비휘발성 메모리 영역 사용을 위한 코드 재생성

이성수[O] 정재민 원유집

한양대학교 컴퓨터·소프트웨어학과

su880214@hanyang.ac.kr, jmjung@hanyang.ac.kr, yjwon@hanyang.ac.kr

Code-Regeneration Method for Non-Volatile Memory Library

Seongsu Lee[O] Jeamin Jung Youjip Won

Department of Computer and Software, Hanyang University

요 약

기존 운영체제의 프로그램이 비휘발성 메모리를 사용하기 위해서는 영속 객체 라이브러리를 사용해야 한다. 본 논문에서는 기존 프로그램 코드를 영속 객체 라이브러리 적용한 코드로 재생성하기 위한 기법을 제시한다. 그리고 라이브러리 적용 코드 재생성 알고리즘을 작성하여 코드 재생성 과정을 자동화할 수 있도록 했다.

1. 서 론

비휘발성 메모리는 바이트 단위 접근이 가능하고, 전원이 차단되어도 데이터를 유지할 수 있다. 이러한 특성은 기존 컴퓨터의 DRAM과 디스크를 사용한 메모리-스토리지 계층 구조를 비휘발성 메모리를 사용한 DRAM 또는 디스크를 대체하는 계층 구조나 하나의 단일 계층 구조를 가능하게 한다. Spin toruqe transfer magne toresistive RAM (STT-MRAM), Ferroelectric RAM (FRAM), Phase-change memory (PCM) 등의 비휘발성 메모리 소자가 개발되었고, 비휘발성 메모리를 위한 프로그래밍 인터페이스 및 파일 시스템 소프트웨어 기술도 개발되었다[1].

기존 운영체제의 가상메모리, 물리메모리 관리, 페이징 등의 메모리 기술은 휘발성 메인 메모리와 메모리-디스크 계층 구조의 영향을 받았다. 현재 운영체제에서 비휘발성 메모리를 사용하려면 비휘발성 메모리를 위한 인터페이스가 필요하며, 이를 위해 여러 영속 객체 라이브러리 및 인터페이스가 개발 되었다. 기존 프로그램이 비휘발성 메모리에서 동작하려면 영속 객체 라이브러리가 코드에 적용되어야 한다. 본 논문에서는 기존 DRAM 기반 프로그램 코드에 영속 객체 라이브러리를 적용하기 위한 기법을 제시하고, 이를 자동화하는 툴 구현에 대해 다룬다.

2. 배 경

영속 객체 라이브러리 중 Heap-based persistent object store (HEAPO)는 libc 기반의 동적 메모리 할당/해제 함수와 흡사한 인터페이스를 제공하여 기존 프로그램 코드에 라이브러리를 적용하기 용이하다[2]. HEAPO는 64bit 어드레스 스페이스에서 Heap에 사용될 영역의 일부인 0x5FFEF8000000-0x7FFEF8000000 영역을 영속성을 보장하는 메모리 공간인 영속 Heap으로 정의한다. 영속 Heap영역에는 영속 객체 저장소가 존재한다. 영속 객체 저장소는 이름을 갖는 메모리 영역이며, 유저 프로그램의 메모리 할당 요청에 의해 영속 객체 저장소의 일부를 할당해준다. HEAPO 라이브러리가 적용된 프로그램은 비휘발성 메모리를 사용하여 영속 객체 생성할 수 있다.

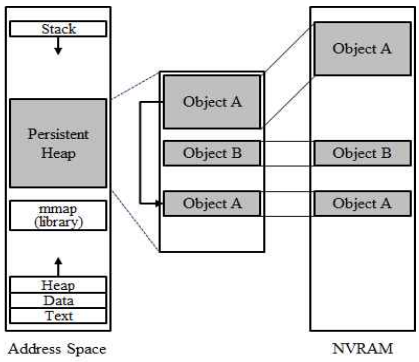

그림 1. 영속 Heap과 비휘발성 메모리

3. 영속 객체 라이브러리 적용

본 논문에서는 기존 코드의 모든 동적 객체 할당/해제를 위한 malloc( )과 free( )에 대하여 영속 객체 라이브러리를 적용하는 것이 아닌 비휘발성 메모리에 적합한 객체를 선별하여, 영속 객체 라이브러리를 적용한다. 영속성을 부여할 객체를 선별하기 위해 코드 프로파일러를 사용했다[3]. 사용된 코드 프로파일러는 동적 객체가 DRAM과 STT-MRAM에 할당 되었을 경우를 가정하여 각각의 읽기/쓰기 에너지 소모량을 구하고, 차이를 구한다. DRAM의 읽기/쓰기 에너지 소모량은 약 0.7 nJ이며, STT-MRAM은 읽기가 0.4 nJ, 쓰기가 2.3 nJ이다. 프로파일러는 객체들의 읽기/쓰기 엑세스에 대해 DRAM과 STT-MRAM에서의 에너지 소모량을 비교하여 STT-MRAM에서의 에너지 소모량이 더 적은 객체는 비휘발성 메모리에 적합하다고 판단한다.

프로그래밍에서 영속 객체 라이브러리는 기존의 동적 메모리 관리 라이브러리에 대응되는 역할을 하지만 인터페이스가 다르다. 영속 객체 할당/해제를 위해 byte 단위 메모리 사이즈와 포인터 변수 이외에도 유저가 명시적으로 지정한 객체 저장소의 이름을 사용한다. 그리고 영속 객체를 할당/해제를 위해서는 먼저 객체 저장소를 생성해야 한다. 영속객체 라이브러리의 메모리 할당/해제 인

터페이스는 pos_malloc(char *name, size_t size)와 pos_free(char* name, void* ptr)이고, 영속성을 갖는 객체 저장소 생성 인터페이스는 pos_create(char* name)이다. pos_malloc( )은 malloc( )처럼 할당 한 메모리 영역의 시작 주소를 리턴한다. 기존 코드에서 영속 인터페이스를 적용하기 위해서는 객체 저장소의 이름을 설정하고, 기존 코드의 malloc( )과 free( ) 대신 pos_malloc( )과 pos_free( )을 적용한다. pos_malloc( )이 처음으로 호출되기 이전에 pos_create( )가 호출되어 객체 저장소를 생성하기 위해 프로그램 main( )의 시작에 pos_create( ) 호출 코드를 생성한다. 만약, 생성하고자하는 객체저장소와 같은 이름의 객체 저장소가 이미 생성되어 있거나, 객체저장소 생성이 실패 했을 경우에는 pos_create( )의 리턴 값을 확인하여 pos_map( )을 호출한다. pos_map(char* name)은 기존의 생성한 객체 저장소를 가상주소공간에 사상한다.

4. 코드 재생성

프로파일러는 영속 객체 라이브러리를 적용 할 동적 할당 코드에 대한 정보를 출력한다. 정보에는 malloc( )이 위치한 파일 이름 및 라인 등을 포함 하고 있기 때문에 코드 파싱 없이 코드 재생성이 가능하다. pos_malloc( )를 사용한 영속 객체 할당 코드 재생성이 완료되면 객체 해제 및 객체 저장소 생성 코드를 재생성하기 위해 프로그램의 모든 소스 파일에 대해 main( )과 free( )를 탐색한다. main( )과 free( ) 탐색은 정규표현식을 사용한다[4]. *, +, ^, ¦, ? 등의 메타문자를 사용한 정규표현식을 설정하면 변수명, 함수명, 주석 등에 사용된 main, free 문자열을 필터링이 가능하다. 탐색과정에서 main( )을 발견하면 pos_create( )와 pos_map( ) 코드를 생성한다.

Algorithm 1 Code regenerating for non-volatile memory library

```
1: function malloc_substitution(malloclist)
2:     if malloclist is null then return NULL
3:     end if
4:     /* Each item of malloclist includes file name, line number of malloc( ) */
5:     for each item of malloclist
6:         get file from item
7:         get line from item
8:         substitute malloc() at line in file to pos_malloc()
9:         insert pos header in file
10:    end for
11: end function
12:
13: function main_free_substitution( )
14:     for each file in current directory and subdirectories
15:         modification ← false
16:         if file includes main() then
17:             get line of main() from file
18:             insert pos_create() at line in file
19:             modification ← true
20:         end if
21:         if file includes free() then
22:             index ← 0
23:             for each free() in file
24:                 line[index] ← linenumber of free()
25:                 index ← index + 1
26:             end for
27:             index ← 0
28:             for each line[index]
29:                 get argument of free() at line[index]
30:                 set pos_free() statement with argument
31:                 substitute free() at line[index] in file to pos_free() statement
32:                 index ← index + 1
33:             end for
34:             modification ← true
35:         end if
36:         if modification then
37:             insert pos header in file
38:         end if
39:     end for
40: end function
```

그림 2. 코드 재생성 알고리즘

영속 객체 해제 코드 재생성은 해제하려는 객체가 영속객체인지 일반 객체인지 확인이 필요하다. 프로파일링 기반 코드 재생성은 프로그램 코드내에서 malloc( )과 pos_malloc( )이 혼재된다. 라이브러리를 적용한 프로그램의 동적 할당 메모리 해제는 해제하고자하는 메모리 영역의 주소 값에 따라 free( ) 또는 pos_free( )를 선택적으로 호출하는 코드를 생성해야 한다. 영속 Heap영역은 주소 값으로 일반 Heap영역과 구분이 가능하다. free( )의 파라미터로 사용되는 포인터형 변수의 이름을 추출하여, 조건문을 사용하는 코드를 생성한다. 라이브러리 적용으로 코드가 재생성된 파일에는 라이브러리 헤더를 인클루드하는 코드를 생성한다. 그림 2는 코드 재생성 알고리즘이다. 알고리즘 구현은 텍스트처리에 적합한 쉘 스크립트 언어를 사용했다.

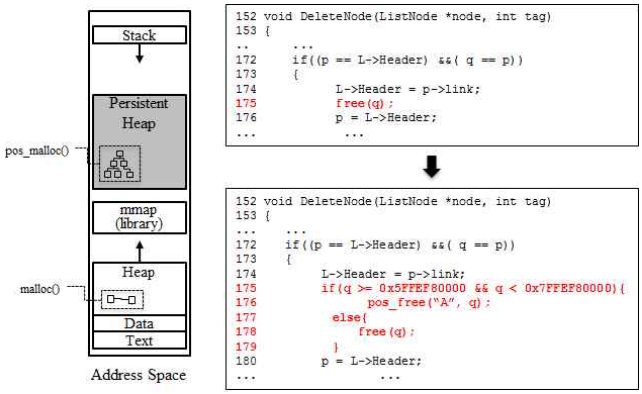

그림 3. 영속 객체 해제 코드 재생성

5 결론 및 향후 연구

본 논문의 코드 재생성 기법은 libc 동적 메모리 할당/해제와 유사한 인터페이스를 제공하는 영속 객체 라이브러리에 대해 적용 및 응용이 가능 할 것으로 기대된다.

제시한 코드 재생성 기법은 비휘발성 메모리에 할당된 객체의 재사용을 고려하지 않는다. 비휘발성 메모리의 영속 객체는 프로세스의 수명과 관계없이 유지되고 재사용이 가능하다. 객체 재사용을 고려한 코드 재생성은 비휘발성 메모리 어플리케이션 프로그래밍 모델 연구가 필요하다.

Acknowledment

본 연구는 지식경제부 및 한국산업기술평가관리원의 산업원천 기술개발사업(정보통신)의 일환으로 수행하였음. [No.10041608, 차세대 메모리 기반의 스마트 디바이스용 임베디드 시스템 소프트웨어]

참고 문헌

[1] Katelin Bailey, et al. “Operating system implications of fast, cheap, non-volatile memory“ in HotOS'13 Proceedings of the 13th USENIX conference on Hot topics in operating systems, Pages 2-2

[2] Taeho Hwang, et al. “HEAPO: Heap-Based Persistent Object Store“ in ACM Transactions on Storage(TOS), Volume 11 Issue 1, February 2015.

[3] System design methodology, Newmemory profiler https://github.com/sdmlab/NVRAMPF

[4] Bruce Barnett, UNIX Regular Expression.http://www.grymoire.com/Unix/