수행하는데 걸리는 시간이다. 위의 1 ~ 5 의 구간은 T1 ~ T5 로 표기하여 나타낸다.

표 1 은 각 구간(T1 ~ T5)에 대한 플랫폼별 수행시간을 나타내고 있고, 그림 3 은 표 1 에 대한 그래프이다.

| | Pentium 100 | Pentium 120 | P MMX 233 |
|---|---|---|---|
| T1 | 1550 | 1454 | 817 |
| T2 | 700 | 623 | 300 |
| T3 | 2050 | 1595 | 967 |
| T4 | 4220 | 3632 | 1593 |
| T5 | 4960 | 4186 | 982 |
| Total | 13480 | 11490 | 4659 |

[표 1] 구간별 측정값(nsec)

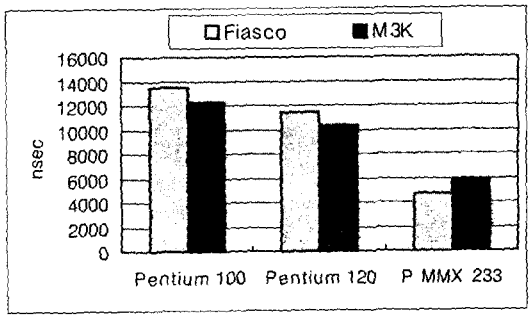



[그림 3] 구간별 성능 측정값

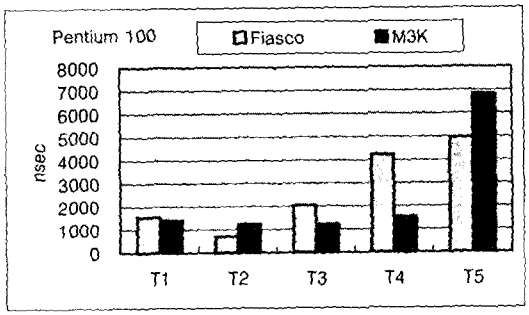
그림 3 를 보면 알 수 있듯이 IPC 수행 중에서 문맥전환에 가장 긴 수행시간이 걸린다는 것을 확인할 수 있다. 이것은 M3K 의 IPC 에 대한 성능 측정에서도 확인할 수 있었던 내용으로 마이크로 커널 기반의 IPC 수행 시간에 가장 큰 영향을 미치는 요소라고 할 수 있다.

### 5. M3K 와 Fiasco 의 IPC 성능비교 및 분석

동일한 플랫폼에서 M3K 와 Fiasco 의 IPC 수행시간에 대한 측정값은 다음과 같다. 그림 4 는 Fiasco 와 M3K 의 전체 IPC 수행시간 값을 보여주고 있고, 그림 5 는 Fiasco 와 M3K IPC 의 구간별 수행시간을 나타내고 있다.

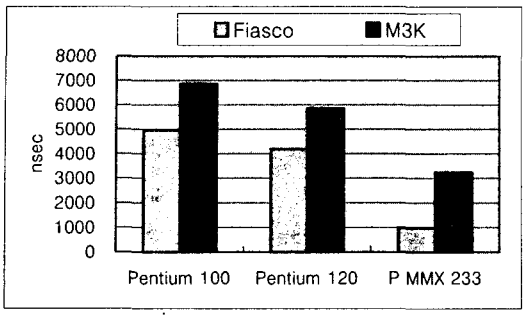



[그림 4] M3K 와 Fiasco IPC 성능 비교




[그림 5] M3K 와 Fiasco 의 구간별 성능 측정값

그림 4 를 보면, T1 과 T2, T3 구간의 경우 거의 차이가 없는 수행시간을 보여주고 있지만, T4 구간의 경우에는 Fiasco 의 수행시간이 M3K 에 비해서 길어진 것을 볼 수 있다. T4 에서 수행 되는 작업은 sender 를 확인하고 관리하는 것과 실제 데이터 복사를 처리하는 부분이다. 이때 IPC 메시지를 처리하기 위해서 Fiasco 의 경우 레지스터를 이용하여 데이터를 복사하기 때문에 메모리 복사를 하는 M3K 보다 더 작은 수행시간이 걸린다. 하지만, sender 를 확인하고 관리하는 부분에 있어서 Fiasco 는 큐를 두고 큐를 통해서 Sender 를 확인하고 관리할 수 있는 반면, M3K 의 경우 메시지의 header 만으로 확인하고 관리하는 방법을 택하기 때문에 상대적으로 단순하고 이것이 더욱 큰 영향을 미치고 있다. 따라서 Fiasco 는 하드웨어적인 최적화를 통한 성능상 이점이 큐 관리의 복잡성에 따른 지연때문에 성능이 좋아지지 않고 있다. 반면에 M3K 는 하드웨어 독립적인 구조를 취하면서 메시지 복사의 시간은 약간 늘어나지만, 큐 관리와 같은 부분에서 단순한 소프트웨어 구조를 택하여 T4 구간의 수행시간은 Fiasco 보다 짧게 걸리고 있다.




[그림 6] 플랫폼에 따른 구간 5 의 비교

T5 의 경우에는 Fiasco 의 수행시간이 M3K 에 비해서 나은 성능을 보여주고 있는데, 이것은 Fiasco 에서의 소프트웨어적인 문맥전환[4]이 M3K 의 하드웨어를 이용한 문맥전환에 비해서 더 좋은 성능을 보여주었기 때문이라 분석된다. 또, 그림 4 를 보면, 하드웨어가 좋아질수록 전체 IPC 를 수행하는데 있어서 Fiasco 의 수행시간이 M3K 에 비해서 더 적게 걸리는데, 이것은 그림 6 에서처럼, 문맥전환에 걸린 시간이 하드웨어가 좋아질수록 더 큰 차이로 Fiasco 가 적게 걸리기 때문에 이러한 결과를 보여주게 된다.

## 6. 결론 및 향후 계획

지금까지 Fiasco 와 M3K IPC 성능에 대해서 상호 비교 분석해 보았다. 비교결과, M3K 의 하드웨어 독립적인 IPC 구조가 보여주는 성능이 Fiasco 에서의 하드웨어적인 특성을 활용한 IPC 구조가 보여주는 성능에 비해서 성능상 큰 저하가 없음을 보여주고 있다. 오히려 소프트웨어적인 최적화와 소프트웨어적인 문맥전환이 IPC 성능에 더 큰 영향을 미치는 것을 알 수 있었다.

향후, M3K IPC 성능 향상을 위해서 하드웨어적인 문맥전환이 아닌 소프트웨어적인 문맥전환을 이용하여 IPC 성능상 가장 큰 비중을 차지하는 문맥전환 오버헤드를 줄이는 연구와 소프트웨어적인 최적화를 통해서 오버헤드를 줄이는 연구를 진행하도록 할 것이다

## 참고문헌

[1] http://os.inf.tu-dresden.de/fiasco/

[2] 양순섭, 고영웅, 조유근, 신현식, 최진영, 유혁, "컴포넌트 기반 커널을 위한 프레임워크" 춘계 정보과학회 학술대회 논문집, 1999

[3] 김영호, 고영웅, 유혁, "M3K 에서 쓰레드 컴포넌트 구현" 추계 정보과학회 학술대회 논문집, 1999

[4] 김영호, 고영웅, 유혁, "M3K 에서 IPC 컴포넌트 설계 및 구현" 추계 정보과학회 학술대회 논문집, 2000

[5] M. Acceta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, M. Young, MACH: A New Kernel Foundation for UNIX Development, In Proceedings of USENIX, summer 1986

[6] A.S. Tanenbaum, M.F. Kaashoek, R.van Renesse, and H. Bal, The Amoeba Distributed Operating System - A Status Report, Computer Communications, vol. 14, pp. 324-335, July/Aug. 1991

[7] Jochen Liedtke, On microkernel construction, In Proceedings of the 15th ACM Symposium Operating System Principles (SOSP) (Copper Moutain Report, Colo., Dec. 1995). ACM Press, New York, 1995. pp 237-250