RPC 이벤트 모델에 기반한 통신 이벤트의 디버깅

서 영 애[†] · 조 영 욱[††] · 이 공 선[††] · 박 창 순[†††]

요 약

분산 프로그램의 디버깅이 순차 프로그램의 디버깅보다 어려운 이유 중의 하나로 수행 중인 프로세스들간의 통신을 들 수 있다. 비록 분산 시스템을 위해 구축된 디버거들이 많이 있으나, 프로세스들 간의 통신 이벤트를 효과적으로 디버깅할 수 있는 실용적인 수준의 디버거는 없는 실정이다. 본 논문에서는 분산 시스템의 개발에서 널리 이용되는 RPC 통신을 실시간으로 추적하고 통신내용을 파악함으로써 분산 프로그램을 효율적으로 디버깅하고자 하였다. 이를 위하여 RPC 통신 이벤트를 디버깅 객체로 정의한 이벤트 모델을 제안하고 이에 기초하여 RPC 통신 이벤트 추적 기능을 구현하였다. RPC 프로토콜을 분석한 결과를 이용함으로써 메시지의 송·수신 이벤트를 라이브러리 함수의 호출과 자료의 값으로 기술할 수 있었고, 라이브러리 함수의 호출이 발생한 위치를 인지함으로써, 모든 통신 이벤트의 발생을 탐지할 수 있었다. 제안된 RPC 이벤트 추적 기능은 온라인 분산 디버거인 유니뷰 시스템에 구현되었다.

Debugging of Communication Events Based on the RPC Event Model

Young-Ae Seo[†] · Young-Wook Cho[††] · Kong-Seon Lee[††] · Chang-Soon Park[†††]

ABSTRACT

One of the reasons that debugging distributed programs is much more difficult than sequential programs is the communication among processes. Even though there are many debuggers constructed on distributed system environment, no available debugger provides an efficient way to debug communication events. In this paper, we show the way to debug RPC communication, which is regarded as one of the most popular communication protocol in distributed system development. This paper presents the implementation of the RPC event trace function based on the RPC event model, which is proposed to accommodate communication events into debugging objects. Analyzing conventional RPC protocols, we formalized library function calls as corresponding actions to sending and receiving of messages. By recognizing the locations of library calls, the debugger can detect all occurrences of communication events. This RPC event trace function is implemented on the on-line distributed debugger UniVIEW system.

1. 서 론

최근, 응용 소프트웨어 개발에 있어서 분산 시스템의 중요도가 높아지고 있으며, 이에 따라 분산 환경 소프트웨어의 생산성을 높이기 위한 도구의 개발 및 상품화가 활발히 제시되고 있다. 그러나, 분산 처리 시스템의 개발은 여전히 기존의 순차적 프로그램에 비해 개발비용이 높고 소프트웨어의 질을 보장하기가 어려운 문제점이 있다. 이는 분산 시스템 디버깅의 어려움에서 그 원인을 찾을 수 있으며, 분산 디버깅이 어려운 주요한 원인 중의 하나로 분산 시스템을 구성하는

† 정 회 원 : 한국전자통신연구원 연구원
†† 정 회 원 : 한국전자통신연구원 선임연구원
††† 정 회 원 : 한국전자통신연구원 책임기술원
논문접수 : 1998년 8월 25일, 심사완료 : 1998년 10월 19일

프로세스들간의 통신을 들 수 있다[5, 6].

분산 시스템은 여러 대의 컴퓨터에 분산된 프로세스들간의 상호 통신에 기반하여 작업이 수행되기 때문에, 이러한 통신의 내용을 검사하고 통신에 의한 논리적 흐름을 따라 이동하면서 프로그램을 디버깅하는 기능은 분산 시스템의 개발에 많은 도움을 줄 수 있다.

분산 시스템에서 프로세스들간이 서로 통신하는 방법에는 여러 가지가 있으나, 그 중 특히 RPC(Remote Procedure Call) 매커니즘을 이용하는 방법이 많이 선호되고 있다[9]. 이는 RPC를 이용하는 것이 외부적으로 송·수신 메시지를 명시하는 것보다 프로시듀어를 이해하거나 특정 문제에 관하여 추론하기가 용이하기 때문이다.

본 논문에서는 RPC 프로토콜을 이용하여 통신하는 분산 시스템을 대상으로 온라인 디버깅 시에 통신 이벤트를 실시간으로 디버깅할 수 있는 효율적 방안을 제시하고자 하였다. 이를 위하여 RPC 이벤트 모델을 제안하고 이를 기반으로 하여 이벤트 추적 기능을 설계 및 구현하였다. 제안된 이벤트 추적 기능은 한국전자통신연구원에서 개발한 분산 디버거인 유니뷰 시스템에 구현되었다.

본 논문의 2절에서는 이와 관련된 관련 연구들에 대하여 살펴보고, 3절에서는 제안한 RPC 이벤트 모델에 대하여, 4절에서는 이벤트 모델의 구현에 관해 설명한다. 5절에서는 구현된 사용자 인터페이스에 대해 살펴보고, 6절에서 연구의 결과를 요약한 후 이후의 연구 방향을 살펴봄으로써 결론을 맺는다.

2. 관련 연구

분산 시스템의 디버깅 방식에는 프로그램을 한번 수행시키고 그 결과를 기록한 후 이를 사후 분석하는 오프라인 방식과 프로그램을 수행시키면서 대화식으로 디버깅하는 온라인 방식이 있다.

오프라인 디버깅은 프로그램의 수행 시에 발생한 이벤트들을 기록한 이벤트 트레이스를 사후 디버깅에 이용함으로써 프로그램을 다시 수행시키지 않고서도 디버깅이 가능한 방법이다. 이러한 오프라인 디버깅이 각광받는 이유는 분산 시스템에 내재하는 비결정성 때문이다. 순차 프로그램과는 달리 분산 시스템에서는 예측할 수 없는 통신 소요시간으로 인해 반복되는 수행이 이전과 같은 수행경로를 거친다는 보장이 없다. 그러므로, 분산 디버깅에서는 같은 과정을 다시 재현하기 위해 이벤트 트레이스를 이용하여 인위적으로 같은 경로로 재수행(replay)시키는 방식[6, 9, 11]이나, 광역 조건식(global predicate)을 이용하여 이벤트 트레이스를 분석하는 방식[12, 15]을 통하여 오류 상황이 발생한 지점이나 원인을 찾아내고자 하는 것이다.

이러한 오프라인 디버깅 방식에서는 추적하고자 하는 이벤트의 범위에 통신 이벤트들을 포함시킴으로써 프로세스들간의 통신을 디버깅할 수 있으며, 이를 이용하여 RPC 통신을 디버깅하고자 하는 연구[16]도 있어왔다. 그러나, 이러한 방법은 프로그램의 수행결과를 자동으로 추출하여 이를 기록, 분석하기 위해서는 많은 데이터와 계산량이 요구되며, 디버깅 시 제공되는 정보의 양이 상대적으로 제한되어 있어 통신 상의 버그를 찾기에 부족한 단점이 있다.

이에 반해, 온라인 디버깅은 대화식 순차 디버깅을 분산 시스템의 디버깅에 확장한 형태로서, 사용자는 시스템의 에러가 발견될 때까지 수행을 제어하면서 프로그램을 실행시키는 방법이다. 이 방법은 오프라인 디버깅에서와는 달리 동일한 수행경로를 재연출하기 위해서는 디버깅하는 프로그램을 여러 번 실행시켜야 한다. 이는 오프라인 디버깅과 비교할 때 앞에서 언급한 분산 프로그램의 비결정성으로 인하여 동일한 수행경로를 거치지 않을 수도 있으며, 프로그램을 여러 번 반복 실행시켜야 하는 단점이 있으나, 소스 코드의 특정 위치에서 프로그램을 정지시키거나, 프로그램의 자료 구조를 검사하는 등의 순차 디버깅에서의 일반적인 디버깅 행위를 가능하게 하여, 프로그램 수행에 대한 보다 자세한 관찰이 가능한 장점이 있다[16]. 그러나, 분산 시스템을 위한 기존의 온라인 디버거들[8, 13, 14]은 통신 이벤트를 디버깅할 수 있는 방법을 제시하지 못하였다.

이외에 분산 디버거를 이용하지 않고서 TCPdump나 순차 디버거 등을 이용하여 RPC 통신 발생에 관한 정보를 얻을 수도 있으나, 이러한 방법은 통신 이벤트의 발생을 실시간으로 알 수 없거나, 통신 이벤트의 연관관계를 알 수 없는 등의 문제점이 있었다.

3. RPC 이벤트 모델

일반적으로 RPC에 기반하여 통신하는 프로그램은 서비스를 제공하는 서버의 역할을 하거나 서비스의 제

공을 요청하는 클라이언트의 역할을 한다. 서버 프로그램은 svc_register() 라이브러리 함수를 호출함으로써 제공할 서비스를 등록하고, 클라이언트 프로그램은 clnt_create() 라이브러리 함수를 호출하여 서버 프로그램과의 연결을 구축하고, 서버 프로그램과의 연결에 필요한 정보를 가진 클라이언트 핸들을 생성한다. 서버와 클라이언트간의 연결이 이루어진 후, 클라이언트는 클라이언트 핸들을 이용하여 clnt_call() 라이브러리 함수를 호출함으로써 서버에게 서비스를 요청하고, 클라이언트로부터 서비스의 요청을 기다리고 있던 서버 프로그램은 등록된 디스패처 루틴을 이용하여 요청된 서비스를 처리한 후 그 결과를 클라이언트에게 전달한다.

이와 같이 RPC 프로토콜은 디스패처 루틴의 등록, 클라이언트 핸들의 생성, 원격 서비스의 요청, 요청된 서비스 처리의 네 단계 과정으로 이루어지며, 이들은 각각 수신자(receiver) 생성, 송신자(sender) 생성, 메시지 송신(sent), 메시지 수신(received)으로 대응시킬 수 있다.

이상으로부터 RPC 이벤트 추적을 위한 디버깅 객체[1]를 수신자, 송신자, 메시지 송신, 메시지 수신의 네 가지로 정의할 수 있으며, 각 디버깅 객체는 RPC 특정 라이브러리 함수들로 표현 가능하다. 본 논문에서는 수신자와 송신자는 연결 객체(connection object), 메시지 송신과 수신은 이벤트 객체(event obejct)로 분류하며, 이들은 각각 대응되는 RPC 프로토콜에 의해 다음과 같이 정의된다.

- 수신자 : RPC 통신을 요청 받아 처리하는 주체로서 서버 프로그램이 svc_register() 함수를 호출하여 등록된 디스패처 루틴에 대응한다.

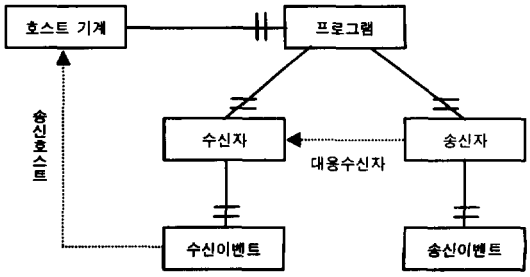
- 송신자 : 원거리 함수의 호출을 신청하는 창구로서, RPC 프로토콜에서는 서버 프로그램을 지정하여 생성되는 클라이언트 핸들이 송신자에 대응한다.

- 메시지 송신 : 원거리 함수의 호출이 메시지 송신에 대응한다. 본 연구에서는 clnt_call() 함수의 호출을 송신에 대응하는 것으로 본다.

- 메시지 수신 : 원거리 함수 호출의 서비스가 메시지 수신에 대응한다. 본 연구에서는 등록된 디스패처 루틴을 한번 수행하는 것을 수신에 대응하는 것으로 본다.

송신자에 대응하는 클라이언트 핸들은 clnt_create() 함수를 호출할 때마다 하나씩 생성되며, 수신할 프로그램의 호스트와 이름에 관한 정보를 가진다. clnt_call() 함수는 클라이언트 핸들을 패러미터로 가짐으로써 대응하는 수신 프로그램에 대한 정보를 제공한다. 후단부 디버거는 메시지가 수신되는 위치에서만 송신 호스트와 송신 이벤트의 유일한 메시지 구분 번호를 알 수 있다.

(그림 1)은 이상에서 정의된 디버깅 객체들간의 연관 관계를 보여준다. 분산 시스템은 여러 개의 호스트 프로세서로 구성되며, 각 호스트 프로세서에는 한 개 이상의 프로그램이 동작한다. 각 프로그램은 여러 개의 송신자와 수신자를 가질 수 있으며 각각의 송·수신자는 하나 이상의 이벤트를 송·수신 할 수 있다. 이 때, 송신자는 대응되는 수신자에 관한 연결 정보(호스트 주소와 포트 번호)를 가지고 있으며, 수신된 이벤트는 이벤트를 송신한 호스트 정보를 알고 있다[2].



(그림 1) 통신 이벤트 객체의 연관도
(Fig. 1) Relationship between Event-related Debugging Objects

RPC 프로토콜에 기반한 이상의 디버깅 객체들을 사용하여, RPC 이벤트 모델 M = (송신자, 수신자, 메시지 송신, 메시지 수신)을 다음과 같이 정의할 수 있다.

(1) 수신자 = $\{R(p, index^{p}_{r})\}$

1) 본 논문에서는 함수나 프로그램 소스 등과 같이 디버거에서 사용자에게 보여지고 다루어지는 디버깅 정보들을 디버깅 객체라 부른다.

2) 전통적인 RPC 프로토콜에서는 프랜스포트 자료구조에 호스트의 주소에 관한 정보가 있으나,송신프로그램을 알 수 있는 충분한 정보는 제공되지 않는다.

(2) 송신자 = {$S(q, index^q_s, p')$, where p' is the corresponding receiver program}

(3) 메시지 송신 = {$s(j, S, message\_id)$, where $S \in$ 송신자}

(4) 메시지 수신 = {$r(i, R, message\_id, sender\_host)$, where $R \in$ 수신자}

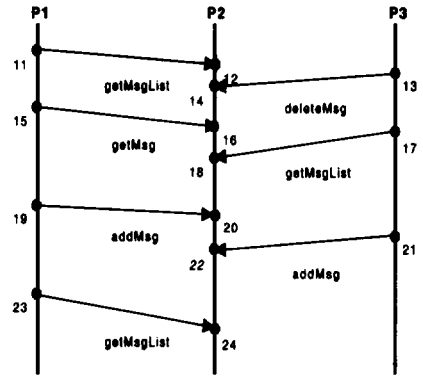
여기서, p, q, p'는 프로그램을 나타내며, i, j는 각각의 이벤트 객체에 대해 유일한 값을 가지는 양의 정수이다. 하나의 프로그램은 여러 개의 송·수신자를 가질 수 있기 때문에, 프로그램 p에서의 수신자 인덱스를 나타내는 $index^p_r$와 프로그램 q에서의 송신자 인덱스를 나타내는 $index^q_s$를 두어 이들을 구별한다.

이상의 RPC 이벤트 모델을 바탕으로 하여 통신 이벤트의 발생을 추적, 기록한 이벤트 추적 리스트(event trace list) L은 $L = c_1...c_ie_je_{j+1}...e_n$ 로 정의된다. 여기서, c_i와 e_j는 각각 연결 객체와 이벤트 객체를 나타내며, $1<= i <= l$, $1<= j <= n$이다.

다음은 분산시스템의 이벤트 추적 리스트의 한 예이며, 이를 프로세스-시간 다이어그램(process-time diagram)의 형태로 표현한 그래프가 (그림 2)이다.

$S1 = S(P1, 1, P2)$ at host1
$R2 = R(P2, 1)$
$S3 = S(P3, 1, P2)$ at host3
...

e_{11} = s(11, S1, getMsgList), e_{12} = r(12, R2, getMsgList, host1)
e_{13} = s(13, S3, deleteMsg), e_{14} = r(14, R2, deleteMsg, host3)
e_{15} = s(15, S1, getMsg), e_{16} = r(16, R2, getMsg, host1)
e_{17} = s(17, S3, getMsgList), e_{18} = r(18, R2, getMsgList, host3)
...



(그림 2) 이벤트 추적 리스트의 그래프 표현
(Fig. 2) Graphical Representation of Event Trace List

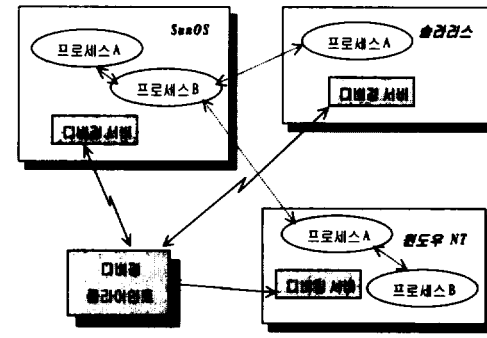
(그림 2)에서 수직선은 프로세스를 나타내고, 수직선 위에 표시된 심벌들은 해당 프로세스에서 송신 혹은 수신한 이벤트들을 나타내며, 11에서 24까지의 숫자는 해당 이벤트의 인덱스를 나타낸다. 화살표로 표시된 선들은 한 프로세스에서 보내진 이벤트가 다른 프로세스에서 수신되었음을 의미한다.

4. RPC 이벤트 추적 기능의 구현

본 논문에서 제안된 RPC 이벤트 모델은 분산 디버거인 유니뷰 시스템에 구현되었다. 본 절에서는 우선 유니뷰 시스템에 대해 알아본 후, RPC 이벤트 모델에 기초한 이벤트 추적 기능의 구현에 관해 설명한다.

4.1 유니뷰 시스템

유니뷰 시스템은 높은 수준의 그래픽 사용자 인터페이스를 제공하는 분산 디버거로서, 현재 UNIX와 NT플랫폼을 포함한 이기종 환경 하에서 동작하는 분산 시스템을 디버깅할 수 있다[1, 2, 10]. (그림 3)은 유니뷰를 사용하여 분산 처리 시스템을 수행하는 모델을 보여준다. (그림 3)의 어플리케이션은 네트워크 상의 세 개의 호스트에서 작동하는 어플리케이션으로서 다섯 개의 프로세스로 구성되어 있다. 이들 프로세스는 네트워크를 통해 통신하면서 상호 의존성을 가지고 동작한다. 각 호스트에는 디버깅 서버(이하 서버)가 하나씩 설치되어 그 호스트에서 작동하는 프로그램의 디버깅을 담당한다. 이 서버는 모두 중앙의 디버깅 클라이언트(이하 클라이언트)와 네트워크를 통해 통신하며, 클라이언트는 서버들과의 통신을 통해 전체 어플리케이션을 디버깅하는 단일화된 인터페이스를 제공한다.



(그림 3) 유니뷰를 이용한 디버깅 모델
(Fig. 3) Debugging Model Using UniView

이와 같이 유니뷰는 디버깅 서버와 디버깅 클라이언트의 두 개의 하위시스템으로 구현되었다. 디버깅 서버는 호스트의 후단부 디버거를 이용하여 프로세스의 디버깅을 수행한다. 현재는 UNIX와 NT플랫폼에서의 후단부 디버거로 GNU 디버거인 gdb[4]를 사용하고 있다. 디버깅되는 프로세스에 대해 각각 하나의 gdb프로세스가 존재하고 각 디버깅 서버는 파이프를 통하여 gdb와 연결되어 gdb프로세스를 제어한다. 클라이언트는 서버들과 통신하면서, 서버가 제공해 주는 모든 디버깅 관련 정보들을 모아 이를 통일된 그래픽 사용자 인터페이스를 통하여 사용자에게 보여준다.

이와 같은 클라이언트/서버 구조로 인하여 유니뷰는 서로 다른 호스트에서 동작하는 여러 프로그램에 대해 단일한 사용자 인터페이스를 제공할 수 있으며, 중앙 클라이언트에서 모든 디버깅 프로그램에 대한 제어와 인터페이스를 담당함으로써 분산 디버깅을 수행할 수 있다.

4.2 RPC 이벤트 모델의 구현

2절에서 제안된 RPC 이벤트 모델을 구현하기 위해서 유니뷰의 디버깅 서버는 모든 통신 객체 및 이벤트 객체의 생성을 탐지하고, 해당 디버깅 객체의 생성 시에 이를 디버깅 클라이언트에게 알려주어야 한다. 또한, 디버깅 클라이언트는 객체 생성에 관한 통지가 올 때마다 디버깅 서버들로부터 필요한 정보를 수집한 후, 디버깅 객체들간의 **연관 관계를 파악하여 사용자에게** 이벤트의 발생을 알려주어야 한다.

이벤트 추적의 시작을 클라이언트로부터 요청받은 서버는 통신 객체 및 이벤트 객체의 생성을 감시하기 위하여 이와 관련된 RPC 라이브러리 함수들에 후단부 디버거를 이용하여 브레이크포인트를 설정한다. 이후 프로그램은 해당 함수들을 수행할 때마다 후단부 디버거에 의해 정지된다. 서버는 정지된 위치를 이용하여 송·수신자의 생성 또는 이벤트의 송·수신을 파악한 후, 후단부 디버거를 이용하여 추출 가능한 이벤트 관련 정보를 얻어낸다. 이렇게 얻어진 정보를 매개변수로 하여 클라이언트에게 특정 이벤트의 발생을 알려준다. 서버는 이벤트 추적과 관련하여 다음의 네 가지 이벤트의 발생을 관련 정보와 함께 클라이언트에게 알려준다.

ReceiverCreated(프로그램ID, 수신자ID)
SenderCreated(프로그램ID, 송신자ID, 수신호스트, 수신프로그램ID)
EventSent(프로그램ID, 송신자번호, 메시지ID)
EventReceived(프로그램ID, 수신자ID, 송신호스트, 메시지ID)

현재, 디버깅 서버에서 이들 이벤트의 발생을 감시하기 위해 브레이크포인트를 설정하는 함수는 svc_register(), clnt_create(), clnt_call(), 그리고 사용자에 의해 정의된 디스패처 루틴들이다.

이벤트의 발생을 연락받은 디버깅 클라이언트는 이를 이벤트 추적 리스트에 연락받은 순서대로 저장하고, 수신자와 송신자 및 메시지의 송신과 수신간의 연관 관계를 찾아야 한다.

리스트에 기록된 이벤트의 순서는 서버와 클라이언트 사이의 통신 지연으로 인하여 논리적 시간의 선후 관계를 따르지 않을 수 있다. 즉, 사용자 어플리케이션에서는 수신자가 생성된 후에야 송신자의 생성이 가능하고 메시지가 송신된 후에야 그 수신이 가능하지만, 클라이언트가 이들 이벤트의 발생을 연락 받는 순서는 바뀔 수 있다. 수신자와 송신자의 연결 관계는 SenderCreated이벤트의 발생 시 서버가 제공해주는 정보들로부터 직접 찾을 수 있는 반면, 메시지 송·수신의 경우는 이벤트 추적 리스트를 분석하여 연관 관계를 찾아야 한다.

본 논문에서는 한 호스트에서 송신된 메시지들이 같은 수신 프로그램으로 전달되는 경우에 메시지들의 수신 순서는 그 호스트에서 송신된 순서와 일치한다는 가정[3]하에서 수신 메시지를 송신 호스트에서 보내진 송신 메시지들에 순서대로 대응시킴으로써 이벤트의 연관 관계를 찾을 수 있었다. (그림 1)에서 보듯이, RPC 이벤트 모델에서 메시지가 수신될 때는 송신 호스트의 정보만을 알 수 있으나, 메시지의 송신 시에는 수신 프로그램에 대한 충분한 정보를 가지기 때문에 이러한 관계 설정이 가능하다.

5. 구현된 사용자 인터페이스

유니뷰 시스템에 구현된 이벤트 추적 기능 및 이와 연계되어 제공되는 순차 디버깅의 기능들은 크게 다음의 세 가지로 요약할 수 있다.

(1) 응용 시스템의 수행 시 제공되는 이벤트 추적 기능

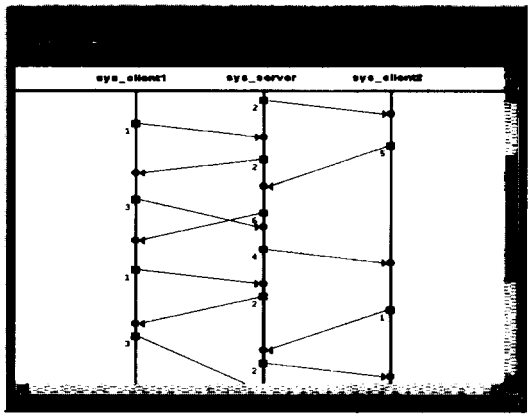
3) 이는 UDP(User Datagram Protocol)를 제외한 RPC 통신 프로토콜의 일반적 구현에서 성립하는 가정이다.

(2) 이벤트 관련 심볼릭 정보 제공 기능(소스코드, 원격호출 패러미터의 값, 스택상태 등)
(3) 추적된 이벤트의 필터링 기능

이러한 기능의 구현으로 인하여 유니뷰 시스템은 특정 통신 이벤트가 발생한 시점에서 프로그램을 정지시키기, 통신 이벤트를 수신한 프로그램을 수신 시점에서 정지시키고 디버깅하기, 통신 메시지나 매개변수값 확인하기, 통신 이벤트의 순서 및 상호관계 검사하기 등의 통신과 관련된 프로그램의 논리를 효과적으로 조사하는 데 필요한 대표적인 디버깅 행위를 지원할 수 있다. 이하에서는 위의 (1), (2), (3)의 기능을 구현된 사용자 인터페이스를 중심으로 차례로 살펴보겠다.

5.1 이벤트 추적 기능

(그림 4)는 (1)의 이벤트 추적 기능을 제공하는 이벤트 추적 윈도우이다. 이 윈도우는 응용 시스템이 실행되어 이벤트가 생성될 때마다 이를 동적으로 화면에 표시하여 준다. (그림 4)에서 수직선은 각각 실행 중인 프로그램을 나타내며, 수직선 위의 사각형과 동그라미는 해당 프로그램의 송·수신 이벤트를, 이벤트들간의 화살선은 이벤트의 연관 관계를 나타낸다. 송신 이벤트의 왼쪽에 표시된 숫자는 메시지 ID로서, 현재는 서버로부터 받은 이벤트의 번호이지만, 사용자가 제공한 정보를 이용하여 실제 메시지의 이름으로 표기할 예정이다.

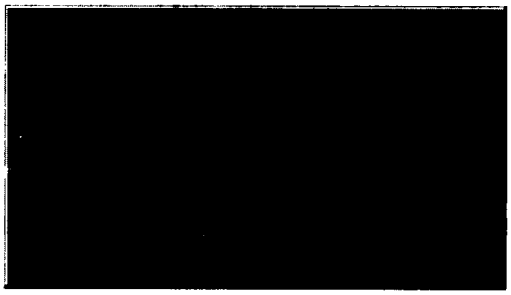


(그림 4) 이벤트 추적 윈도우
(Fig. 4) Event Trace Window

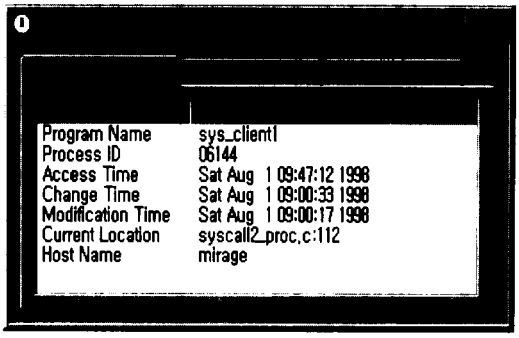
이벤트 추적 윈도우는 발생한 이벤트를 동적으로 보여주는 기능 이외에도 사용자가 이벤트의 발생을 지켜보면서 효율적으로 프로그램을 디버깅할 수 있도록 다양한 디버깅 정보 및 기능의 수행을 위한((2)와 (3)의 기능 수행을 위한) 인터페이스를 제공한다.

5.2 이벤트 관련 심볼릭 정보 제공 기능

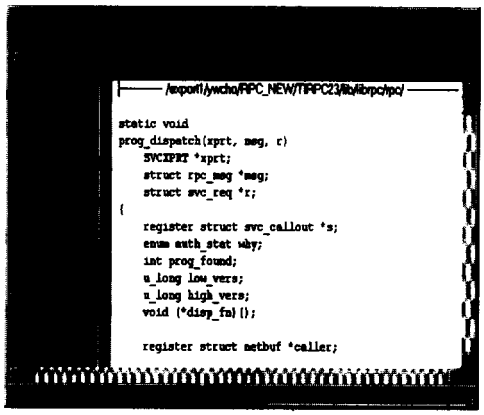
(2)의 이벤트와 관련된 심볼릭 정보를 제공하는 기능을 제공하는 윈도우로는 이벤트 정보 제공 윈도우(그림 5)와 프로그램 정보 제공 윈도우(그림 6), 이벤트 관련 소스 윈도우(그림 7), 프로그램 관련 프로그램 제어 윈도우(그림 8)가 있다. 사용자는 이벤트 추적 윈도우를 통해 이벤트의 발생을 지켜보다가, 관심이 있는 이벤트나 프로그램에 관련된 정보를 볼 수 있으며 (그림 5, 그림 6), 이벤트가 발생한 곳의 소스 파일을 볼 수 있다(그림 7). 송신 이벤트의 경우는 해당 이벤트에 대한 clnt_call() 함수의 소스가 보여지며, 수신 이벤트의 경우는 디스패처 루틴의 소스가 보여진다.



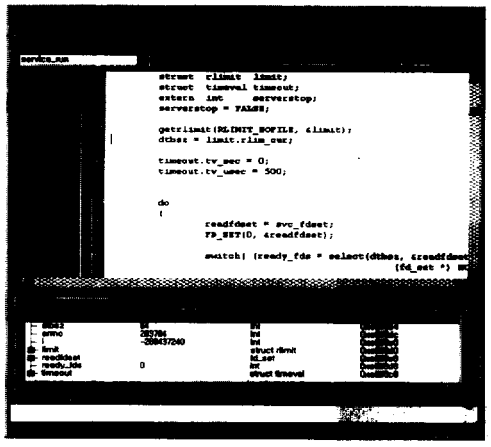
(그림 5) 이벤트 정보 제공 윈도우
(Fig. 5) Event Information Window



(그림 6) 프로그램 정보 제공 화면
(Fig. 6) Program Information Window



(그림 7) 이벤트 관련 소스 윈도우
(Fig. 7) Event Related Source Window

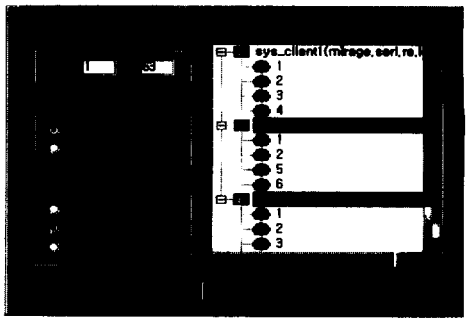


(그림 8) 프로그램 관련 프로그램 제어 윈도우
(Fig. 8) Program Related Program Control Window

또한, 사용자는 이벤트 추적 중인 프로그램의 실행을 정지시킨 상태에서 프로그램의 소스 파일을 보면서 브레이크포인트를 설정하고 변수 값을 보거나, 스텝인, 스텝 아웃 등과 같은 일련의 순차 디버깅 기능을 수행할 수 있다. (그림 8)은 이를 위해 제공되는 프로그램 제어 윈도우이다. 윈도우의 하단부는 활성 프레임의 지역 변수나 감시 변수 리스트들에 대해 프로그램이 정지할 때마다 현재 값을 보여 준다.

5.3 이벤트 필터링 기능

사용자는 이벤트 추적을 잠시 멈추고 지금까지 추적된 이벤트 중 필터링 연산을 통하여 관심 있는 이벤트만을 선택하여 볼 수 있다. (그림 9)는 이벤트의 필터링을 위해 제공되는 대화상자로서, 사용자는 대화상자의 오른쪽에 있는 트리에서 관심 있는 특정 프로그램이나 이벤트들을 선택하고, 원하는 형태의 필터링 스코프와 필터링 모드를 선택하면, 이벤트 리포트 윈도우에 필터링된 결과가 보여진다. 이벤트 리포트 윈도우는 이벤트 추적 윈도우에서처럼 통신 이벤트의 디버깅과 관련된 다른 윈도우를 열 수 있다.



(그림 9) 이벤트 필터링 대화상자
(Fig. 9) Event Filtering Dialog Box

6. 결 론

본 논문에서는 분산 시스템을 구성하는 프로세스들 간의 RPC 통신을 실시간으로 추적하고 통신 내용을 파악함으로써 분산 시스템의 통신상의 에러를 효과적으로 디버깅하고자 하였다. 이를 위하여 RPC 통신 이벤트의 디버깅을 위한 이벤트 모델을 제안하였으며 이를 기초로 하여 RPC 이벤트 추적 기능을 온라인 디버거인 유니뷰 시스템에 구현하였다.

본 시스템에서 구현된 이상의 이벤트 추적 기능은 오프라인 방식에 기반하여 이벤트 추적 기능을 구현한 기존의 시스템들과 비교할 때 다음의 두 가지 중요한 차이가 있다.

우선, 대화식 디버깅 인터페이스에 의해 분산 프로그램의 수행을 제어하면서 동적으로 이벤트의 추적이 가능하다는 것과 이벤트의 송·수신에 따른 소스코드 네비게이션 등의 순차적 디버깅 기능이 제공된다는 점이 그것이다. 이는 사용자에게 통신 에러를 찾아내는 데 필요한 보다 많은 디버깅 정보와 기능을 제공하여 통신 에러의 디버깅을 용이하게 한다는 장점이 있다.

그러나, 본 논문에서 제안한 이벤트 추적 기법은 온

라인 디버깅에 기초하여 이루어졌기 때문에 이는 온라인 디버깅의 일반적인 단점을 공유하게 된다. 즉, 분산 시스템의 비결정성으로 인하여 에러가 발생한 상황을 다시 재현하는 것이 어려울 수 있으며 또한, 동일한 상황을 재현하는 것이 가능하더라도, 시스템을 정지시키는 등의 디버깅 행위가 프로그램의 수행을 방해하여 에러가 발생한 상황이 재현되지 않을 수도 있다.

이를 해결하기 위한 방법으로 오프라인 디버깅에서 제안한 이벤트 트레이스를 사용하여 분산 시스템을 재실행시키는 방법이 있다. 본 연구에서는 앞으로 재실행 기법을 유니뷰 시스템의 대화식 디버깅 기능에 결합함으로써 이러한 문제점을 해결하고자 한다.

참 고 문 헌

[1] 정보통신부, "분산처리 진단/교정 소프트웨어 개발에 관한 연구", 1차년도 연구개발 결과 보고서, 1997.

[2] 성명제 외 4인, "분산 디버거 유니뷰 시스템의 개발", 정보과학회 98년 논문지(C), 10월호, 1998(게재 확정).

[3] J. Bloomer, *Power Programming with RPC*, O'Reilly & Associates, Inc.

[4] Cygnus support, *Working with gdb*. 매뉴얼.

[5] Gartner Group, "Internet Development : Mastering New Tools," ADM Strategic Analysis Report, March 1997.

[6] H. Garcia-Molina, et. al., "Debugging a distributed computing system," IEEE TSE Vol.SE-10, No.2, pp.210-219, 1984.

[7] L. Lamport, "Time, clocks and the ordering of events in a distributed system," Comm, ACM, pp.558-565, Feb. 1976.

[8] LANL Distributed Computing Group, http://w3.lanl.gov:8010/computer-information/ComputingNews/1995Archives/November1995HTMLVersion/Suresh_HPD.html, 1996.

[9] T. LeBlanc and J.Mellor-Crummey, "Debugging parallel programs with instant replay," IEEE Trans.on Computers, Vol.C-36, No.4, pp.471-482, 1987.

[10] E. Lee, et. al., "UniVIEW : A distributed debugger for client/server systems," SERI journal, January, 1998 (in English).

[11] W. Lloyd and Phiol Kearns, "Tracing the Execution of Distributed Programs," Journal of Systems Software, Vol.21, pp.201-214, 1993.

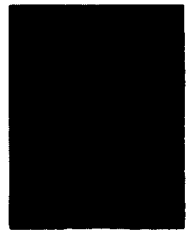
[12] Y Manabe and M.Imase, "Global Conditions in Debugging Distributed Programs," J. Parallel and Distributed Computing, Vol.15, pp.62-69, 1992.

[13] NAS Parallel Tools Team, http://science.nas.nasa.gov/Software/p2d2/, 1996.

[14] Total View Home Page, http://www.dolphinics.com/tw/tvover.htm, 1997.

[15] S. Venkatesan and B. Dathan, "Testing and Debugging Distributed Programs Using Global Predicates," IEEE TSE Vol.21, No.2, pp.163-177, 1995.

[16] W. Zhou, "The Design and Implementation of a Distributed Program Monitor," J. System Software, 22:63-77, 1993.



서 영 애

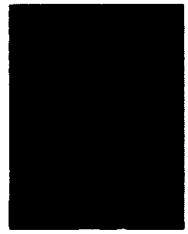
e-mail : yaseo@etri.re.kr

1996년 경북대학교 전자계산학과 졸업(학사)

1998년 포항공과대학교 전자계산학과 졸업(석사)

1998년~현재 한국전자통신연구원 연구원

관심분야 : 분산처리시스템, 실시간 시스템, 자연언어처리 등



조 영 욱

e-mail : ycho@etri.re.kr

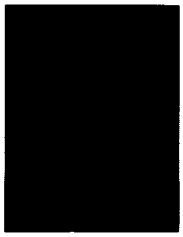
1985년 부산대학교 계산통계학과 졸업(학사)

1987년 충남대학교 계산통계학과 졸업(석사)

1990년~1998년 5월 시스템공학연구소 선임연구원

1998년~현재 한국전자통신연구원 선임연구원

관심분야 : 분산처리시스템, 실시간 시스템, 개발환경 등



이 공 선

e-mail : klee@etri.re.kr

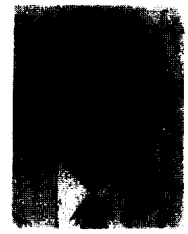
1984년 연세대학교 기계공학과 졸업(학사)

1992년 연세대학교 전자계산학과 졸업(석사)

1986년~1998년 시스템공학연구소 선임연구원

1998년~현재 한국전자통신연구원 선임연구원

관심분야 : 분산시스템 테스팅, 소프트웨어 공학



박 창 순

e-mail : cpark@etri.re.kr

1975년 서울대학교 응용수학과 졸업(학사)

1992년 연세대학교 전자계산학과 졸업(석사)

1992년~1998년 시스템공학연구소 책임기술원

1998년~현재 한국전자통신연구원 책임기술원

관심분야 : 네트워크 컴퓨팅, 실시간 시스템