Continuation 기법을 이용한 임베디드 리눅스 커널 메모리 관리 기법

(Kernel Memory Management of Embedded Linux using Continuation)

김 세 원, 홍 철 호, 김 영 필, 유 혁*
(Se-Won Kim, Cheol-Ho Hong, Young-Phil Kim, Chuck Yoo)

Abstract: This paper presents Linux kernel memory management methodology using Continuation. Continuation means the rest of the computation given a point in the computation. This is used for sharing kernel stack in micro kernel operating systems such as Mach or L4. Kernel stack is usually allocated in non-swapping area and each kernel thread has its own kernel stack. Therefore there are more running kernel thread in system, more kernel memory occupied by kernel stack is increased. In this paper, we propose the technique to efficiently manage kernel stack for embedded Linux using Continuation.

Keywords : Continuation, 리눅스, 커널 메모리 관리

I. 서 론

Continuation은 연산의 나머지(Rest of Computation)라는 의미로 특정 시점을 기준으로 그 이후의 연산을 의미한다. 본래 Continuation은 프로그래밍 언어에서 실행 흐름 제어를 정의하기 위한 수학적인 개념이었다[1]. Scheme과 같은 언어에서 사용되는 first-class Continuation이 대표적인 결과물이다.

운영체제의 실행 흐름을 보다 유연하게하고 성능을 향상시키기 위해 Continuation을 적용할 수 있다[1,2,4]. 과거 마이크로 커널 기반의 운영체제인 Mach에서 Continuation을 이용하여 시스템 서비스 간 IPC 성능을 개선시키기 위해 Continuation을 사용했다. 그 결과 각 스레드가 차지하는 메모리 공간의 크기를 최대 85%까지 감소했으며, IPC의 경우 14%, 예외처리는 60%까지 속도를 향상 시켰다.

리눅스의 스레드 모델은 2.6 버전부터 크게 향상되었다. Native Posix Threading Library(NPTL)라고 일컬어 지는 1:1 스레드 모델을 제공함으로써, 유저 스레드의 실행이 커널 스레드에 그대로 매핑된다[5]. 이로서 리눅스에서 스레드는 보다 정교한 실행 조건을 만족 할 수 있게 되었다. 특히, 임베디드 시스템의 경우, 하드웨어를 주기적으로 폴링(polling)하거나 특정 작업의 정확한 시간에 주기적인 실행을 요구하기도 한다. 리눅스의 새로운 스레드 모델은 이러한 임베디드 시스템의 요구 사항에 적합하기 때문에, 임베디드 시스템용 운영체제로 리눅스가 사용되어 왔다.

문제는 임베디드 시스템과 같이 하드웨어 자원의 제약을 갖는 시스템은 리눅스와 같은 범용 운영체제의 기술을 그대로 사용하기가 어렵다. 특히, 메모리의 경우 PC와 비교했을 때, 많게는 1/8정도밖에는 제공할 수 없다[3]. 이러한 문제는 과거 컴퓨터 기술 연구에서도 제기됐던 문제로, 기존 Continuation 관련 연구 동기가 되기도 했다.

본 연구는 임베디드 시스템에서 리눅스의 커널 스레드를 사용하는 환경에서 효율적인 메모리 활용을 위한 Continuation 기법에 대해 고찰 하고자 한다. 더 나아가 일반적인 임베디드용 운영체제로 많이 사용되는 리눅스에서의 커널 스레드 구현 방안에 대해 논의하고자 한다.

* 교신저자(Corresponding Author)
김세원, 김영필, 홍철호, 유혁: 고려대학교 컴퓨터학과

II. 본 론

1. Continuation

프로그래밍 언어에서 기원한 Continuation개념은 Mach 커널에서 실행 흐름을 조절하기 위한 기법으로 소개되었다[1]. 마이크로 커널 기반의 Mach는 시스템 서비스를 구성하는 스레드 간의 IPC가 시스템 성능을 저하하는 주된 요인으로 지적되어왔다. 이를 해결하기 위한 방안으로 Continuation이 제안되었고, 그 결과 메모리 사용량 및 IPC 지연시간을 줄일 수 있었다.

```
example(arg1, arg2) {
    use arg1;
    current_thread()->save_arg = arg2;
    wait(example_cont)
    //NOTREACHED
}
example_cont() {
    arg2 = current_thread()->save_arg;
    use arg2;
    return_from_systemcall(SUCCESS);
}
```

그림 1. Continuation을 이용한 시스템콜 구현 예시

그림 1은 example 이라는 시스템콜에서 Continuation을 이용하여 다시 작성된 코드를 보인다. 사용자 스레드에서 example 이라는 시스템 콜을 호출하고 이 시스템 콜은 arg1을 사용하고 wait()을 이용하여 잠시 실행을 멈추고 arg2를 사용한다. 이때 wait()을 호출하는 시점을 기준으로 그 이후를 example_cont()라는 함수로 추상화된 Continuation을 구현했다. example_cont()함수로 인해 커널 스레드의 실행은 wait()이후의 실행부터는 이전의 실행 컨택스트를 기억할 필요가 없다. 즉, example()을 실행 하는 동안 커널 스택에 저장된 상태 정보들은 wait()을 호출하기 전에 커널 스택이 아닌 공간에 저장했기 때문이다.

반면 커널은 이처럼 한 스레드에서 wait()을 통해 자신의 Continuation을 전달함을 인식하게 되면, 이전 스레드의 커널 스택의 내용을 무시할 수 있게 된다. 그리고 wait()을 통해 문맥 교환이 발생하고 다음 실행할 커널 스레드는 이전 스레드의 커널 스택을 그대로 물려받을 수 있다. 이것이 Continuation 기법을 이용하여 커널 스레드 간 커널 스택을 공유하는 기법방법이다.

2. 리눅스에서 고려해야 할 점

Continuation을 이용한 커널 스택을 공유하기 위해서는 다음 3가지의 경우를 고려해야 한다.

1) 스레드가 Continuation 함수를 등록하고 블록될 때

2) 블록된 스레드가 Continuation으로부터 복귀 할 때

3) Continuation을 실행하고 다시 사용자 모드로 복귀할 때

위 3가지 경우는 모두 Continuation과 관련해서 커널 스택의 변화가 발생하는 곳이다. 1)의 경우 스레드 자신이 블록되면서, 현재 커널 스택의 내용을 정리해야한다. 2)에서는 스레드가 wait으로부터 복귀하여 미뤄두었던 Continuation을 실행하는데, 실행에 필요한 정보를 커널 스택에 할당해야 한다. 마지막으로 3)에서는 커널 모드로의 실행을 마치고 복귀를 한다. 복귀 시점의 커널 스택은 Continuation을 실행 한 실행 문맥이 저장되어 있다. 하지만, 이 정보는 더 이상 필요가 없기 때문에 비워주면 된다.

앞서 설명한 커널 스택을 사용의 변화가 발생하는 3가지 지점 말고, 스레드 초기화 과정의 커널 스택관련 연산에서 고려해야 할 사항이 있다. 일반적인 커널 스레드 구현은 각 스레드 마다 개별적인 커널 스택을 소유하게 된다. 리눅스 역시 마찬가지며, 이를 지원하는 커널 루틴은 kernel_thread()라는 함수이다. 커널 스레드간 스택을 공유한다는 의미는, 모든 스레드가 커널 스택을 가지고 있을 필요가 없다는 뜻이다. 따라서 현재 실행중인 스레드가 Continuation을 통해 커널 스택을 다른 스레드에게 양도 할 수 있다면, 문맥 교환으로 인해 이어서 실행하는 스레드는 양도된 커널 스택을 사용 할 수 있어야 한다. 그러므로 스레드의 생성과 커널 스택의 초기화 과정은 분리될 필요가 있으며, 스레드의 스택에 대한 소유권은 보다 유연해질 필요가 있다.

커널 스택의 구현에 있어서 다음과 같은 조건을 만족 시킬 수 있어야 한다.

- 커널 스택은 커널 모드 실행 시에 스레드에 동적으로 할당되고, 커널 모드 실행이 종료되어 사용자 모드로 복귀 될 때에는 커널 스택을 반환 할 수 있어야 한다.
- 커널은 스레드의 커널 스택의 집합인 pool을 만들고 스레드의 커널 스택에 대한 요청과 반납을 처리 할 수 있어야 한다.
- 일반적으로 커널 스택은 사용자 모드의 컨택스트 정보를 같이 담고 있는데, 커널 스택을 공유하기 위해서는 이러한 정보를 분리할 필요가 있다.

Ⅲ. 결론 및 향후과제

Continuation기법과 그것을 리눅스에서 활용하기 위해 고려해야 할 방법들을 살펴보았다. Continuation을 사용하기 위해서는 커널의 많은 부분의 수정과 재설계를 요구한다. 특히 스레드와 같이 커널의 핵심 구성요소를 수정하는 것은 그것을 사용하는 다른 부분 까지 영향을 미칠 정도로 방대한 작업이 될 수 있다.

하지만, Continuation을 통해서 얻을 수 있는 커널 메모리의 효율적인 활용은 하드웨어 자원이 일반 PC나 서버보다 열악한 임베디드 시스템에서 유용한 기법이 될 것이다.

본 논문은 리눅스에서 Continuation을 구현하기 위해 과거 연구들의 선행 조사로부터 고려사항을 도출 한 것이며, 이를 통해 리눅스 커널을 수정하고 구현해 나갈 계획이다.

참 고 문 헌

[1] Richard P. Draves, Brian N. Bershad, Richard F. Rashid, and Randall W. Dean, "Using Continuations to Implement Thread Management and Communication in Operating Systems", Proceedings of the ACM SIGOPS Operating Systems Review, volume 25, Issue 5, pp.122-136, 1991.

[2] Atul Adya, Jon Howell, Marvin Theimer, William J. Bolosky, and John R. Douceur, "Cooperative Task Management without Manual Stack Management", Proceedings of the USENIX Annual Technical Conference, June 2002.

[3] "한백전자 임베디드 플랫폼 하드웨어 사양" http://www.hanback.co.kr/htm/sub2_9b.htm

[4] Mattew Warton, "Single kernel stack L4", BE Thesis, University of NSW, Sydney 2052, Australia, 2005

[5] "Linux Kernel 2.6:the Future of Embedded Computing, Part1" http://www.linuxjournal.com/article/7477

저 자 소 개

김 세 원(Se-Won Kim)

2004년 2월 : 고려대학교 컴퓨터학과 학사
2006년 2월 : 고려대학교 컴퓨터학과 석사
2006년 3월~현재 : 고려대학교 컴퓨터학과 박사과정

관심분야 : kernel architecture, system architecture
Email : swkim@os.korea.ac.kr

김 영 필(Youngpil Kim)
2002년 2월 : 고려대학교 컴퓨터학과 학사
2004년 2월 : 고려대학교 컴퓨터학과 석사
2004년~현재 : 고려대학교 컴퓨터학과 박사과정

관심분야 : kernel architecture, self-managing kernel
Email : ypkim@os.korea.ac.kr

홍 철 호(Cheol-Ho Hong)
2001년 2월 : 고려대학교 컴퓨터학과 학사
2003년 2월 : 고려대학교 컴퓨터학과 석사
2007년~현재 : 고려대학교 컴퓨터학과 박사과정

관심분야 : 가상 머신 플랫폼, 멀티 코어 운영체제
Email : chhong@os.korea.ac.kr

유 혁(Chuck Yoo)
1982년 2월 : 서울대학교 전자공학과 학사
1983년 2월 : 서울대학교 전자공학과 석사
1986년 8월 : Master of Computer Science in University of Michigan
1990년 8월 : Ph.D of Computer Science in University of Michigan
1990년~1995년 : Sun Microsystems Lab. Researcher
1995년~현재 : 고려대학교 컴퓨터학과 교수
관심분야 : 시스템 가상화, 멀티 코어 플랫폼, 커널 네트워킹, 센서 네트워킹, 멀티미디어 스트리밍
Email : hxy@os.korea.ac.kr