Cell Phone Remote Debugger에서 Memory View 최적화 연구

이영준*, 문승진
수원대학교 컴퓨터학과
e-mail : yjlee*, sjmoon@suwon.ac.kr

A study on Memory view optimization in cell phone remote debugger

Young-Jun Lee*, Seung-Jin Moon
Dept of Computer Science, The University of Suwon

요 약

빠른 IT기술의 발전과 함께 휴대폰 시장 역시 대량생산에서 벗어난 차별적인 모델을 개발하기 위해 많은 개발자들이 연구 중에 있다. 개발자들이 빠르고 안정적인 응용 프로그램을 개발하기 위해 디버거를 필요로 하고 있다. 그러나 기존의 하드웨어 디버거를 이용하면 별도의 추가비용과 장비를 휴대해야 하는 단점이 있다. 따라서 본 논문에서는 이동성 보장과 디버거 장치 추가비용의 절감을 위한 소프트웨어 리모트 디버거의 기능중 하나인 메모리 뷰에 대해서 알아보고 큰 용량의 디버깅 작업을 빠르고 쉽게 처리 할 수 있는 메모리 뷰의 속도 향상 알고리즘에 관하여 논한다.

1. 서 론

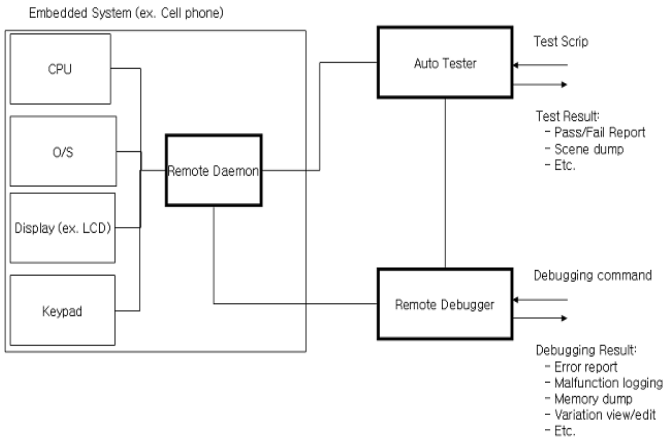
최근 IT기술은 빠르게 성장함과 동시에 다양하고 안정화된 제품들을 요구하는 시장에 직면해 있다. 휴대폰시장 역시 기존의 대량생산에서 벗어나 다양하고 차별적인 모델을 개발하는데 치중을 두고 있다. 따라서 휴대폰 응용프로그램 개발자들은 다수의 프로그램을 빠른 시간동안에 오류 없이 개발하기 위해 필요로 요구 되는 도구중 하나인 디버거를 사용한다. 그러나 하드웨어 디버거 장치를 이용함으로서 별도의 추가비용과 장비의 이동성 제약이 따른 다른 단점이 있다. 이에 소프트웨어 리모트 디버거는 하드웨어 디버거 장치의 단점을 보완하여 별도의 디버거 장치 없이 휴대폰과 PC만을 이용하여 휴대폰의 바이너리 정보와 ELF 파일을 이용하여 응용프로그램을 디버깅 할 수 있는 기능과 휴대폰의 *auto scripts*를 구성하여 시뮬레이션을 통한 오류를 찾아내 디버거와 연동하여 디버깅 할 수 있는 *auto tester* 기능이 구현된다.

본 논문에서는 리모트 디버거의 기능중 하나인 메모리 뷰의 느린 속도를 대처하기 위한 속도 향상 알고리즘을 논하기 위하여 2장에서는 리모트 디버거의 전체적인 구성과 세부적인 구현 방법에 대하여 서술하고, 3장에서는 본 논문에서 제안하고자 하는 *dump mode*의 구현 방법 중 *file reading time* 단축 알고리즘에 대하여 4장에서는 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 remote debugger

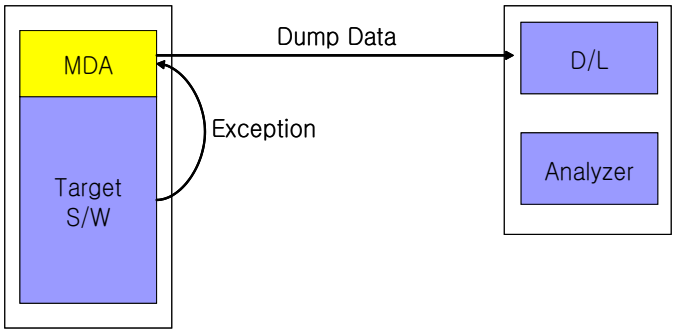
리모트 디버거의 구성은 브레이크된 휴대폰의 메모리를 덤프하여 생성된 바이너리파일과 이미 정의 되어진 ELF파일을 비교하여 memory view/edit, task / ISR call stack, ram data analysis, remote debugging등의 기능 등을 포함하는 리모트 디버거와 auto scripts를 작성하여 휴대폰의 버그를 찾아낼 수 있는 auto tester로 구성되어 있다.



(그림 1) remote debugger overview

2.2 dump mode

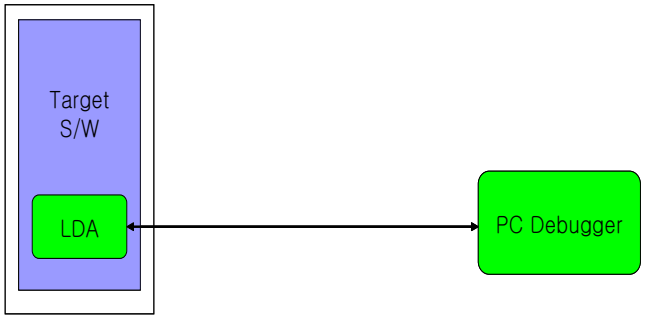
덤프모드는 휴대폰의 램 정보를 추출하고 ELF 파일의 분석을 통해 *elf header, section header, program header, symbol table*등의 정보를 기반으로 휴대폰에서 덤프해온 바이너리 파일의 *task / ISR call stack address* 및 *memory / watch view* 및 레지스터정보 같은 구체적인 정보를 트리형식으로 표현해 주는 것을 구현한다.



(그림 2) dump mode

2.3 live mode

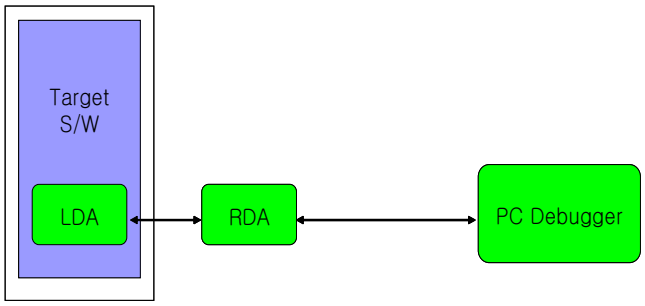
라이브 모드는 덤프모드에서 구현된 ELF 정보를 바탕으로 휴대폰이 동작중인 상태에서 PC와 연동하여 memory/watch view, screen capture, breakpoint /single step, ram data analysis를 구현하여 중대한 버그 발생 시 휴대폰을 끄지 않은 상태에서 램 데이터만 덤프하여 사용 오류를 찾아낼 수 있으며, 찾은 데이터를 실시간으로 수정하여 휴대폰에 적용하는 기능이다.



(그림 3) live mode

2.4 remote live mode

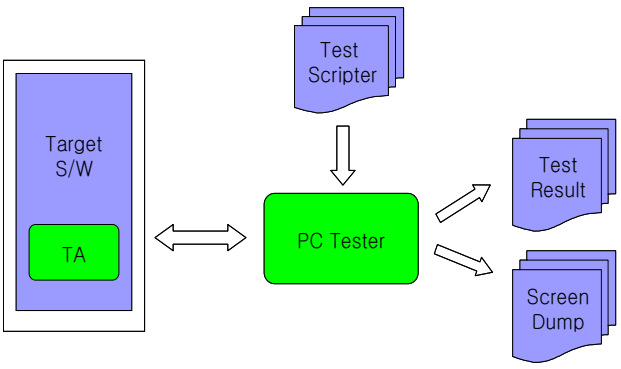
리모트 라이브 모드는 라이브 모드의 기능에서 원거리 네트워크상에서 두 대의 PC를 이용하여 리모트 디버거의 모든 기능을 이동, 공간 제약 없이 사용할 수 있도록 하는 리모트 디버거의 최종 목표이다.



(그림 4) remote live mode

2.5 auto tester

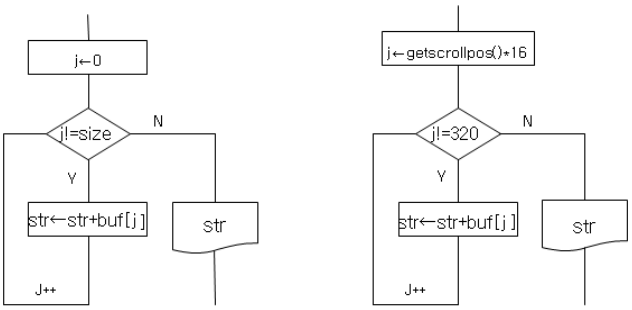
휴대폰의 동작을 시뮬레이터를 이용하여 동일한 동작을 반복함으로써 에러를 체크 하는 것으로 test scripter를 통해 스크립트 파일을 실행하는 기능으로, 기존 사람의 손으로 휴대폰의 모든 기능 또는 단순 반복 기능 등을 시뮬레이터로 대신함으로써 개발자들의 작업능률과 자원 낭비를 방지할 수 있는 기능이다.



(그림 5) auto tester

3. memory view file reading time 단축 알고리즘

3.1 memory view의 algorithm



[수정전] [수정후]

(그림 6) memory view algorithm

그림 6은 메모리 뷰의 구현을 알고리즘으로 나타낸 것이다. 수정전에는 *j*가 0부터 파일의 크기인 *size*까지 변하면서 *str*에 파일내용을 저장하는 방법이고, 수정후에는 *j*에 스크롤값을 읽어 들이는 *getscrollpos()* 값과 16을 곱하여 파일에서의 위치를 계산한값이 저장되어 한 페이지의 크기 값 만큼 변하면서 *str*에 파일내용을 저장하여 출력하는 방법이다.

3.2 실험환경

<표 1> 실험환경

분 류	내 용
C P U	intel P4-2.8
R A M	1024MB
V G A	geforce 6200
O S	windows xp
language	VC++ 6.0

본 논문의 실험에 사용한 환경은 표 1과 같이 CPU는 인텔사의 펜티엄4-2.8G를 사용하였고 램은 1024MB, 그래픽카드는 지포스 6200을 운영체제는 윈도우XP, 언어는 비주얼 C++ 6.0을 사용하였다.

3.3 memory view 구현

<표 2> memory view code(수정전)

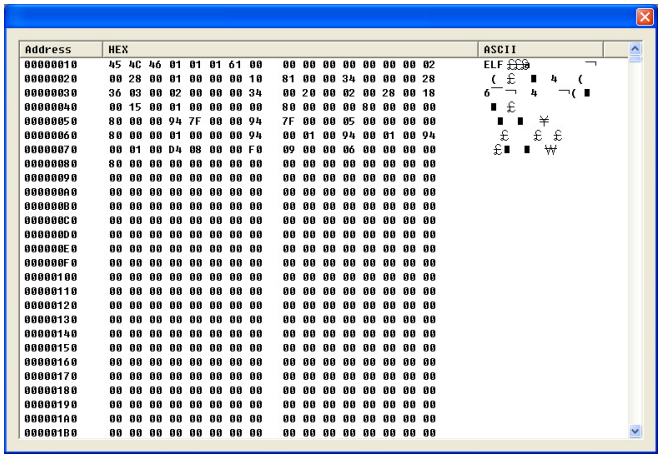
```
fseek(fp, 0, seek_end);
size = ftell(fp);
buf = new char[size];
memset(buf, 0, size);
fseek(fp, 0 , seek_set);
fread(buf, size, 1, fp);
lv_item li;
li.mask=lvif_text;
li.state=0;
li.statemask=0;
li.iimage=0;
li.iitem=0;
for(int j = 0; j < size; j++)            // 0부터 파일의 크기만큼 반복
{
        if((int)buf[j]<0) {
        int temp = (int)buf[j]+256;
        str1.format("%.2x ",temp);
        }
        else        str1.format("%.2x ",buf[j]);
        str2 += str1;
        if ((int)buf[j]==00)
        {
                str3 += " ";
        }
        else
        {
                str1.format("%c",buf[j]);
                str3 += str1;
        }
        if((j+9)%16 == 0) str2 += "  ";
        if((j+1)%16 == 0) {
        str1.format("%.8x",j+1);
        li.isubitem=0;
        li.psztext = (lptstr)(lpctstr)str1;
        m_pdumpdlg->m_listctrl.insertitem(&li);
        li.isubitem=1;
        li.psztext = str2.getbuffer(str2.getlength());
        m_pdumpdlg->m_listctrl.setitem(&li);
        li.isubitem=2;
        li.psztext = (lptstr)(lpctstr)str3;
        m_pdumpdlg->m_listctrl.setitem(&li);
        str1 = "";
        str2 = "";
        str3 = "";
        li.iitem++;
        }
}
updatedata(false);
delete buf;
fclose(fp);
```

<표 3> memory view code(수정후)

```
fseek(fp, 0, seek_end);
size = ftell(fp);
buf = new char[size];
memset(buf, 0, size);
fseek(fp, 0, seek_set);
fread(buf, size, 1, fp);
int temp=0;
lv_item li;
li.mask=lvif_text;
li.state=0;
li.statemask=0;
li.iimage=0;
li.iitem=0;
unsigned long dd=size/16;                          // 파일크기 /16
m_scroll.setscrollrange(0,dd);                     // 스크롤바범위지정
m_scroll.setscrollpos(0);                          // 스크롤바초기위치
scroll=m_scroll.getscrollpos()*16;                 // 스크롤바값*16
m_listctrl.deleteallitems();                       // 화면지움
for(unsigned long i = scroll; i<(scroll+320); i++) // scroll부터 화면크기
                                                      만큼 반복
{
        if((int)buf[i]<0)
        {
                int temp = (int)buf[i]+256;
                str1.format(" %.2x ",temp);
        }
        else        str1.format(" %.2x ",buf[i]);
                        ⋮
                      중략
                        ⋮
```

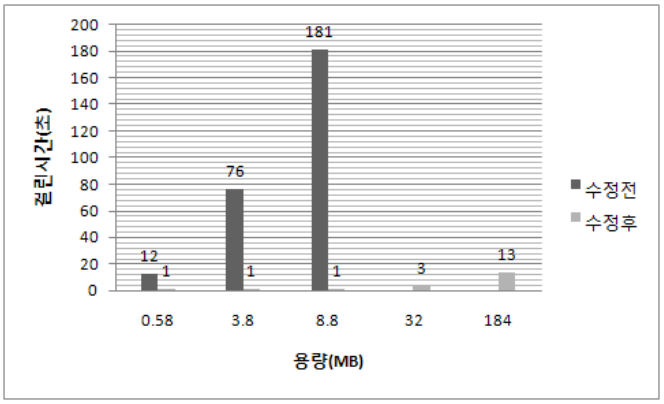
덤프모드의 메모리 뷰를 앞의 그림 6의 알고리즘에서 설명했듯이 두 가지 방법으로 구현해 보았다.

ELF 파일의 크기를 *size*에 파일의 내용을 *buf*에 저장하여 수정전의 방법 표 2은 전체적으로 출력을 하고 수정후의 방법 표 3은 한화면 크기만큼을 출력하는 방법이다. 두 가지 방법이 동일한 출력 그림 7을 나타내지만 출력이 나오는 속도 면에서 수정후의 방법으로 구현했을 때가 더 빠르게 나온다.



(그림 7) memory view 출력화면

3.4 수정전・후 결과 비교



(그림 8) 수정전・후의 시간

그림 8에서 보는바와같이 첫 번째 방법에서는 598KB를 읽는데 12초, 3.85MB는 76초, 8.8MB는 181초 걸렸으나 뒤의 방법으로 했을 때 3.85MB는 1초, 8MB는 1초, 32MB는 3초 184MB는 13초가 걸렸다. 첫 번째 방법으로 했을 경우 8.8MB는 181초가 걸렸으나 두 번째 방법의 경우는 단 1초밖에 걸리지 않았다.

리모트 디버거의 ELF 파일의 크기는 184MB이다. 이 파일을 읽기 위해서는 수정전의 방법으로는 불가능했고 수정후 방법으로 읽었을 경우 13초가 걸렸다.

이 결과 덤프 모드에서 메모리 뷰를 구현하기 위해서는 두 번째 방법이 훨씬 뛰어나다고 볼 수 있다.

4. 결론 및 향후 연구 과제

본 논문에서는 임베디드 시스템의 하나인 휴대폰의 디버깅을 위한 리모트 디버거의 전체적인 구성과 기능구현에 대하여 서술하고 구현기능중 하나인 메모리 뷰의 시간단축 방안에 대한 알고리즘에 대해 논하였다.

향후 구성된 기능 구현 및 테스트를 통한 리모트 디버거는 하드웨어 장비 없이 휴대폰과 PC만으로 디거깅을 할 수 있게 되어 휴대폰 응용프로그램 개발자들에게 많은 도움이 되리라 생각된다. 또한 개발 비용과 시간이 단축됨으로써 휴대폰 시장이 한층 더 발전 할 수 있을 것이다.

참고문헌

[1] Mark Wilding, Dan Behman, Self-service-linux. 박재호, 이해영역, “리눅스 문제 분석과 해결”, 에이콘출판사, 2006.
[2] TIS Committee, “Executable and Linking Format (ELF) Specitication Ver.1.2”, May 1995
[3] TIS Committee, “Debuggin Information Format (DWARF) Specification Ver 2.0”, May 1995
[4] Daniel Jacobowitz, “Remote Debugging with GDB”, http://www.kegel.com/linux/gdbserver.html, 2002
[5] Minheng Tan, “A minimal GDB stub for embedded remote debugging”, http://www1.cs.columbia.,edu/~sedwards/classes/2002/w4995-02/tan-filan.pdf,2002
[6] Richard M. Stallman, “Debugging with GDB, 4th ed.”, Cygnus Support, 1996
[7] “Korea Embedded Linux Project”, http://www.kelp.or.kr
[8] “KLDP”, http://www.kldp.org