모델기반의 커널 테스팅 프레임워크
(MOdel-based KERnel Testing (MOKERT) Framework)

김 문 주 † 홍 신 ††
(Moonzoo Kim) (Shin Hong)

요 약 최근 내장형 시스템이 점점 많은 분야에 사용되며, 시스템에 특화된 운영체제 커널에 대한 필요성이 커지고 있다. 하지만, 커널 개발은 코드의 복잡성 등의 이유로 말미암아 테스팅에 큰 비용이 소요됨에도 불구하고, 높은 신뢰성을 달성하기가 어려운 실정이다. 이러한 커널 개발 및 테스팅의 어려움을 극복하기 위해, 운영체제 커널의 동시성 오류 검출을 지원하는 모델 기반의 커널 테스팅 (MOKERT) 프레임워크를 제안한다. MOKERT 프레임워크는 주어진 C 프로그램을 Promela 정형 명세 모델로 변환하고 나서 Spin 모델검증기를 사용하여 검증하고, 검증반례가 생성된 경우, 이 검증반례를 실제 커널 코드에서 실행을 시켜서 진위를 확인한다. 본 연구에서는 MOKERT 프레임워크를 리눅스 proc파일시스템에 적용하여, ChangeLog에 보고된 오류가 실제로 자원경쟁문제를 일으킴을 확인하였을 뿐만 아니라, 커널 패닉을 일으키는 새로운 오류도 발견하였다.

키워드 : 모델검증 기법, 테스팅, 검증반례 분석, 모델추출

Abstract Despite the growing need for customized operating system kernels for embedded devices, kernel development continues to suffer from insufficient reliability and high testing cost for several reasons such as the high complexity of the kernel code. To alleviate these difficulties, this study proposes the MOdel-based KERnel Testing (MOKERT) framework for detection of concurrency bugs in the kernel. MOKERT translates a given C program into a corresponding Promela model, and then tries to find a counter example with regard to a given requirement property. If found, MOKERT executes that counter example on the real kernel code to check whether the counter example is a false alarm or not. The MOKERT framework was applied to the Linux proc file system and confirmed that the bug reported in a ChangeLog actually caused a data race problem. In addition, a new data race bug in the Linux proc file system was found, which causes kernel panic.

Key words : model checking, testing, counter example analysis, model extraction

· 본 연구는 2008년 교육과학기술부의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구이며(KRF-2008), 교육과학기술부/한국과학재단 우수연구센터 육성사업의 지원(R11-2008-007-03002-0)과 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업(IITA-2009-(C1090-0902-0032))의 연구결과로 수행되었습니다.
· 이 논문은 2009 한국소프트웨어공학학술대회에서 '검증 반례 재연을 통한 모델 기반 커널 테스팅'의 제목으로 발표된 논문을 확장한 것이며, 추가성과는 Model-Based Testing 2009에 발표하였음

† 종신회원 : KAIST 전산학과 교수
moonzoo@cs.kaist.ac.kr
†† 학생회원 : KAIST 전산학과
hongshin@gmail.com
논문접수 : 2009년 3월 16일
심사완료 : 2009년 5월 15일

정보과학회논문지 : 소프트웨어 및 응용 제36권 제7호(2009.7)

1. 서 론

소프트웨어 테스팅은 일반적으로 소프트웨어의 총 개발 기간 및 가용 자원의 50% 이상을 차지한다. 여러 소프트웨어 시스템 가운데 운영체제 커널은 다음과 같은 원인으로, 개발 및 분석이 어려운 소프트웨어 중 하나로 알려졌다.

- 코드의 복잡함
  운영체제 커널은 다양한 하드웨어 구성요소(CPU, 메모리, 컴퓨터 주변장치)와 소프트웨어 구성요소(장치 드라이버 모듈, 시스템 콜 라이브러리)를 관리하기 위해서 많은 수의 서비스 루틴을 가지고 있다. 그뿐만 아니라, 커널은 성능향상을 위해 복잡한 형태의 최적화된 알고리즘을 사용한다.
- 동시성으로 말미암은 동작 경우의 수 증가
  커널은 다중 스레드 프로그램이므로 동시성 오류(con-

currency bug)를 일으킬 수 있으며, 특정한 동시성 오류는 특정 스케줄링 시나리오에 의해서만 발생한다. 동시에 수행되는 스레드의 개수에 따라서 실행 가능한 스케줄링 시나리오의 수는 지수 적으로 증가하며, 대부분의 커널에서 사용자는 프로세스 스케줄러를 정교하게 제어할 수가 없다. 이러한 이유로 커널의 동시성 오류를 발견하고 수정하는 것은 어렵다.

- 유닛 테스팅의 어려움

대부분의 커널은 성능의 향상을 위해 단일(monolithic) 커널 구조를 사용한다. 그러므로 커널의 유닛을 독립적으로 테스팅하기 위해서는, 커널 전체의 구현에 대한 이해를 요구하여, 많은 비용이 소요된다. 예를 들어, proc 파일 시스템을 테스트하려면 가상 파일 시스템, 메모리 관리, 스케줄러 등에 관한 실행환경을 테스팅에 적합하게 구성하는 과정이 필요하다. 그 이유는 앞서 언급한 부분들이 proc 파일 시스템의 수행에 밀접한 관계가 있기 때문이다. 위와 같이, 커널의 특정 부분을 유닛 테스팅 하려면 실행환경을 상세하고 적합하게 구성해야 하며, 이는 때때로 유닛 테스팅 그 자체보다 더 큰 노력을 요구한다.

적합한 도구의 부재

커널 테스팅을 지원하는 도구들이 적은 이유는, 테스팅 툴에 의해 발생하는 부작용으로 커널이 멈추거나, 예상치 못한 작동을 일으킬 가능성이 있기 때문이다. 또한, 일반적인 테스팅 도구는 커널 환경에서는 사용할 수 없는 라이브러리에 의존하는 경우가 많아서, 커널 테스팅에 적합하지 않다. 이러한 이유로 커널 개발자들은 커널을 분석할 때 printk()나 커널 로그에 의존하는 경우가 대부분이다.

커널의 복잡성과 크기를 고려할 때, 커널을 구성요소 단위로 분석하는 것이 필요하다. 하지만, 위에서 언급한 이유로 말미암아 구성요소 단위로 커널을 테스팅하는 것은 실질적으로 어렵다. 결과적으로 전통적인 테스팅 방법은 커널을 개발 및 분석하는 데 있어 큰 도움을 주지 못하며, 이로 인해 커널은 특정 개발자들만이 개발할 수 있는 영역으로 남아 있다. 하지만, 내장형 시스템이 일상생활 모든 영역에 널리 사용되면서 특정 목적을 위한 운영체제(휴대폰 또는 센서 네트워크를 위한 운영체제) 개발의 필요성이 점차 증가하고 있다. 그러므로 커널의 구성요소 단위 분석을 지원하는 프레임워크를 개발하는 것이 중요하다.

본 연구는 모델추출[1]과 모델검증 기법[2]을 이용해 커널 상의 동시성 오류를 검출하는 모델 기반의 테스팅 프레임워크를 개발하는 것을 목적으로 한다. 앞서 언급한 이유로 말미암아 전통적인 방식의 테스팅으로 커널을 검증하는 것은 현실적으로 어려움이 많기에, 모델검증 기법이 커널을 분석하는데 대안이 될 수 있다. 그 이유는, 모델검증 기법은 분석하고자 하는 커널의 구성요소와 그와 관련된 환경을 추상화를 통하여 모델링하므로, 복잡한 커널을 구성요소 단위로 독립적으로 분석하는 것이 가능하다. 게다가 모델검증 기법은 모든 가능한 실행 시나리오를 분석하고, 구체적인 검증반례(counter example)를 제시해 주기 때문에, 커널을 분석하는데 유용하다. 하지만, 추상 모델과 실제 프로그램 간의 차이로 말미암아 모델검증 기법이 제시한 검증반례가 실제 프로그램상의 에러인지, 아니면 오경보(false alarm)인지의 여부를 개발자들이 판단하기는 어려우므로, 순수한 모델검증 기법만으로는 커널을 검증하는 데 문제가 있을 수 있다.

본 논문에서 제시하는 모델기반의 커널 테스팅 프레임워크는 모델검증 과정에서 발견한 검증반례를 모델이 아닌, 실제 운영체제 커널 코드의 실행에서 재연하는 기능을 제공한다. 이 기능을 통해 모델 기반의 커널 테스팅 프레임워크는 커널 개발자들에게 모델검증 기법과 테스팅의 장점을 동시에 제공할 수 있다. 이 기능은 모델검증에서 발견된 오류의 원인 분석, 결함 패치(bug patch)의 적합성 확인, 추상화 모델의 정밀화 등의 작업을 가능하게 한다. 따라서 모델 기반의 커널 테스팅 프레임워크의 사용은, 비록 추상화 모델과 검증반례를 재연하기 위한 실행환경을 구성하기 위한 작업이 필요하지만, 커널의 분석에서 전통적인 테스팅 기법의 어려움을 극복할 수 있다.

2. 관련 연구

커널과 같은 동시성을 갖는 프로그램의 신뢰성을 향상하려는 방법 중, 동시성 프로그램 테스팅의 경우, 가능한 많은 프로그램 수행 경우를 테스트 하려고 노력하고 있다. [3]에서는 문맥전환(context switching)이 일어나는 시점마다 무작위적으로 실행될 스레드를 고르는 알고리즘을 제안하고, 실험을 통하여 이 알고리즘을 이용한 테스팅이 운영체제 스케줄링에 의존한 테스팅 기법보다 더 많은 서로 다른 수행시나리오를 검사하기 때문에, 프로그램의 동시성 오류를 찾을 확률을 높일 수 있음을 보이고 있다. 하지만, 동시성 프로그램 테스팅의 경우 오류의 검출이 확률적이라는 근본적인 단점이 있다. 본 연구에서 제안하는 모델기반의 커널 테스팅(MOdel-based KERrnel Testing(MOKERT)) 프레임워크는, 모델검증 도구를 사용하여 모든 수행 가능 시나리오가 요구사항을 만족하는지를 검사하기 때문에, 확률적 테스팅 기법보다 높은 신뢰성을 제공할 수 있다. [4]에서는 메시지 전달이나 세마포어 등의 명백한 동기화(synchronization) 메커니즘을 사용하는 경우, 하나의

```
...
44: proc 1 (proc_readdir)       line 116 [(  (!i)  )]
45: proc 1 (proc_readdir)       line 125 [((proc_subdir_lock==1))]
46: proc 2 (remove_proc_entry) line 156 [((proc_subdir_lock==0))]
47: proc 2 (remove_proc_entry) line 157 [p = proc_parent.next]
...
```

그림 1 모델검증 도구 SPIN이 생성하는 반례의 예

수행시나리오를 기반으로, 자원경쟁(data race)을 일으킬 수 있는 여러 수행시나리오를 변형하여 생성하여 테스트하는 RichTest라는 툴을 구현하고 있으나, 실제 동시성 프로그램에서는 공유 변수를 통한 간접적인 동기화 사용이 빈번하므로 실제적인 효용성이 크지 않다.

또한, 다중 스레드 프로그램의 효과적인 분석과 디버깅을 위하여, 실제 프로그램의 실행을 기록하고, 이를 재연하는 연구가 진행되고 있다. [5]에서는 자바 프로그램의 실행을 기록하고 다시 재연하는 가상기계를 구현하였다. 가상기계 상에서 다중 스레드 자바 프로그램을 테스트하고, 오류 실행이 발견되는 경우, 오류 실행을 재연함으로써 결함을 분석하고 디버깅할 수 있도록 하였다. [6]과 [7]에서는 프로그램의 오류가 일어나기 전 시점까지의 프로그램의 실행을 기록하고, 이를 재연하여 프로그램 오류의 원인을 분석할 수 있도록 지원하는 효과적인 알고리즘을 제시하고 있다. 이와 같은 연구들에서는 테스트 중에 관찰이 가능한 오류에 대해서만 재연 기능을 제시할 뿐이므로, 만일 테스팅 과정 중 오류가 발견되지 않는 경우, 프로그램에 오류가 없음을 확신하지 못한다. 하지만, 본 연구에서 제안하는 MOKERT 프레임워크는 모델검증을 통해 모든 수행 가능한 시나리오를 분석하여, 모든 오류를 검출 및 재연할 수 있으므로 위의 연구들과는 차별성을 갖는다.

위에 언급한 연구들의 약점인, 효과적인 테스트 케이스생성의 어려움을 극복하고자 제안된 일반적인 모델 기반 테스팅 기법은, 프로그램의 특성을 표현한 모델을 근간으로 테스트 케이스들을 생성하는 기술이다. [8]에서는 Java PathFinder 모델검증 도구를 이용하여 테스트 케이스를 생성하고 있다. [9]는 비행 안내 시스템에 관한 사례 연구를 각각 기호적(symbolic), 한정적(bounded), 명시적(explicit) 모델검증 도구에 적용해 테스트 케이스를 생성하고, 각 모델검증 도구가 테스트 케이스를 생성하는 시간 및 자원 정도를 비교하고 있다. [10]에서는 추상 상태 기계(abstract state machine) 기반의 명세를 Promela 모델로 자동 변환하여, Spin 모델검증 도구가 생성하는 검증반례를 테스트 케이스로 이용한다. 위와 같은 기법의 경우, 모델검증 도구가 생성하는 검증반례를 이용해 테스트 케이스를 생성하지만, 본 연구에서 제안하는 MOKERT프레임워크와는 달리, 검증반례를 재연하는 기능은 없다.

본 연구와 밀접한 관련이 있는 소프트웨어 검증 도구 중 CHESS[11]는 무상태 검색(stateless search) 기법을 사용하여 동시성 프로그램의 많은 스케쥴링 시나리오를 체계적으로 테스트하며, 검출된 오류시나리오를 재연하는 기능을 제공한다. 하지만, 이 기법은 검증 대상 및 환경이 완전히 구현된 상태에서만 적용 가능하며, 기반 플랫폼(WIN32API 등)에 크게 의존하기 때문에, 하드웨어 바로 위에서 동작하는 커널을 직접적으로 테스트할 수는 없다. 다른 소프트웨어 모델검증 도구들 [12,13]은 술어 추상 기법이나 SAT 인코딩을 통해 검증 대상 C 프로그램으로부터 정형 모델을 자동으로 추출하여 모델검증을 수행하나, 상태폭발문제 등의 문제점들로 말미암아 그 적용범위가 크게 제한되어 있다. MOKERT는 검증 대상 프로그램의 완전 자동화된 분석을 목적으로 하고 있지는 않으며, 그 대신 전문가의 지식을 활용하여 어플리케이션에 대한 추상화 모델과 환경을 생성함으로써 커널에 존재하는 동시성 오류의 실제적인 검출을 목표로 한다([14] 참조).

3. 모델검증 도구 SPIN과 모델추출 도구 Modex

3.1 모델검증 도구 SPIN

모델검증 도구 SPIN[2]은 모델 명세 언어로 Promela를 사용한다. C 언어와 Promela 언어의 공통적인 특성들로 말미암아, C 프로그램으로 작성된 검증 대상 프로그램은 이에 대응되는 Promela 모델로 변환할 수 있다. 예를 들어, C언어에서의 스레드는 Promela 모델에서 프로세스로 표현할 수 있으며, C에서 사용되는 모든 제어 구문들(if, goto, while 등)은 Promela의 제어 구문으로 자동으로 변환할 수 있다. 또한, Promela는 typedef와 배열 등을 사용하여 복잡한 자료 구조를 표현할 수 있다.[1] 또한, Promela는 C 언어의 함수 호출에 대응할 수 있는 inline 함수를 제공한다. 이에 더하여, SPIN 4.0 이상의 버전에서는 Promela언어에 C 구문을 내장하여 사용할 수 있다. 이 기능을 통하여, C언어로 작성된 프로그램에서 Promela 모델로의 자동변환의 가용성이 증가하였다.

1) 하지만, 포인터 변수의 직접적인 표현은 불가능하며, 배열 등을 통하여 간접적으로 표현해야 한다.

모델검증 도구 SPIN이 생성하는 검증반례에는 다음과 같은 정보들이 포함되어 있다.

- 단계 번호 (step number)
- 현재 프로세스의 고유 번호(process ID)
- 현재 프로세스가 수행하는 proctype 이름
- 현재 수행중인 Promela 모델의 행 번호 *l*
- Promela 모델의 *l* 번째 행 구문

예를 들어, 그림 1에 검증반례의 45번째 단계는 프로세스 1번이 `proc_readdir`이라는 proctype의 125번째 행을 실행한 것이고, 그 구문은 `proc_subdir==1`이라는 것을 표현하고 있다. 그리고 45번째 단계에서 46번째 단계로 넘어가면서 프로세스 1번에서 프로세스 2번으로 스케줄이 바뀌고, 프로세스 2번이 `remove_proc_entry`라는 proctype의 156번째 행을 실행함을 보이고 있다.

3.2 모델추출 도구 Modex

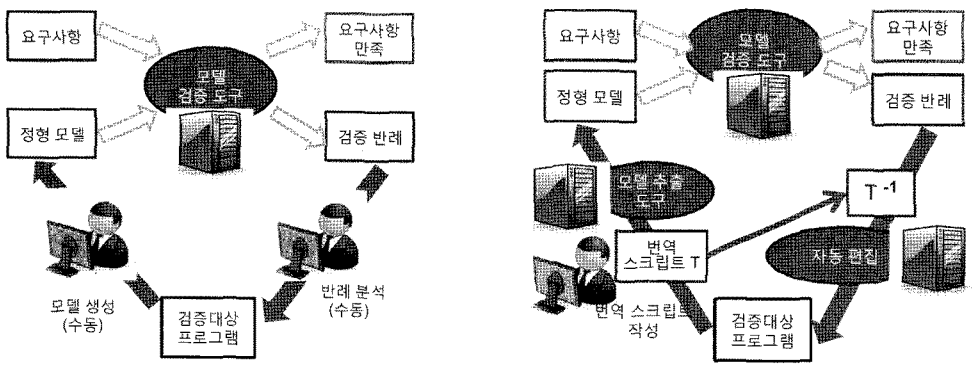
Modex[1]는 C 프로그램과 변환 스크립트를 입력으로 받아, C 프로그램을 Promela 언어로 자동 변환하는 모델 추출 도구이다. Modex는 C 프로그램에서 `if`, `while`, `goto`와 같은 제어 구문을 그에 상응하는 Promela 제어 구조로 자동 변환한다. 그 결과 Modex가 생성한 Promela 모델은 검증 대상 C 프로그램과 동일한 제어 구조를 가지게 된다. 제어 구조를 제외한 다른 C 구문들은 Promela 모델의 내장형C 구문으로 표현된다. C 프로그램에서 수식들은 `c_expr{…}` 키워드로 시작하는 내장형 C 수식으로 표현되며, 변수 대입이나 함수 호출 등은 `c_code{…}` 키워드로 시작하는 내장형 C 구문으로 표현된다. Promela명세의 내장형 구문들은 SPIN 모델검증 도구가 생성하는 검증 프로그램의 코드에 그대로 복사되어 사용되게 된다.

사용자는 변환 스크립트를 사용하여 변환과정을 조절할 수 있다. Modex에서 사용하는 변환 스크립트는 변환표(translation table)로 표현된다. 하나의 변환표에는 변환 대상 C 프로그램의 각 함수에 대한 변환 규칙(translation rule)이 기술되어 있다. 하나의 변환 규칙은 특정 문자 패턴으로 시작하는 검증 대상 C프로그램의 구문이, 어떻게 Promela 구문으로 변환되어야 함을 명세하고 있다.

4. 모델 기반 커널 테스팅 프레임워크

4.1 개 요

모델 기반 커널 테스팅(MOdel-based KERnel Testing(MOKERT)) 프레임워크는 모델검증 기법의 결과인 검증반례를 실제 검증 대상 프로그램에 적용함으로써, 커널 코드에서의 동시성 오류 테스팅의 어려움을 극복하는 것을 목표로 한다. 모델검증 단계에서의 반례를 실제 검증 대상 프로그램에서 재연하는 것은 소프트웨어 산업 현장에 모델검증 기술을 적극적으로 도입하는 데 중요한 요소임에도, 대부분의 모델검증 프레임워크는 이 문제에 대해 적합하게 논의하고 있지 않다. 전통적인 모델 기반 테스팅 방식은 그림 2(a)에 표현된 것과 같이, 정형 모델을 사용자가 수작업으로 생성하고, 반례의 분석도 사용자가 수작업으로 수행한다. 이러한 과정은 많은 시간과 노력이 소모된다. MOKERT 프레임워크에서의 모델 기반 테스팅 방식은 그림 2(b)에 표현되어 있다. MOKERT 프레임워크에서는 Modex를 사용하여 검증 대상 C 프로그램으로부터 (반)자동으로 정형 모델을 추출한다. 이 추출 과정에서는 변환 스크립트 T가 사용된다. 변환 스크립트 T에는 변환 대상 C 프로그램의 특정 구문이 Promela의 어떠한 구문으로 변환되어야 하는지 명세되어 있다. 이러한 T의 역인 T^{-1}은 변환된 Promela모델의 특정 구문이 변환대상 C 프로그램의 어



(a) 사용자가 제공하는 모델을 이용하는 전통적인 모델 기반 테스팅 방법

(b) 모델 추출 도구를 이용하는 (반)자동 모델 기반 테스팅 방법

그림 2 모델 기반 테스팅

떠한 구문을 변환한 것인지에 대한 정보를 나타내며, 따라서 T^{-1}을 이용하면 주어진 반례가 C프로그램에서 어떠한 구문의 실행을 표현하는 것인지 이해할 수 있다. 그리고 이 정보를 바탕으로, 검증 대상 프로그램이 반례에서 보이는 것과 같은 형태의 실행을 하도록 대상 프로그램을 수정(instrumentation)한다. 이렇게 수정된 프로그램을 실행함으로써, 실제 커널 프로그램에서 반례의 재연(replay)이 가능하다. MOKERT는 반례의 재연에 초점을 맞춘 만큼, liveness property를 제외한 일반적인 safety property를 요구명세 대상으로 한다. MOKERT 프레임워크가 제공하는 반례 재연 기능은 다음과 같은 장점이 있다.

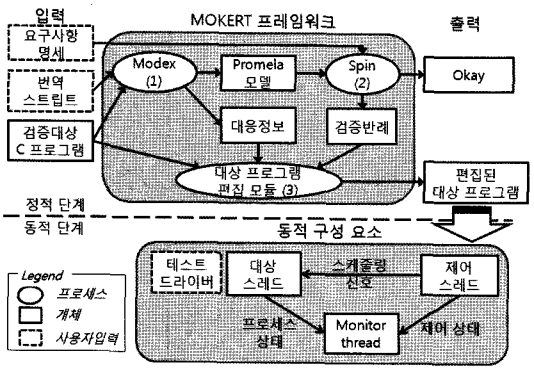


그림 3 MOKERT프레임워크의 구조

- 결함의 원인 분석 용이

MOKERT는 모델검증 단계에서 발견한 반례를 단계별로 실행할 수 있는 기능을 제공한다. 그러므로 사용자는 커널 코드상의 오류를 발생시키는 결함의 원인을 보다 편리하게 분석할 수 있다. 이는 동시성 오류를 찾는데 장점이 될 수 있는데, 그 이유는 결함의 재연을 위해 사용자가 직접 스레드 스케줄링을 조작해야 하는 불편함을 해결할 수 있기 때문이다.

- 모델 개선(model refinement)

모델검증 과정 중, 초기에 생성된 모델은 과도하게 추상화되어 있을 가능성이 크다. 올바른 검증 결과를 위해서는 모델을 점차 세밀하게 만드는 과정을 반복해서 적용할 필요가 있다. MOKERT에서는 정형 모델에서 반례가 발견되는 경우, 그 반례가 올바른 것인지 아니면 과도한 추상화로 생긴 오경보(false alarm) 인지의 여부를 자동으로 확인할 수 있는 기능을 제공한다. 반례가 오경보인 경우, 정형 모델을 개선하여야 한다. 이러한 개선 과정은 술어 추상화(predicate abstraction) 기술에서 사용하는 반례 기반 추상화 개선 기술(counter example guided abstraction refinement: CEGAR)[15]과 개념적으로 유사하나, 추상 모델의 개선을 수동으로 수행한다는 점에서 차이가 있다.

- 결함 패치 검증(validation of bug patch)

커널에서의 오류는 대부분 프로그램 코드를 사람이 눈으로 검사하거나(code inspection)나 토의, 혹은 가상 시나리오를 바탕으로 보고되며, 커널 영역에 대한 테스팅의 어려움으로 인해, 테스팅으로 실제적인 오류를 발견하는 경우는 비교적 적다. 따라서 리눅스의 ChangeLog는 실제로는 불필요하거나 올바르게 결함을 해결하지 못한 패치를 포함하고 있으며, MOKERT 프레임워크를 사용함으로써 결함 패치가 올바른지에 대한 여부를 검증할 수 있다(5.2절 참조).

4.2 MOKERT 프레임워크의 구조

MOKERT의 전반적인 구조는 그림 3에서 표현된 형태와 같다. MOKERT프레임워크는 정적 단계와 동적 단계로 구성된다. MOKERT프레임워크의 정적 단계는 다음과 같은 세부 과정들로 구성된다.

(1) 검증 대상 C 프로그램을 변환 스크립트를 기반으로 Modex를 통하여 Promela 모델로 변환한다. 생성된 Promela 모델은 프로세스, 환경 모델, 검증 요구사항 명세 등으로 구성된다. 이 과정에서, Modex의 중간 결과물로 Promela 모델의 행 번호와 검증 대상 C 프로그램의 행 번호 사이의 대응 정보가 생성된다.

(2) 모델검증 도구 SPIN은 이전 단계에서 생성된 Promela 모델이 검증 요구사항 명세를 만족하는지를 검사한다. 만약 모델이 검증 요구사항 명세를 만족한다면 모델검증 도구 SPIN은 검증 결과를 사용자에게 보고하고, 그렇지 않았으면 반례를 생성한다.

(3) 검증 대상 C 프로그램은 (1) 단계에서 생성된 Promela 모델의 행 번호와 검증 대상 C 프로그램의 행 번호 사이의 대응 정보와 (2) 단계에서 생성된 반례를 기반으로 하여 검증 대상 프로그램이 반례를 재연할 수 있도록 대상 프로그램을 수정(instrumentation)한다. 수정된 검증대상 C 프로그램은, 반례의 문맥전환이 일어나는 부분에 대응하는 검증 대상 C 프로그램의 부분에 probe가 삽입된 형태이다. 또한, 수정된 프로그램에는 MOKERT의 동적 단계를 지원하기 위해 제어 스레드(controller thread)와 모니터 스레드(monitor thread)가 포함되며, 이 밖에 모델검증에서의 환경 모델과 동일한 테스팅 환경을 구성하기 위한 테스트 드라이버가 포함된다. 수정된 대상 C 프로그램은 컴파일러를 이용하여 실행 가능한 형태의 프로그램으로 만들어진다.

동적 단계에서는 검증 대상 프로그램으로부터 생성된 스레드들이 제어 스레드, 모니터 스레드와 함께 실행된다. 제어 스레드는 반례의 재연을 위해 검증 대상 스레

드들의 probe와 통신을 하며 그들의 스케줄을 조정한다. 모니터 스레드는 검증 대상 스레드들과 제어 스레드의 상태를 관찰하고 반례가 올바르게 재연되고 있는지 확인하는 기능을 제공한다.

MOKERT 프레임워크에서는, 검증반례가 생성되면 사용자가 직접 테스트 드라이버를 구축해야 한다. 하지만, 이 작업은 일반적인 테스트를 위한 테스트 드라이버를 구축하는 작업보다 간단하며, 그 이유는 다음과 같다. 첫째, 사용자는 테스트 드라이버를 구축할 때에 모델검증 단계에서 사용했던 추상 환경 모델을 참고할 수 있다[16]. 둘째, MOKERT 프레임워크에 필요한 테스트 드라이버는, 하나의 검증반례가 나타내는 특정 시나리오만을 지원하면 되므로, 범용적인 테스트베드를 구축하는 데에 필요한 노력에 비해 적은 양의 노력만이 요구된다.

4.3 동적(run-time) 단계

동적 단계에서 모니터 스레드는 검증 대상 스레드들과 제어 스레드의 상태를 관찰함으로써 반례의 재연이 올바르게 이루어지고 있는지를 관찰한다. 모니터 스레드는 검증 대상 프로그램의 실행이 반례의 마지막에 도달하게 되면, 사용자에게 재연이 성공적으로 이루어졌다고 알린다. 이와 동시에 테스트 드라이버는 테스트 결과를 통해, 재연에 의한 실행이 요구사항을 만족하는지를 알려주게 된다. 다음은 반례 재연의 가능한 결과를 나타낸다.

(i) 반례의 마지막에 도달하면서 요구사항을 만족하지 않는 경우

(ii) 반례의 마지막에 도달하였지만 요구사항을 만족하는 경우

(iii) 반례의 마지막에 절대 도달하지 않는 경우

첫번째 경우는 사용자가 반례의 재연을 통해 기대하는 결과이다. 두번째와 세번째는 반례의 재연에 실패한 경우를 가리킨다. 만약 일정 시간 내에 반례의 마지막에 도달하지 못하는 경우, 모니터 스레드는 사용자에게 현재 검증 대상 스레드들의 상태와 함수 호출 스택의 상태 등의 로그 정보와 함께 반례 재연에 실패함을 알린다. 반례 재연의 실패는 다음과 같은 이유에 기인한다.

(i) 정확하지 않은 모델링

검증 대상 프로그램에 대한 모델이 과도하게 추상화된 경우, 실제 프로그램에서 반례를 재연할 수 없다. 즉, 모델에서 반영하지 못한 커널 전체의 환경 설정으로 반례의 실행이 실패할 수 있다. 예를 들어, 커널 전체적으로 인터럽트가 제한되어 있거나, 우선순위 스케줄링이 진행중인 경우, 제어 스레드가 검증 대상 스레드의 스케줄을 조절하는 것은 불가능하므로 반례를 재연할 수 없다.

(ii) 정확하지 않은 테스트 드라이버 구축

테스트 드라이버가 잘못 작성되어 검증 환경을 올바르게 구성하지 못하는 경우, 반례의 재연에 실패하게 된다.

proc_readdir()	remove_proc_entry()
68: do { /* missing atomic_inc(&de->count) */ 69: spin_unlock(&proc_subdir_lock);	
	10: spin_lock(&proc_subdir_lock); ... 23: if (!atomic_read(&de->count)) 24: free_proc_entry(de); ... 31: spin_unlock(&proc_subdir_lock);
70: if (filldir(dirent, de->name, de->namelen, 71: filp->f_pos,de->low_ino,de->mode>>12)<0)	

그림 4 proc_readdir() 함수와 remove_proc_entry() 함수간의 자원경쟁

5. 사례연구: Proc 파일 시스템의 proc_readdir()와 remove_proc_entry() 간의 자원경쟁 오류

본 절에서 기술하는 사례연구는, 실제 커널의 동시성 오류검출에 MOKERT 프레임워크가 어떻게 사용되는가를 구체적으로 기술함으로써, 본 프레임워크의 유효성을 실험적으로 보이고자 한다.

5.1 오류 설명

Proc 파일 시스템은 UNIX 계열의 운영체제에서 사용되는 가상 파일 시스템 중 하나이다. Proc 파일 시스템을 통하여 사용자는 특정 프로세스의 정보를 열람하거나 커널 모듈과의 통신을 수행할 수 있다. 리눅스 Changelog 2.6.22에는 리눅스 커널 2.6.21에서 발견된 proc_readdir() 함수와 remove_proc_entry() 함수 사이의 자원경쟁 오류를 보고하고 있으며, 이를 해결하기 위한 결함 패치를 소개하고 있다.

이 결함에 의하여 발생 가능한 자원경쟁 상태는 그림 4에 설명되어 있다. remove_proc_entry() 함수의 24번째 행에서 de(proc파일 시스템에서 하나의 디렉토리 항목을 나타내는 개체)를 제거하기 전, 23번째 행에서 de의 count값을 읽음으로써 현재 항목에 대한 동시 사용자가 없는지를 확인한다. 하지만, proc_readdir() 함수에서는, de 값을 읽기 전에 de의 count 값을 증가시키지 않고 있다. 따라서 remove_proc_entry() 함수에서는 proc_readdir() 함수가 de를 읽으려고 하는지 파악하지 못한 채 de를 제거하게 되고, 결과적으로 proc_readdir() 함수의 70번째 행에서는 remove_proc_entry() 함수의 수행으로 이미 제거 작업이 수행된 de 개체의 내용을 읽는 명령을 수행하게 된다.

5.2 오류 재연

MOKERT 프레임워크를 통해 proc_readdir() (총 67행), remove_proc_entry() (총 35행) 그리고 동기화 관련 함수 등 위 두 함수와 관련된 함수들이 Modex를 통하여 Promela 모델로 변환되었다. proc_readdir() 함수에 관한 Promela 프로세스는 총 68행이고, remove_proc_entry() 함수에 관한 Promela

프로세스는 총 36행이었다. 이에 더하여, 검증 환경(총 76행)은 "Jan", "Feb", "Mar", "Apr"로 명명된 4개의 항목을 가지는 하나의 proc 파일 시스템 디렉토리로 설정하였다. 이러한 환경 설정은 그림 5와 같다. 검증 환경은 `proc_readdir()` 함수와 `remove_proc_entry()` 함수에 관한 두 프로세스를 생성하며, `remove_proc_entry()` 함수가 "Feb" 항목을 지우는 것과 동시에 `proc_readdir()` 함수가 같은 디렉토리에 존재하는 모든 항목을 읽는 작업을 수행하도록 설정하였다.

요구 사항으로는 `proc_readdir()` 함수의 70번째 행에서 접근하는 디렉토리 항목은 이전에 제거되지 않은 것이어야 한다는 조건으로 표현하였다. 이 요구 사항을 검증하기 위해 각 디렉토리 항목에 `removed`라는 표식을 추가하였으며, 이는 디렉토리 항목이 제거되었을 때 true의 값을 가지게 된다. 모델검증 도구 SPIN은 이 모델로부터 59단계로 이루어진 반례를 생성하였다.

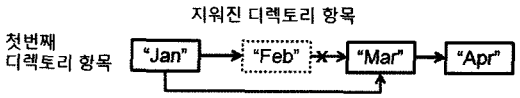


그림 5 디렉토리 항목 구조의 예

실제 운영 체제 커널을 대상으로 반례를 재연하기 위하여, 테스트 드라이버(총 98행)는 모델에서의 환경과 동일한 형태의 환경을 생성하도록 작성되었다. 커널에서 반례를 재연한 결과, `remove_proc_entry()` 함수로 제거한 디렉토리 항목은 `proc_readdir()` 함수의 결과로 출력되는 반면, 제거하지 않은 항목들이 출력되지 않는 증상을 확인할 수 있었다. 다시 말하여, `proc_readdir()` 함수는 그 실행 결과로 "Jan", "Feb" 만을 출력하였고, "Mar", "Apr"은 출력하지 않았다. 이는 그림 4에서 설명하는 `proc_readdir()` 함수와 `remove_proc_entry()` 함수 간의 자원경쟁 상태로 말미암은 잘못된 출력 결과이다.

이러한 결함은 리눅스 2.6.22 에서 `proc_readdir()` 함수의 68번째 행에 `atomic_inc(&de->count)` 구문을 추가함으로써 해결되었다. Promela 모델에 리눅스 2.6.22에서 추가된 구문을 반영하였을 때, 더 이상 오류가 발생하지 않았으며, 따라서 리눅스 2.6.22의 패치가 올바름을 확인할 수 있었다. 본 사례연구는 2명의 대학원생이 3일에 걸쳐서 수행하였으며, 그 중 2일은 proc 파일시스템 C 코드를 이해하기 위해 소요되었다.

5.3 새로운 자원경쟁 오류 검출

리눅스 커널 2.6.21 proc 파일 시스템을 검증한 연구를, 본 논문을 작성할 시점에서 가장 최근 리눅스 커널인 2.6.28.2로 확장 적용한 결과, 새로운 자원경쟁 문제로 인하여 커널 패닉이 발생되는 새로운 오류를 발견하였으며, 관련 리눅스 커널코드를 관리하는 담당자에게 보고하였다. 새로이 발견된 오류는, 스레드 T1이 `remove_proc_entry()`를 통해 하나의 디렉토리 "d1"를 삭제하려 할 때, 동시에 다른 스레드 T2가 `remove_proc_entry()`를 사용해 하위 디렉토리인 "d1/d2"를 삭제하면서 "d1/d2"를 가르키는 포인터의 값을 null로 바꾼 경우, T1이 이러한 변화를 파악하지 못하고 null 포인터를 참조하기 때문에 커널 패닉을 발생시킨다.

본 사례연구는 리눅스의 버전이 높아지면서 `remove_proc_entry()`의 길이가 35행에서 70행으로 증가하면서, 추가적인 모델링 및 환경을 구성했어야 함에도 불구하고 대학원생 1명이 하루 만에 검증을 완료할 수 있었다. 이는 기존의 리눅스 2.6.21 proc 파일 시스템을 위해 구성한 모델 및 재연 테스트베드를 재활용할 수 있었기 때문이다. 본 사례연구를 통하여, MOKERT가 기존에 보고된 오류의 진위를 판별할 수 있을 뿐 아니라, 새로운 동시성 오류를 검출하는 용도에 효과적으로 사용될 수 있다는 것을 보일 수 있었다.

6. 결 론

본 연구는 운영체제 커널의 동시성 오류를 효과적으로 검출하기 위한 MOKERT 프레임워크를 제안하고 있다. MOKERT는 사용자가 추상화 모델링 작업을 통해서 운영체제 커널의 각 구성요소를 독립적으로 분석한 결과를, 검증반례 재연 기능을 통하여 실제 커널 코드에 적용하여 확증하는 것을 지원한다. 따라서, MOKERT 프레임워크는 동시성을 가지는 커널의 동작을 모델검증을 통해 면밀히 검사함으로써 커널 코드 분석의 신뢰성을 향상시키는 데에 쓰일 수 있다.

본 연구에서는 리눅스 커널의 proc 파일 시스템을 대상으로 하는 사례 연구를 통하여, 이 프레임워크의 실효성을 확인하였다. 사례 연구를 통해서, 모델검증을 기반으로 하는 테스팅 기술의 발전을 위하여 모델추출 기술의 추가적인 개발 및 발전 되어야 함을 알 수 있었다. Modex를 포함하는 현재의 모델추출 기술은 아직 사용자의 높은 숙련도를 요구하고 있기 때문이다. 저자들은 이번 연구에서 개발된 MOKERT 프레임워크를 이용하여, 더 많은 리눅스 파일시스템의 컴포넌트들을 분석하여 신뢰성을 향상시키는 연구를 진행하는 한편, 공개 real-time OS 등의 분야로 적용대상을 확장할 예정이다.

참 고 문 헌

[1] G. J. Holzmann and R. Joshi, "Model-driven software verification," *Spin Workshop*, April 2004.

[2] G. J. Holzmann, "The Spin Model Checker," Wiley, New York, 2003.
[3] K. Sen, "Effective random testing of concurrent programs," *Proceedings of the twenty-second IEEE/ACM International Conference on Automated software engineering*, 2007.
[4] R. H. Carver and Y. Lei, "A General Model for Reachability Testing of Concurrent Programs," *IEEE International Conference on Formal Engineering Methods*, 2004.
[5] M. Christiens, J. D. Choi, M. Ronsse and K. Bosschere, "Record/Replay in the Presence of Benign Data Races," In International Conference on Parallel and Distributed Processing Techniques and Applications, 2002.
[6] S. Artzi, S. H. Kim, and M. D. Ernst, "ReCrash: Making Software Failures Reproducible by Preserving Object States," *European Conference on Object-Oriented Programming*, 2008.
[7] S. Narayanasamy, G. Pokam, and B. Calder, "BugNet: Recording application-level execution for deterministic replay debugging," *IEEE Micro*, 26(1): 100-109, 2006.
[8] W. Visser, C. S. Pasareanu and S. Khurshid, "Test input generation with Java PathFinder," *International Symposium on Software Testing and Analysis*, 2004.
[9] M. P. E. Heimdahl, S. Rayadurgam, W. Visser, G. Devaraj and J. Gao, "Auto-generating Test Sequences Using Model Checker: A Case Study," *Formal Approaches to Software Testing*, 2004.
[10] A. Gargantini, E. Riccobene and S. Rinzivillo, "Using Spin to Generate Tests from ASM Specifications," *Abstract State Machines*, 2003.
[11] M. Musuvathi, S. Qadeer and T. Ball, "Chess: A systematic testing tool for concurrent software," Microsoft Research Technical Report MSR-TR-2007-149, 2007.
[12] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar, "The software model checker BLAST: Applications to software engineering," *International Journal on Software Tools for Technology Transfer*, 2007.
[13] E. Clarke, D. Kroening, and F. Lerda, "A tool for checking ANSI-C programs," Tools And Algorithms for Construction and Analysis of Systems, 2004.
[14] P. Camara, M. Gallardo and P. Merino, "Model extraction for ARINC 653 based avionics software," *Spin workshop*, 2007.
[15] E. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, "Counterexample-guided abstraction refinement," *Computer Aided Verification*, July 2000.
[16] M. Kim, Y. Kim, Y. Choi and H. Kim, "Pre-testing flash device driver through model checking techniques," *IEEE International Conference on Software Testing, Verification and Validation*, April 2008.

김 문 주

1995년 KAIST 전산학과 학사. 2001년 Univ. of Pennsylvania 박사. 2002년~2004년 SECUi.COM 차장. 2004년~2006년 POSTECH 연구원. 2006년~현재 KAIST 전산학과 조교수. 관심분야는 정형검증, 소프트웨어 테스팅, 내장형 소프트웨어

홍 신

2007년 KAIST 전산학전공(학사). 2007년~현재 KAIST 대학원 전산학전공 석사과정. 관심분야는 소프트웨어 검증, 모델 체킹, 프로그램 분석