# core 파일을 이용한 프로세스 수행 흐름 분석 도구의 설계 및 구현

홍석일○ 국중진 홍지만
숭실대학교 컴퓨터학과
hongsukil@gmail.com○, tipsiness@gmail.com, jiman@ssu.ac.kr

# Design and Implementation of a Process Flow Analysis Tool by using core file

Sukil Hong○ Joongjin Kook Jiman Hong
Soongsil University

**요 약**

기존의 프로세스 흐름 분석 도구(ltrace, strace) 및 디버깅 도구(gdb)를 사용하여 프로세스의 여러 시점을 한 번에 분석하기는 불가능하다. 또한 주로 콘솔 기반으로 수행하므로 사용에 어려움이 따른다. 본 논문에서 설계 및 구현한 프로세스 흐름 분석 도구는 프로세스의 수행 도중 원하는 시점마다 core 파일을 생성하고, core 파일을 이용하여 프로세스의 메모리 및 레지스터에 대한 정보를 분석한다. 여러 core 파일을 동시에 비교함으로써 프로세스의 수행에 대해 여러 시점을 비교 가능토록 하였고 또한 qt 라이브러리를 이용하여 비주얼적인 디스플레이를 통해 알아보기 쉽도록 구현하였다.

## 1. 서 론

프로그램이 시작하면 종료를 하기까지 단 한 번의 수행 흐름을 갖게 된다. 또한 프로그램 수행 도중 예상치 못한 오류로 인하여 종료되거나, 수행이 완료된 경우 수행 중이던 내용은 모두 사라지게 된다. 수행이 종료되면 진행 중이었던 임의의 시점에서 어떤 데이터를 가지고 있는지 알아내기 쉽지 않다. 가령 프로그램 내에서 화면 출력 등을 통하여 전역 또는 지역 변수, 함수 호출 진행 등의 정보는 간단하게 알아낼 수 있지만, 메모리 및 레지스터 정보는 디버깅 도구 등을 사용하지 않으면 알아내기 쉽지 않다. 또한 프로그램의 수행 정보를 저장하여도 그 정보를 활용하는 것은 다른 도구를 사용해야 한다. 프로그램을 수행하면서 프로세스 정보들(메모리, 데이터, 레지스터 등)을 임의의 시점에서 저장함으로써 수행 도중 변하는 프로그램 내부 정보를 알 수 있으며 프로그램 수행 상태를 파악할 수 있다. 본 논문에서는 mkcore() system call을 이용하여 프로그램에 대한 core 파일을 생성하고 생성된 core 파일 정보를 바탕으로 VCTrace 도구를 이용하여 프로그램의 수행 흐름을 분석하는 기법을 제안하고자 한다. 본 논문의 2장에서는 프로세스의 수행 분석에 대한 기법과 관련된 내용을 언급하고, 3장에서는 논문에서 설계하고 구현한 도구에 대해 설명한다.

## 2. 관련 연구

예전부터 프로세스에 대한 수행 분석 및 디버깅에 대한 연구가 끊임없이 이루어지고 있다. 이는 프로그램을 개발하는 것도 중요하지만 프로그램의 수행과정 및 결과에 큰 비중을 두고 있다는 뜻이다. 실제로 개발 기간 중 가장 많은 시간을 소비하는 것이 디버깅 과정이다. 기존의 프로세스 수행 분석 도구에는 ltrace[1], strace[2], gdb[4] 등이 있다. ltrace는 프로세스가 호출하는 공유 라이브러리 함수에 대한 내용을 보여주며, strace는 프로세스가 호출하는 system call 에 대한 정보를 알 수 있다. 하지만 이러한 프로세스 분석 도구만으로는 프로세스의 정보(메모리, 레지스터 등)를 알아내기 힘들다. 한편 프로세스의 상태 정보를 저장하고, 저장된 파일을 이용하여 프로세스의 수행을 복구하는 ckpt 도구도 있다[3]. 하지만 ckpt에서 생성된 파일에서 프로세스의 정보를 추출하는 것은 쉽지 않다.

## 3. 설계 및 구현

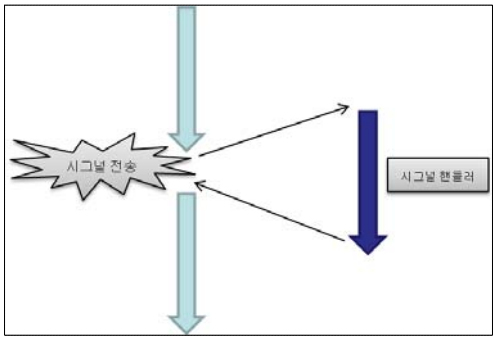
### 3.1. core 파일 생성

core 파일을 생성하는 방법은 여러 가지가 있다. 프로그램 수행 중 세그먼트 폴트 같은 오류가 발생할 경우 커널이 직접 core 파일을 만들어 주는 경우와 사용자의 의도에 따라 생성하는 경우가 있다. 하지만 이런 경우에는 단 하나의 core 파일만 생성되므로 한 시점의 정보만 분석 가능하다. 본 논문에서는 임의의 여러 시점에서 core 파일을 생성함으로써 여러 시점의 정보를 비교하고

자 한다.

### 3.1.1. Signal을 이용하는 방법

프로세스 조작과 관련이 깊은 시그널은, 커널 또는 프로세스가 프로세스 내/외부에서 발생하는 이벤트를 프로세스에게 전달하는 처리이다. 이러한 시그널이 프로세스에 전달되면 프로세스에서는 미리 정의한 시그널 핸들러가 실행된다. 시그널 핸들러를 처리하는 동안 프로세스의 수행은 중지되고, 시그널 핸들러가 종료되면 다시 기존의 프로세스 수행을 진행하게 된다[그림 1].

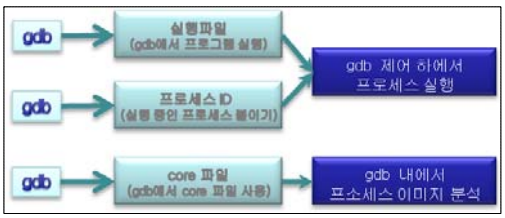



[그림 1] 시그널 처리 흐름

31개의 시그널 중 기본 동작(default action)으로 core 파일을 생성하는 시그널이 몇 있다. 가령 SIGQUIT / SIGILL / SIGTRAP / SIGABRT / SIGSEGV 등과 같은 시그널이 프로세스에 전달되면 시그널 핸들러에서 core 파일을 생성한다. 이런 시그널들은 대부분 프로세스 수행 중 오류와 관련된 내용이 많으므로 덤프를 수행한 이후 프로세스의 수행을 종료한다. 따라서 core 파일을 오직 하나만 생성하게 된다[5].

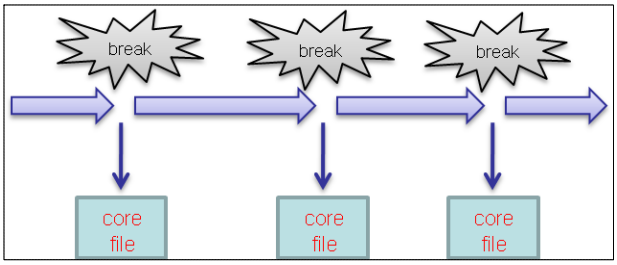
### 3.1.2. gdb를 이용하는 방법

gdb는 리눅스에서 가장 애용하는 디버거 툴로써 안정적이고 무료이다. gdb는 실행 중인 프로세스를 제어하거나 프로세스 이미지(core file)를 메모리에 올리도록 설계되었다.




[그림 2] gdb의 수행 방법

[그림 2]의 gdb의 수행 방법 중, gdb의 제어 하에서 프로세스를 수행하는 방법이 있다. gdb를 이용하여 프로그램을 시작하거나, 수행중인 프로세스의 id를 이용하여 gdb에서 수행하는 것이다. 두 가지 방법을 사용하면 gdb에서 원하는 대로 프로세스의 진행을 제어할 수 있다. 가령 gdb에서 프로세스의 수행 과정 중 원하는 지점에 중단점을 설정하여 수행을 진행하고, 중단점을 설정한 시점에서 프로세스의 수행이 멈추면 그 시점에서 core 파일을 생성할 수 있다.




[그림 3] gdb에서 core 파일 생성

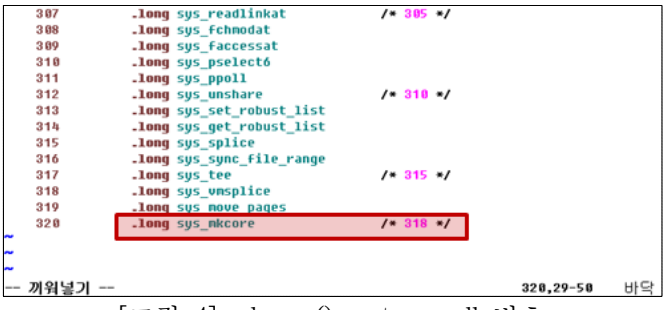
예를 들어, [그림 3]에서 프로세스 수행 중간에 세 번의 중단점이 있고 그 시점에 수행이 중단된다. 그때 gdb 명령 중 'generate-core-file'을 사용하여 core 파일 생성이 가능하다.

### 3.1.3. system call을 이용하는 방법

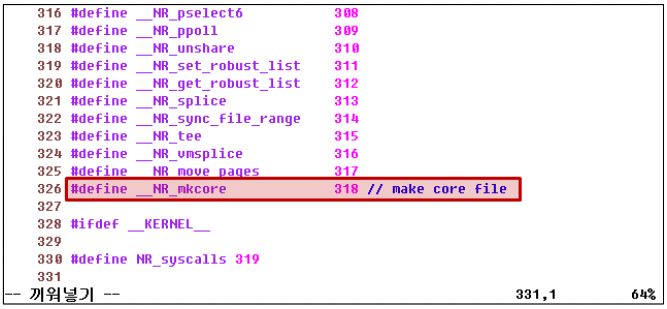
시그널을 이용하는 방법과 gdb를 사용하는 방법은 각각 단점이 있다. signal을 이용하는 방법은 오직 한 번의 core 파일을 생성하고 프로그램 수행을 종료하게 된다. 이런 경우 하나의 core 파일만 있기 때문에 한 번의 시점에 대한 분석은 가능하지만 프로세스 수행 흐름을 분석하기는 힘들다. 또한 gdb를 이용하는 경우 여러 개의 core 파일을 생성 가능하지만 gdb에 의존적으로 수행해야 하기 때문에 독립적으로 core 파일 생성이 불가능하다. 본 논문에서는 core 파일을 생성 가능하도록 mkcore() system call을 추가하였다. 이를 통해 응용프로그램 내에서 임의의 지점에서 core 파일을 생성하고 수행을 계속 진행하게 된다.

### 3.1.4. mkcore() system call

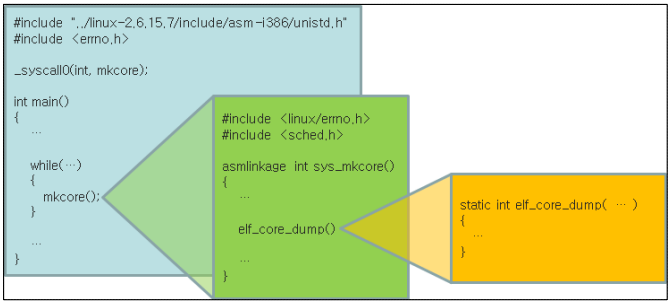
본 논문의 작업은 페도라 코어 6에서 수행하였다. linux.org에서 2.6.18.1의 커널 소스를 다운로드 하였고, mkcore() system call을 추가하였다.




[그림 4] mkcore() system call 번호




[그림 5] mkcore() system call 등록




[그림 6] mkcore() 수행

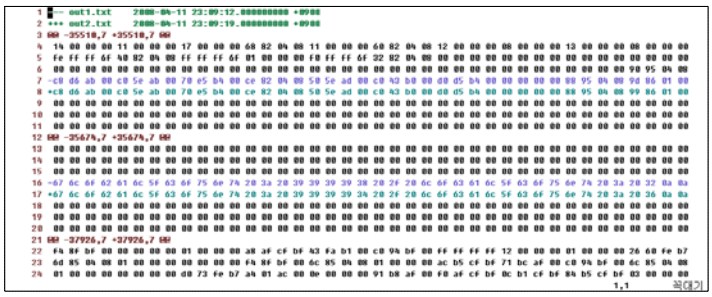
### 3.2 core 파일을 이용한 프로세스 수행 분석

#### 3.2.1 core 파일 확인

응용 프로그램 내에서 mkcore() system call을 호출하여 만들어진 core 파일들은 ELF 형식을 가지게 된다[6]. 생성되는 core 파일들은 생성된 순서에 따라 증가되는 번호를 파일 이름으로 갖는다. 일반적인 core 파일 이름은 "core.프로세스번호" 형태이지만, mkcore() system call을 통하여 생성하게 되면 "core.프로세스번호_버전번호" 형태로 만들어진다. 각 버전번호는 프로세스의 흐름에 따라 생성되는 순서를 통하여 식별자로써 부여된다.

#### 3.2.2. 프로세스 수행 분석

생성된 core 파일을 분석하기 위해서 gdb, objdump, readelf 와 같은 도구를 참조 하였다. 이러한 도구들은 core 파일에 대한 분석을 가능하게 하므로 프로세스 이미지의 정보를 추출하는데 편리하게 사용할 수 있다.

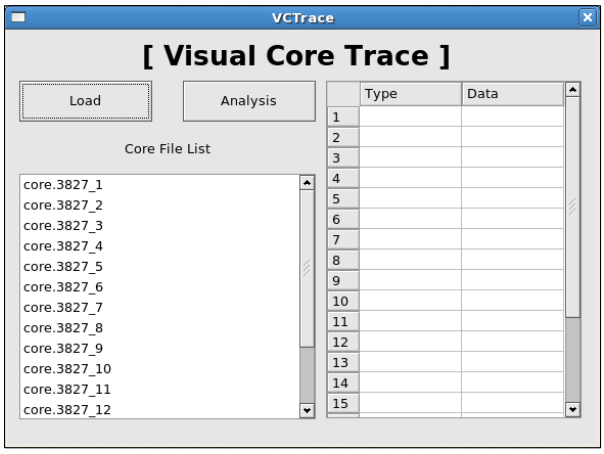


[그림 7] core 파일의 헥사 데이터

[그림 7]은 두 개 버전의 core 파일에 대한 비교 파일이다. 기본적인 영역 (ELF 헤더, 코드 영역 등) 은 변화하지 않고 데이터와 스택영역의 헥사 값이 변화하였다. 하지만 이렇게 헥사 파일로 분석해 나가는 것은 쉽지 않기 때문에 좀 더 비주얼한 도구의 필요성이 있다. 또한 응용 프로그램의 크기에 따라 생성되는 core 파일의 크기도 달라지기 때문에 헥사 데이터의 분석은 한계가 있다.

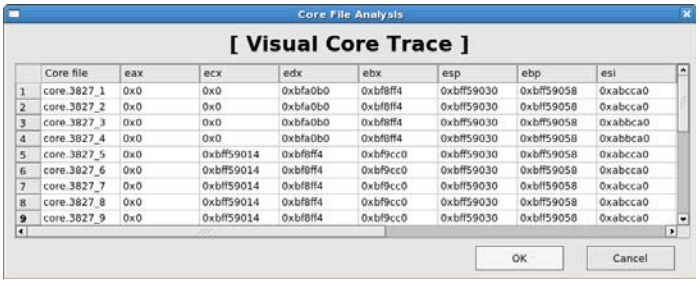
#### 3.2.3. VCTrace (Visual Core Trace)

core 파일 분석을 쉽게 할 수 있도록 qt를 이용하여 프로세스 분석 도구를 개발하였다. 비주얼 디버깅 도구인 "DDD"[7]를 참조하여 만든 해당 도구는 "core.프로세스번호" 파일을 인자로 전달하여 해당 프로세스의 모든 core 파일을 프로그램으로 불러들인다. 또한 생성된 순서에 따라 표현함으로써 프로세스의 수행 흐름 파악을 할 수 있도록 하였다. VCTrace는 레지스터 및 메모리에 대한 데이터를 표현하고 프로그램 카운터 등을 나타냄으로써 프로세스 수행 흐름 상황을 체크한다.




[그림 8] VCTrace 시작 화면

[그림 8]은 VCTrace를 시작한 화면이다. 그림을 보면 3827번 프로세스의 core 파일들을 모두 불러온 상태이다. 그 다음 하나의 core 파일을 선택하여 "Load" 버튼을 클릭하면 우측 리스트 박스 창에 정보들이 표시된다. 이 화면은 하나의 core 파일에 대한 정보 표시를 수행한다. 또한 "Analysis" 버튼을 클릭하면 [그림 9] 와 같은 다이얼로그 창이 생성되며, 모든 core 파일에 대해 정보를 나열하게 된다.




[그림 9] Core File Analysis

### 4. 결론

본 논문에서는 프로세스 수행 도중 생성한 core 파일들을 이용하여 프로세스 흐름 분석 도구를 설계 및 구현

하였다. 기존의 다른 도구들과 달리 지속적인 core 파일 생성에 의해 프로그램의 전체적인 수행시간은 다소 증가하였다. 또한 32비트 아키텍처를 기반으로 구현하였기 때문에 64비트 아키텍처에 대한 확장을 추가하여야 한다. 하지만 여러 시점의 core 파일을 통한 프로세스의 수행 흐름 분석이 가능하고, 비주얼적인 도구를 통하여 프로세스 정보를 쉽게 표현하도록 하였다.

**[참고문헌]**

[1] http://freshmeat.net/projects/ltrace/
[2] http://freshmeat.net/projects/strace/
[3] Jiman Hong, Taesoon Park, H.Y Yeom and Yookun Cho, Kckpt : An Efficient Checkpoint Facility on UnixWare, 15th International Conference on Computers and Their Applications, pp. 303-308, March 2000
[4] http://sourceware.org/gdb/
[5] Asim Shankar, A system for Process Checkpointing and Restarting(Using a core dump), April 19, 2003
[6] Executable and Linking Format (ELF) Specification , Tool Interface Standards (TIS)
[7] http://www.gnu.org/software/ddd/