2000 년 한국정보처리학회 추계 학술발표논문집 제 7 권 제 2 호

회선분배장치 과부하 데몬을 위한 부하분산 방법의 구현과 분석

문형섭, 노봉남
전남대학교 정보통신협동과정
e-mail : hsmoon@lgic.co.kr, bongnam@chonnam.chonnam.ac.kr

An Implementation and Analysis of load Balancing Mechanism for Overloaded Daemon In WDCS

Hyeong-Sub Moon*, Bong-Nam Noh*
*Dept. of Interdisciplinary Program of Information and Telecommunication, Chonnam University

요 약

회선분배장치를 제어관리하는 W/S 의 Daemon 중에서 Event 처리를 담당하는 EvtProc Daemon 의 경우를 들어 효율적인 부하 분산 알고리즘을 설계하였다.
본 논문에서는 대량의 Event 발생 시 담당 프로세스가 연속적으로 fork, exec 하지 않고 일정시간 대기하므로써 fork 와 exec 의 오버헤드를 줄이고 같은 Op Code 의 연이은 Event 발생에 대기중인 프로세스가 이를 처리함으로써 더욱더 오버헤드를 줄일 수 있다. 그리고 Event 를 처리하는 Child 생성에 있어 주 Child 가 부하분산을 담당하게 함으로써 효율적이고 안정적으로 관리할수 있는 방안을 제시하였고 이를 구현 분석하였다

1. 서론

동기식 회선분배 장치는 비동기식 신호인 DS-1, DS-1E, DS-3 신호 또는 동기식 신호인 STM-1 신호를 수용하여 이 신호들을 VC-11, VC-12 또는 VC-3 신호 단위로 회선 교환하고 또, 회선 교환된 이 동기식 고속 다중 신호를 DS-1, DS-1E, DS-3 신호 또는 동기식 신호인 STM-1 신호를 송출하는 기능을 하는 전송장치이다. 이 장치는 Unix OS 를 사용하는 W/S 을 주 제어부로 사용하여 관리와 유지보수에 보다 높은 효율성을 가져다준다.

이 장치의 구성은 하부 i/o Shelf 및 기타 Shelf, 관리 및 감시기능의 admin Shelf 로 구성된다.
admin Shelf 는 Unix 를 운영체제로 사용하는 W/S 으로써 하부 Shelf 들을 제어하는 명령처리부 daemon 과 정보의 저장소를 제공하는 DB, 하부 Shelf 의 상황보고를 처리하는 Event 처리부 daemon 으로 나눈다. 각 dameon 은 처리요구에 대한 담당 프로세스를 생성하여 처리를 이관하고 처리결과에 대한 보고를 받는다.
Shelf 에서 보고되는 주기/비주기적인 상황보고에 대한 Event 처리요구는 순간 대량 발생할 수 있으며 이를 처리하는 daemon 은 프로세스를 생성하고 종료 처리하는데 많은 시간을 할애하게 된다. 이러한 과정은 처리를 기다리는 다른 상황보고 Event 에 대한 막대한 처리지연을 발생시킬 수 있고 이를 처리하는 daemon 에 과부하를 발생 시키게 된다

본 논문에서는 이러한 Event 상황보고 처리요구의 순간 대용량 처리능력 한계극복과 처리지연 해결에 대하여 다룬다

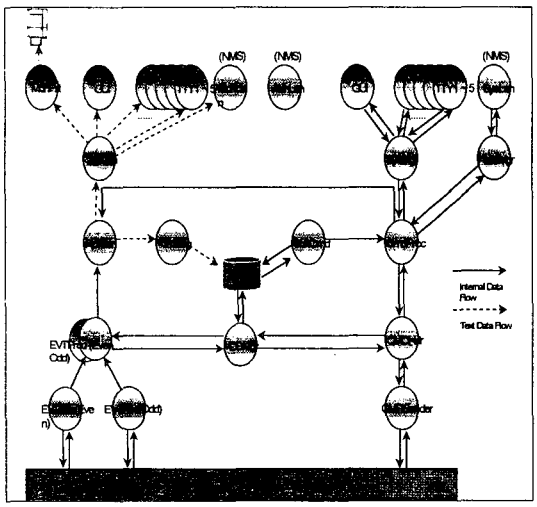
2. 회선분배장치 에서의 Daemon 의 구성

WDCS-8000S 는 Admin Shelf 와 기타 I/O 및 회선상호 연결을 담당하는 Shelf 로 구성 되어 있다. Admin Shelf 는 Unix 운영체제하의 W/S 으로 구성되며 하부 Shelf 관리 어플리케이션은 Unix 프로세스인 dameon 프로세스의 상호 보완적인 구조로 되어 있다.

[그림 1]은 WDCS-8000S 의 daemon 구조도이다. 회선분배장치의 내용을 변경하거나 확인하는 명령을 GUI 로부터 받아 Shelf 에 전달하고 결과를 보고 받는

CmdProc daemon 과 Shelf 에서 보고되는 Alarm/Event 를 처리하는 EvtProc daemon 이 있다 그리고 각 daemon 의 처리내용과 각종 로그, 및 설정사항들을 저장하고 읽기위한 DB 관련 데몬이 있으며 이들 daemon 으로부터 메시지를 받아 Printer 에 출력하는 daemon 이 있다.

본 논문에서 다루고자하는 EvtProc daemon 은 하부 Shelf 에서 발생되는 상황변동 메시지를 받아 처리한다. 이들 메시지는 시간적으로 주기성이 있는 보고와 그렇지 않고 발생 시마다 보고되는 메시지로 분류 된다. 주기/비주기적인 메시지는 소량의 보고로 끝날 수 있고 최악의 사태에 순간적으로 대량의 메시지가 보고 되기도 한다. 소량의 메시지 처리는 짧은 시간에 끝날 수 있지만 대량으로 발생되는 상황에 있어 처리에 막대한 시간이 소요 되기도 한다



<그림 1> 기존 회선분배장치의 Daemon 구조

3. Event 처리 메커니즘 개념 설계

3.1 EvtProc Daemon 처리의 문제점

Shelf 에서 수많은 Event 가 보고되면 EvtProc Daemon 은 그 수만큼 fork 하고 exec 을 수행한다. fork 와 exec 은 UNIX OS 에게 많은 오버헤드를 발생시키는데 fork 보다 exec 은 정도가 더 크다

결국 정상적인 상황에서는 문제가 되지 않는 데몬 프로세스 처리 방식이지만 비정상적인 상황에서는 OS 의 오버헤드 뿐만 아니라 회선분배장치에 대한 모든 기능을 수행하지 못하는 결과를 가져온다

3.2 EvtProc Daemon 처리의 해결안 제시

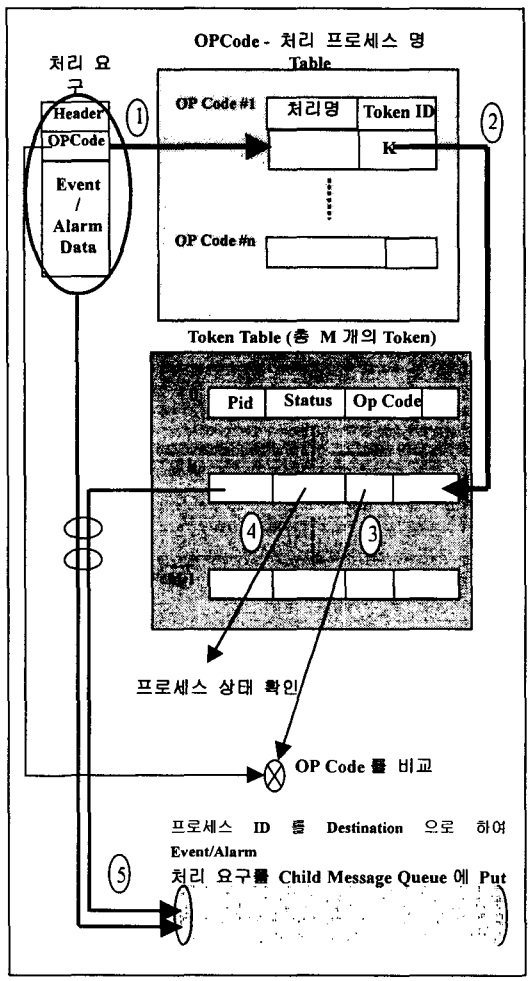
A. 해당 OP Code 에 해당하는 Event 를 처리하는 Child(main) 프로세스가 기동중이라면 그 프로세스에게 단순히 Data 를 전달하기만 하면 되도록 하여 추가의 fork 나 exec 의 오버헤드를 줄이도록 한다

B. Child(main) 프로세스에게 전달되어 지는 Event 처리요구의 과축적상황을 해결하기 위하여 일정개수 이상의 처리 요구가 누적된 경우 Child(main) 프로세스는 또하나의 동일한 프로세스 (Child(sub))를 기동시키고 처리요구를 이관한다음 자신은 다음 메시지를 조사한다. 이 처리는 fork 만 함으로서 exec 의 오버헤드는 감소한다.

C. Child 생성의 한계점을 나타내는 Token 을 사용하여 Child(main)를 관리함으로써 큐에서 넘어오는 상황보고 메시지들을 효율적으로 처리한다

4. Event 처리 개념의 설계

4.1 EvtProc Daemon 의 동작 기능

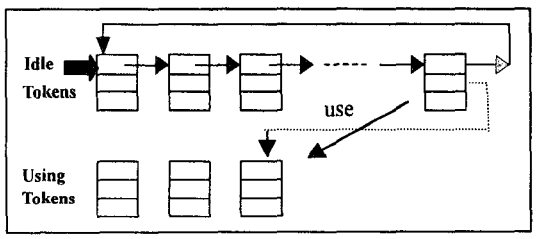


<그림 2> EvtProc 데몬 프로세스 동작 과정

EvtProc Daemon 은 Event 를 보고 받으면 OP Code 를 조사하여 해당 OP Code 의 Child(main) 프로세스가 존재하는지 확인하여 존재하면 그 프로세스에게 데이터를 넘겨준다. 이때 데이터는 큐를 통하여 전달된다. 존재하지 않는다면 Token list 에서 Token 을 할당받아 Child(main)프로세스를 생성하여 처리한다
Child(main)프로세스는 담당 OP Code 에 해당하는 Event 데이터가 일정수를 넘었을때 Child(sub)를 복제하여 처리토록한다

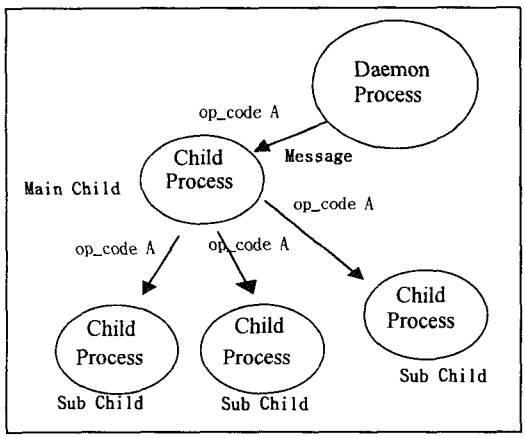
4.2 Child 프로세스 관리 및 동작

과도한 Child 의 생성을 억제하기 위한 알고리즘의 수단으로 Token 을 사용한다.
새로운 알고리즘을 적용하기위한 Child 프로세스 관리 방안으로서 일정개수의 Token 을 리스트로 연결하여 Child(main) 프로세스를 관리한다. Token node 에는 Child(main) 프로세스의 pid, status, OP Code 등의 정보가 들어간다.



<그림 3> Child 프로세스 관리위한 Token 리스트

Token 리스트의 관리는 Critical Region 으로 보호되어 처리되며 할당된 Token node 는 단순히 리스트에서 삭제하는 것으로 처리한다. 이후 Token 의 해제는 해당 Token 을 할당받은 Child(main) 프로세스가 수행하게 되며 Token 의 ID 를 Array 의 index 이자 node 의 key 값으로 사용하여 해당 node 의 정보를 삭제하고 Idle Token 리스트에 연결한다.



<그림 4> Child 프로세스 생성과정

Child(main) 프로세스 생성시 Idle Token 리스트에서 제거한뒤 제거한 Token 을 가지고 처리한다
Daemon 은 전달되는 Event 의 OP Code 를 조사하여 해당 Child(main)의 존재유무를 확인한다. 그리고 해당 Child 에 데이터를 던져주거나 Child(main)을 생성한다

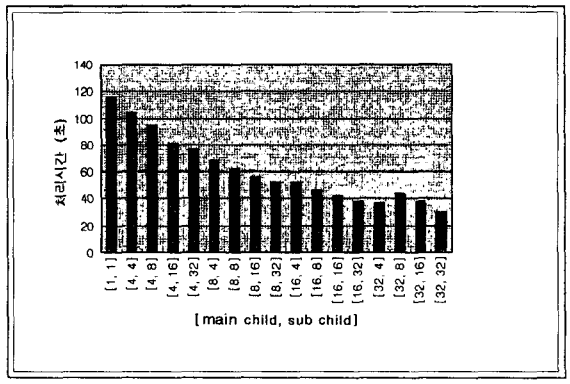
5. 기존 방법과의 비교분석

5.1 실험환경

회선분배장치에서 발생할수 있는 최대 Event 는 약 8000 번이다. 그래서 각각 1000 번, 2000 번, 4000 번, 8000 번의 이벤트를 발생시키는 조건으로 실험하였다.
기존의 방법은 Event 발생시 마다 child 프로세스가 처리하는방법으로 main child 와 sub child 가 1 인 경우에 해당한다.
제안한 알고리즘의 평가 방법은 각각의 Event 발생시 마다 main child 의 수를 4, 8, 16, 32 로 제한하고 다시 main child 당 sub child 의 수를 4, 8, 16, 32 로 제한하여 처리시간을 측정하였다.

5.1 실험결과

Event 발생시 Child 프로세스의 제한수마다 측정한 처리시간을 나타낸 그래프가 <그림 5> 에 있다.



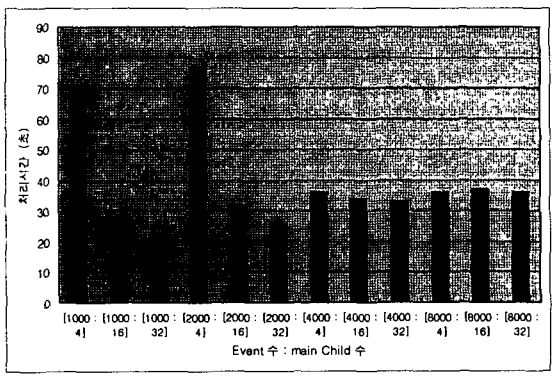
<그림 5> 각각의 Event 발생시 평균 처리시간

이 그래프에서 기존의 방법 즉, main child 의 수가 1 이고 sub child 의 수가 1 인경우보다 해당 op_code 마다 main child 를 여러 개 두어 분산처리함으로서 처리시간을 단축함을 알수있다. 그리고 main child 에 대한 sub child 의 생성도 처리시간 단축에 영향을 미친다..
<그림 6> 은 sub child 의 수가 32 인 조건으로 Event 의 발생건수에 따라 main child 의 제한수를 늘린 경우를 나타내는 그래프이다. 여기서 Event 발생건수가 2000 개일때 main child 의 수의 증가는 처리속도 향상에 많은 영향을 미치고 있다. 하지만 Event 의 발생건수가 8000 개에 달하면 main child 수의 증가는 그다지 처리속도에 영향을 끼치지 못한다. 오히려 악영향을 미칠수도 있다.

그러므로 main child 의 제한수를 결정하는 것도 처리속도 향상에 변수가 된다.

그리고 main child 프로세스의 생성제한수는 바로 Token 의 개수와도 같으므로 Token 의 수 결정으로 연결된다.

그러므로 이러한 분석방법으로 이벤트 발생건수당 가장 효율적인 Token 의 개수를 정하여 알고리즘에 적용하는 것은 성능향상에 영향을 미친다..



<그림 6> *Event 와 main child 수 에 따른 처리속도*

6. 결론 및 향후과제

본 논문에서는 회선분배장치에서 Event 를 처리하는 Daemon 에 효율적인 알고리즘을 적용하여 기존의 방식을 개선하려 하였다.

개선사항으로는 프로세스 생성과 종료의 횟수 감소와 이로인한 Event 처리지연 시간의 감소로 처리능력 향상이 있다. 그리고 주 프로세스에 업무를 이관함으로 daemon 이 가지는 부하를 감소시키고 시간, System 적인 자원의 절약을 가져오는 것이다.

향후과제로 부하분산과 동시에 우선순위 큐를 여러개 두어 대량의 Event 중에서 우선순위별로 나눈다음 다시 Op Code 별로 분산시켜 처리하면 제어관리가 더욱 효율적일것이다. 그리고 비교 분석시 우선순위, 패킷크기, Send time 등 세부적인 변수를 늘려서 가장 효율적인 child 의 제한수를 결정할수 있을 것이다.

참고문헌

[1] Ming-Chwan Chow, UNDERSTANDING SONET/SDH, p.600, Andan Publisher 4 Aufra Place Holmdel.New Jersey 07733, 1995

[2] W. Richard Stevens, Advanced Programming in the UNIX Environment, p.780, Addison Wesley, 1993

[3] 이병기 외 3 인, 광대역 정보통신, p.600, 교학사 , 1996

[4] Bell Communications Research, GR-233-CORE , p.560 , BellCore , 1995

[5] Bell Communications Research, GR-253-CORE , p.560 , BellCore , 1995

[6] ITU-T, ITUT-T G.707International Telecommunication Union , p.120, ITU-T, 1996

[7] Keith Haviland, Ben Salama, UNIX SYSTEM PROGRAMMING, p.329, Addison-Wesley

[8] Brian W. Kernighan, Rob Pike, The Unix Programming Environment, p. 355, Prentice-Hall Software Series

[9] LGIC, WDCS-8000 설명서, p.724, LGIC

[10] A. Silberschatz 저, 김영찬역, 최신운영체제, p. 608, 홍릉과학출판사, 1993

[11] H.M.Deitel, OPERATING SYSTEMS SECOND EDITION, p.800, Addison Wesley, 1994

[12] W.RICHARD STEVENS, UNIX NETWORK PROGRAMMING, SECOND EDITION, p.980, Prentice Hall PTR, 1998