# 운영체제 행위의 이해를 위한 확장적 시각화 도구

강정혜, 박미영, 구금서,° 전용기
경상대학교 컴퓨터과학과
kjh762@hanmail.net, {park,goodman,jun}@race.gsnu.ac.kr

# A Scalable Visualization Tool for Understanding Operating System Behavior

Jung-Hye Kang, Mi-Young Park, Geum-Seo Koo,° and Yong-Kee Jun
Dept. of Computer Science, Gyeongsang National University

요 약

운영체제의 행위를 이해하기 위해서는 운영체제와 프로세스간의 상호작용을 감시하여 효과적으로 보고하는 시각화 도구가 중요하다. 본 연구에서는 리눅스 운영체제를 대상으로 하여 기존의 도구가 제공하지 못하는 추상적 시각화 기능을 선택적으로 제공함으로써 감시대상인 행위를 확장적으로 쉽게 이해할 수 있게 하는 도구를 제안한다.

## 1. 서 론

사용자가 컴퓨터 시스템에 입력한 명령은 운영체제와의 상호작용[4]을 통해서 수행이 된다. 즉 여러 개의 프로세스들이 각종 사건(event)을 통해 커널(kernel)이나 다른 프로세스로 제어가 이동되면서 수행된다. 사용자가 운영체제의 행위를 이해하기 위해서는 이러한 상호작용 과정을 알아야 할 필요가 있지만 이러한 과정은 복잡하게 이루어지므로 사용자는 많은 지식과 노력이 필요하다. 하지만 이러한 과정을 시각화를 통해 제공한다면 문자보고 방식에 비해 이해가 쉬워질 것이다.

이러한 상호작용 과정을 감시하는 기존의 도구들은 개발 목적에 따라서 디버깅, 시스템 성능분석, 시스템의 행위이해 등으로 나뉘어 진다. 이들 도구는 특별히 주문된(custom) 사건이나 특정 사건유형으로 정의된(defined) 사건을 추적(trace)한 자료를 버퍼나 디스크에 저장한 후에 사용자에게 제공한다. 주문된 사건을 감시하는 도구는 사용자가 시스템 설계 시에 정의되어진 추상적 사건들을 추적하는 도구로서, 사용자가 다양한 사건들로 정의되는 추상사건들을 정의하기가 어렵다는 단점이 있다. 정의된 사건을 감시하는 도구는 사용자들에게 중요한 사건들을 미리 정의하고 있다. 하지만 이들 도구는 모든 사건들을 시각화해서 나타내므로 시각화된 결과도 복잡하다는 문제점이 있다.

본 논문에서는 이러한 상호작용 과정을 쉽게 이해할 수 있도록 하기 위해서 모든 프로세스를 System Process Set 및 User Process Set으로 구분하고 Device Set과 Kernel을 포함하여 4개의 추상적 프로세스 군들을 정의한다. 그리고 이러한 추상적 프로세스들을 시각화하기 위해 3개의 모듈로 구성한 도구를 제안한다. 따라서 본 논문에서 제안한 도구는 사용자의 관심을 반영한 프로세스를 확장적으로 시각화하여 운영체제와 프로세스간 상호작용을 쉽게 이해할 수 있게 한다.

2장에서는 기존의 커널 감시도구들과 이들이 갖는 문제점 및 추상적인 시각화의 중요성을 소개하고, 3장에서는 그러한 문제점들을 해결하기 위한 본 도구를 소개한다. 4장에서는 구현된 도구를 실험한 결과를 보이고, 마지막으로 5장에서는 결론과 향후과제를 제시한다.

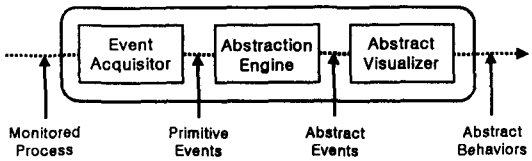
## 2. 연구배경

운영체제와 프로세스간 상호작용 과정을 알기 위해서는 커널 사건들을 감시하는 것이 중요하다. 이러한 감시 도구[4]에는 주문된 사건들을 감시하는 도구인 DProbes[8]와 정의된 사건들을 감시하는 도구인 LTT[5], SpyKer[6], LKST[7], IKD Kernel tracer 등이 있다. 정의된 사건들을 감시하는 도구들은 필요한 사건을 몇 개의 사건유형으로 정의하고, 사용자가 필요로 하는 사건들만으로 감시대상을 정할 수도 있다. 이 도구는 특정 사건만 추적하기 때문에 추가부담을 최소화할 수 있다. 추적된 자료는 커널버퍼에 담겨지고, 다시 디스크에 저장해서 사용자에게 문자 형태나 시각화 형태로 제공된다.

이들 도구 중에서 본 논문이 대상으로 하는 리눅스[1,2] 기반의 시각화를 지원하는 도구로는 LTT, SpyKer 등이 있다. LTT(Linux Trace Toolkit)는 Kernel patch를 통해서 추적된 사건들을 저장하고, Kernel module에서 저장된 사건을 커널버퍼에 저장한다. 커널버퍼의 자료를 읽고, 디스크에 저장하는 기능은 추적모듈(trace module)에서 담당하게 된다. 추적모듈에 의해 디스크에 저장되는 파일은 out.trace 및 out.proc 등의 두 파일인데, 이들 파일은 시각화의 입력으로 사용된다. LTT는 추적된 프로세스에서 발생된 모든 사건과 제어의 이동이 시각화되어 나타나므로 많은 프로세스 수와 각 프로세스에서 발생된 여러 종류의 사건들에 의해 상호작용이 수백 페이지에 걸쳐 시각화된다. 그러므로 이러한 복잡한 결과를 가지고 운영체제의 상호작용을 이해하기는 쉽지 않다는 문제점이 있다. Spyker는 시간에 따라 스레드 라인에 그 스레드의 수행시간과 커널의 수행시간을 타임라인(time line)으로 나타내는데, 프로세스간의 상호작용 및 커널과의 상호작용을 시각화하지 못하는 문제점이 있다.

그러므로 이러한 상호작용 과정의 이해를 돕기 위해서 추적된 자료를 근거로 한 추상적 시각화가 효과적일 수 있다. 추상적 시각화를 위해 추적된 사건들을 추상적 사건(abstract event)로 재구성할 필요가 있으므로, 본 논문에서는 프로세스들을 4개의 프로세스 군으로 분류한 추상적 시각화 도구를 제안한다.

## 3. 추상적 시각화 도구




[그림 3-1] 추상적 도구의 핵심적인 process들

본 장에서는 본 시각화 도구를 [그림 3-1]과 같이 3개의 처리 부분(Event Acquisitor, Abstraction Engine, Abstract Visualizer)으로 나누고, 이들의 기능을 각각 소개한다.

### 3.1 Event Acquisitor

본 도구에서는 LTT를 이용해서 추적 자료를 획득한다. LTT는 운영체제와 프로세스간의 상호작용을 이해하는데 필요한 자료를 제공해 주는 도구이며, 커널감시를 위해 Trace Facility, Trace Module, Daemon 등으로 구성되어 있다. Trace facility는 Kernel patch로 커널을 변경한 부분으로서 추적사건들을 수집하는 역할을 한다. Trace module은 수집된 사건들을 /dev 디렉토리에 저장하는 역할을 담당한다. Daemon은 커널모듈의 버퍼를 읽거나 디스크에 저장하는 역할을 담당한다.

LTT 프로세스는 trace 명령과 응용프로그램을 입력으로 받아서 커널을 감시한 원시사건(primitive event)들을 출력하며, 이 자료는 out.trace 및 out.proc 등의 파일 형태로 디스크에 저장된다. LTT에서 감시하는 사건유형은 [표 3-1]과 같다. 이러한 사건들을 저장하면 자료의 양이 적게는 1MB에서 수십 MB까지 방대해지므로, LTT에서는 압축된 형태로 추적 파일을 제공한다. 본 도구에서는 LTT로 trace dump 파일을 받아서 [표 3-2]와 같이 필요한 부분만을 추적 자료로 이용하였다.

### 3.2 Abstraction Engine

사용자의 관심을 반영하기 위해서 LTT에서 획득한 원시사건들을 추상화하는 부분이다. 추상화는 크게 Process 추상화와 Device 추상화로 분류하며, 추상화된 프로세스 군은 Device, System Process, User Process, Kernel 등으로 분류한다. Device 부분은 원시사건들에서 프로세스로 따로 나타내지 않기 때문에 본 도구에서는 프로세스로 추상화하여 나타낸다.

User Process Set은 사용자가 입력한 명령이나 응용프로그램의 집합이다. 그 외의 프로세스들은 모두 System Process Set에 속하는 것으로 분류했다. 추상화를 위해서 사용자가 원하는 프로세스 군에만 임의의 PID를 부여한다. 원시사건들에서는 각각의 PID가 추상화할 프로세스 군에 해당되는지를 판단하고, 해당되는 프로세스이면 이를 임의의 PID로 변환한다. 이렇게 함으로써 추상적 시각화를 위한 추상사건들을 획득할 수 있다.

### 3.3 Abstract Visualizer

| | | |
|---|---|---|
| Trace start | Interrupt exit | Timer |
| System call entry | Scheduling change | Memory |
| System call exit | Kernel timer | Socket comm. |
| Trap entry | Bottom half | IPC |
| Trap exit | Process | Network |
| Interrupt entry | File system | |

[표 3-1] 추적되는 커널사건 유형

| | | | |
|---|---|---|---|
| Syscall entry | 754 | 14 | SYSCALL : munmap; EIP : 0x0804B3E6 |
| Memory | 754 | 14 | PAGE FREE ORDER : 0 |
| Syscall exit | 754 | 6 | |
| Syscall entry | 754 | 14 | SYSCALL : open; EIP : 0x0804B11C |
| IRQ entry | 754 | 7 | IRQ : 15, IN-KERNEL |
| IRQ exit | 754 | 6 | |
| Sched change | 685 | 18 | IN : 685; OUT : 754; STATE : 2 |
| IRQ entry | 685 | 7 | IRQ : 15 |
| IRQ exit | 685 | 6 | |
| IRQ entry | 685 | 7 | IRQ : 15 |
| IRQ exit | 685 | 6 | |
| Syscall entry | 685 | 14 | SYSCALL : writev; EIP : 0x080C1137 |
| File system | 685 | 22 | WRITE : 16; COUNT : 1 |
| Socket | 685 | 18 | SO SEND; TYPE : 1; SIZE : 64 |
| Process | 685 | 18 | WAKEUP PID : 791; STATE : 1 |
| Syscall exit | 685 | 6 | |
| Sched change | 791 | 18 | IN : 791; OUT : 685; STATE : 0 |
| File system | 791 | 22 | SELECT : 3; TIMEOUT : 2147483647 |
| Memory | 791 | 14 | PAGE FREE ORDER : 0 |
| Syscall exit | 791 | 6 | |

[표 3-2] Primitive events

Abstract Visualizer는 Abstraction Engine에서 출력된 추상자료를 운영체제와 프로세스간 추상적인 행위로 표시하는 시각화 부분이다. 시각화의 구성요소는 .수직간선과 수평간선으로 나타내고, 수평간선은 방향간선과 비 방향간선을 이용했다. 수평간선은 제어의 흐름을 나타내며 이를 나타내는 사건유형으로는 System Call Entry, System Exit, Interrupt Entry, Interrupt Exit, Trap Entry, Trap Exit, Scheduling Change 등이 있다. 이들 제어 흐름의 이해를 돕기 위해 System Call은 파랑, Interrupt는 빨강, Trap은 회색, Scheduling Change는 녹색 등으로 구분한다. 수직간선은 프로세스, 커널 등의 수행시간을 나타낸다. 수행 시간은 모든 사건에 동일하게 지정한다. 즉, 본 도구는 사용자가 운영체제와 해당 프로세스간의 상호작용을 이해하기 위한 것이므로 물리적인 수행시간에 의미를 두지 않고 논리적인 사건의 순서에만 의미를 둔다. 그러므로 수직 방향간선은 논리적 시간에 따른 각 프로세스의 진행을 나타낸다.

## 4. 구현 및 실험

본 논문에서 제시한 도구는 커널 버전 2.2.14-6 RedHat Linux 6.2에서 구현하였다. LTT(버전 0.9.3)는 미리 컴파일한 bzImage_trace와 modules.tgz를 다운로드 받아서 설치하고, /etc/lilo.conf 파일을 수정한 뒤에 재부팅하였다. Event Acquisitor 부분은 기존의 LTT 도구를 이용했고, Abstraction Engine은 C 언어로, Abstract Visualizer 부분은 그래픽 기능이 강한 Java 언어를 이용해서 구현하였다.

Event Acquisitor는 LTT 도구의 trace 명령을 이용해 감시한 자료를 획득한다. 이 자료는 디스크에 out.trace, out.proc로 저장된다. out.trace 파일은 용량이 많기 때문에 압축된 형태로 제공된다. 따라서 본 도구에서는 out.trace 파일을 그대로 이용하지 않고 out.trace 파일을 입력으로 받아서 traceview 파일을 실행시켜 dump 파일을 만든 후 사용한다. 본 도구는 논리적인 사건 순서에 의미를 두기 때문에 모든 사건 유형의 수행시간은 동일하게 지정했다. 또한 본 도구에서는 각각의 Device를 프로세스처럼 구분하여 시각화에 반영했다. 구분 방법은 사건 유형 중 IRQ entry 사건이 발생할 때 ID를 이용해서 각 Interrupt ID마다 새로운 PID를 부여했다.

-Abstraction Engine 부분은 사용자가 지정한 옵션에 따라 추적된 데이터 파일을 재구성한다. 이를 위해 선행되어야 할 작업이 4개의 프로세스 군으로 분류할 수 있는 기준이다. Device 군은 Interrupt ID를 이용해 새로 부여한 추상 프로세스를 정하는 일이고, User Process 군은 사용자가 추적하기 위해 사용자 명령에의 프로세스들이다. 그리고 커널은 PID가 0인 프로세스이다. 이 외의 프로세스들은 모두 System Process 군으로 분류한다. 따라서 사용자가 지정한 추상화 옵션에 따라

이들 군에 속한 프로세스를 특정 하나의 프로세스로 지정함으로써 하나의 프로세스로 추상화하는 것이다. 즉, 선택한 추상화 옵션에 속하는 프로세스는 추상화되고, 선택되지 않은 프로세스 군에 속한 프로세스는 구체적으로 시각화된다.

Abstract Visualizer는 사용자의 관심에 따라 추상화하는 부분이다. 시각화 방법은 점에서 시작해 점에서 끝나는 간선을 이용했다. 수평간선은 제어의 이동이며 수직간선은 수행시간을 표현한다. 또한 사용자가 쉽게 이해할 수 있도록 하기 위해 여러 가지 색상을 이용했다. 수평간선을 나타내는 사건 유형은 [표 4-1]과 같다.

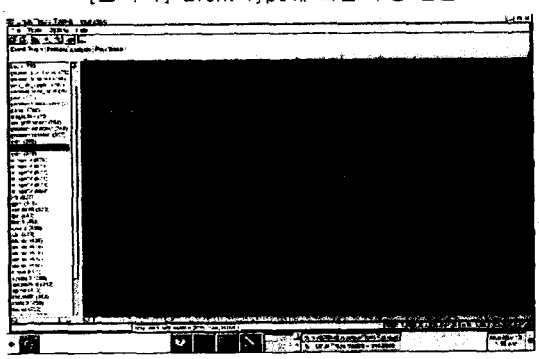
실험을 위해 감시된 프로세스 예는 'cat hap|wc' 명령이며, LTT를 이용해서 'trace 1 ex3; cat hap|wc'를 입력했다. 'trace 1 ex3;'는 1초 동안 추적해서 ex3.trace, ex3.proc 등의 파일을 디스크에 저장하고, 이 파일을 이용해서 dump 파일을 생성한다. 여기서 hap는 수의 합을 구하는 응용 프로그램 이름이다. 이 응용 프로그램을 1초 동안 실행하여 추적된 자료를 시각화하고, 본 도구의 추상화 옵션에 따라 시각화된 결과를 LTT와 비교하여 본 도구가 효과적임을 보였다. [그림 4-1]은 dump 파일을 LTT를 이용하여 시각화한 것이고, [그림 4-2]는 [표 3-2]를 본 도구를 이용하여 시각화한 화면이다. 이 도구에서 Device 부분이 새로운 프로세스로 나타나 있는 것을 볼 수 있다. 이를 통해 어떤 Device가 언제 사용되는지를 알 수 있다. [그림 4-3]은 [그림 4-2]의 시각화에서 Device 그룹만 구체화하고 다른 프로세스 그룹은 추상화한 것이다.

본 도구는 LTT를 포함해서 기존의 도구가 제공하지 못하는 추상화 기능을 제공함으로써 사용자가 운영체제의 행위를 쉽게 이해할 수 있다. 즉, 관심 있는 프로세스 그룹은 자세히 볼 수 있고, 그 외의 프로세스 그룹은 추상화해서 시각화할 수 있기 때문에 사용자 중심의 관점에서 시각화하는 도구이다. 따라서 본 도구는 일반 사용자가 운영체제와 프로세스간의 상호작용을 이해하는데 효과적인 도구가 될 수 있다.

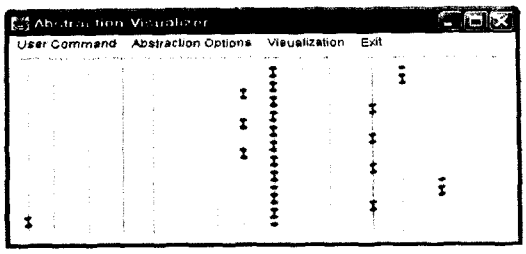
## 5. 결론 및 향후과제

| Event Type | 간선모양 kernel(0) process(PID) | 비고 |
|---|---|---|
| Syscall entry | Syscall 명 | Syscall 명은 구체화된 프로세스 그룹에서만 표시 |
| Syscall exit | | |
| IRQ entry | | Discription: IN-KERNEL |
| | | Discription: IN-KERNEL 이벤 |
| Trap entry | | |
| Trap exit | | |
| Sched change | | |

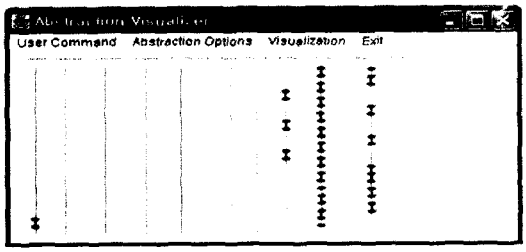
[표 4-1] Event Type에 따른 수평 간선



[그림 4-1] LTT에 의한 시각화




[그림 4-2] 추상화하지 않은 시각화




[그림 4-3] Device 그룹만 구체화

본 연구에서 운영체제의 행위를 이해할 수 있는 실용적 시각화 도구를 제안하였다. 본 도구는 모든 프로세스들을 4개의 프로세스 군으로 통합하여 추상화하고, 원하는 프로세스 군만을 구체적으로 볼 수 있다. 향후과제로는 시각화된 결과에서 System Call, Trap, Interrupt 등과 같이 사용자가 원하는 사건만 나타내고 다른 사건들은 숨기는 옵션 기능을 개선하는 것이 필요하다.

## 참고문헌

[1] Bokhari, S. H., "The Linux Operating System," *Computer*, IEEE, Aug. 1995.

[2] Bech, M., H. Bohme, M. Dziadzka, U. Kunitz, R. Magnus, and D. Verworner, *LINUX Kernel Internals*, Addision-Wesley Longman, 1998.

[3] Wilner, D., "WindView: A Tool for Understanding Real-time Embedded Software Through System Visualization," *SIGPLAN Notices*, ACM, Nov. 1995.

[4] Haviland, K., D. Gray, and B. Salama, *UNIX System Programming*, 2nd Edition, Addison-Wesley, 1999.

[5] Yaghmour K. and M. R. Dagenais, "Measuring And Characterizing System Behavior Using Kernel-Level Event Logging," *USENIX Annual Technical Conference*, San Diego, California, June. 2000.

[6] LynuxWorks, Inc., *SpyKer: Auto-Instrumented Tracing of System Events for Rapid Time-to-Market, 2001.*

[7] Hitachi, Ltd., *Linux Kernel State Tracing Facility Function Specifications,* Version 01-06, *2001-2002.*

[8] Moore, R. J., *Dynamic Probes and Generalised Kernel Hooks Interface for Linux*, IBM, Linux Technology Centre, 2002.