실시간 운영체제에서 효율적 메모리사용을 위한 심플 동적 로더 설계 및 구현

이정원[0], 김용희, 이숭열, 이철훈
충남대학교 컴퓨터 공학과
{booster[0], yonghee, sy-lee, clee}@cnu.ac.kr

A Design and Implementation of Simple Dynamic Loader for Efficient Memory Usage in Real-time Operating System

Jungwon Lee[0] Yonghee Kim, Soong-Yeol Lee, Cheolhoon Lee
Dept. of Computer Engineering, Chungnam National Univ.

요 약

일반적인 임베디드 환경에서는 정적으로 각 기능 요소들을 실행 이전에 미리 컴파일하여 사용하였다. 이 경우 자주 사용하지 않는 모듈도 모두 메모리에 로드되어 많은 용량의 메모리를 요구하게 된다. 따라서 한정된 메모리를 효율적으로 사용하여 시스템의 자원을 낭비하지 않기 위해 동적으로 실행 시 필요한 모듈을 적재하여 사용하고 더 이상 사용하지 않는 라이브러리는 메모리로부터 제거하는 동적 라이브러리 로딩은 좋은 대안이 되고 있다. 본 논문에서는 실시간 운영체제 rRTOS 에 동적 로딩 기능을 적용하기 위해 Simple Dynamic Loader(심플 동적 로더)를 설계 및 구현하였다.

1. 서론

정보가전기술의 발전으로 예전에 비해 대용량의 하드웨어로 컴퓨터가 구성되고 있다. 그러나, 휴대용 장비와 같은 소형의 장비에는 한정된 메모리와 디스크를 사용하기 때문에 효율적으로 기억 장치를 관리하여 불필요한 용량 낭비를 줄이는 기능이 운영체제에 필요하게 되었다. 이러한 기술은 디스크를 메모리로 사용하는 가상메모리 기법과 실행 시에 모듈을 동적으로 메모리에 적재하여 메모리의 낭비를 줄이는 동적 재구성 기법 등이 있다.

본 논문에서는 이러한 동적 라이브러리 로딩기법으로 메모리 사용의 효율성을 높이기 위하여 실시간 운영체제 rRTOS™에 Simple Dynamic Loader 를 설계 및 구현하였다[1][2].

본 논문의 구성은 2 장에서 실시간 운영체제 rRTOS™와 Dynamic Library Loading 을 소개하고, 3 장에서 Simple Dynamic Loader 설계 및 구현을, 4 장에서 테스트 환경 및 결과에 대하여 기술한다. 마지막으로, 5 장에서는 결론 및 향후 연구 계획에 대해 기술한다.

2. 관련 연구

2.1 실시간 운영체제 rRTOS™

rRTOS™는 충남대학교 시스템 소프트웨어 연구실에서 개발한 선점형 우선순위 기반의 실시간 운영체제(Real-time Operating System)이다. 실시간 운영체제 rRTOS™의 기본 기능은 다음과 같다.

2.1.1 우선순위 기반 멀티 태스킹

태스크는 중요도에 따라 우선순위가 256 개까지 부여되고 가장 높은 우선순위의 태스크가 CPU 를 선점하여 수행된다. 동일한 우선순위에 대한 스케줄링 정책으로 FIFO 와 라운드-로빈 스케줄링을 제공한다.

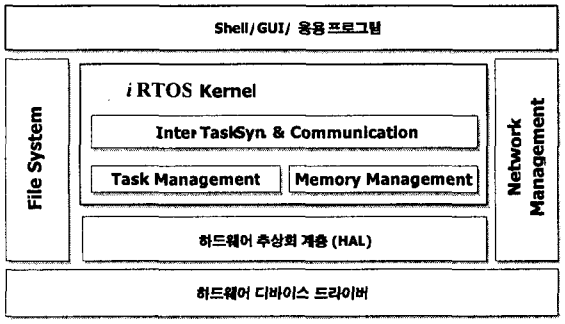
2.1.2 동적 메모리 관리

가변 크기의 메모리를 제공하기 위한 힙 스토리지 매니저와 고정 크기의 메모리 할당을 위한 메모리 풀을 지원하고 있다.

2.1.3 태스크간 통신

* 이 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임.

태스크간 동기화와 상호배제를 위한 세마포어, 특정 이벤트연산에 의한 통신을 위한 이벤트 플래그를 비롯하여 메시지 큐, 메시지 포트, 메시지 메일박스, 태스크 포트 등을 제공한다.



[그림 1] iRTOS™ 의 구조

2.2 Dynamic Library Loading

2.2.1 ELF 오브젝트 파일

오브젝트 파일의 표준이라 할 수 있는 ELF 표준 오브젝트 파일은 아래 3 가지의 대표적인 형식으로 구성되어 있다.

- Re-locatable file: 코드와 데이터영역을 가지고 있으며 다른 오브젝트 파일과 링킹과정(변수나 함수에 대한 심볼을 실제 변수의 주소로 매핑시켜주는 과정)을 통해 executable file 또는 shared object file로 되는 파일이다.

- Executable file: 실행할 수 있는 프로그램 파일이다.

- Shared-object file: re-locatable file 이나 다른 shared-object file 과 동적으로 링킹과정을 통해 실행 이미지를 생성할 수 있는 파일이다.

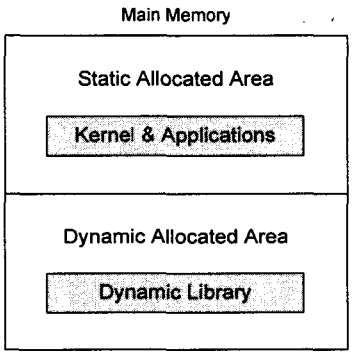
2.2.2 동적 라이브러리 로딩

프로그램의 시작 이후 어느 시점에 적재되는 동적 라이브러리는 shared-object file 을 사용한다. 프로그램 수행 중 해당 라이브러리를 필요로 하는 때에 라이브러리가 자신과 링크되어 있지 않다면 메모리에 로드하고 링크 작업을 수행한다. 즉, 컴파일시 동적으로 적재하여 수행할 라이브러리는 오브젝트 파일의 dynamic symbol table 에 각 심볼의 주소를 가상으로 가지고 있으며, 실제 메모리에 적재시 이 주소를 실제 주소로 바꾸어서 해당 라이브러리를 호출하게 된다.

이를 위해서 유닉스 계열에서는 dlopen(), dlsym(), dlclose() 같은 API 함수를 제공하여 동적으로 로드된 라이브러리의 심볼에 대한 접근을 가능하게 해준다 [3][4][5].

3. Simple Dynamic Loader 설계 및 구현

본 논문에서는 라이브러리를 메모리의 특정영역에 올려서 사용하는 경우를 가지고 로더를 구현하였다. 동적으로 로딩할 라이브러리들은 해당 라이브러리의 심볼을 얻는 모듈과 함께 컴파일 하여 디스크에 저장한다. 즉 shared-object file 이 아닌 executable file을 공유 라이브러리로 사용하는 것이다.



[그림 2] 로더 사용시 메모리 할당 영역

시스템이 구동되며 필수적인 요소들은 정적 할당 영역에 적재되어 실행된다. 시스템의 상황에 따라 동적으로 적재될 라이브러리의 경우는 동적 할당 영역에 해당부분을 적재하게 된다.

3.1 Static Allocation Area

라이브러리를 호출하는 영역에서는 호출하고자 하는 함수의 모조함수(Dummy Function)를 가지고 프로그램을 ELF excutatble file format 으로 컴파일 하여 수행한다. 이때, 모조함수를 호출 할 경우에는 그 함수에서 동적으로 적재할 라이브러리를 메모리에 로딩하고 동적 라이브러리의 심볼을 얻는 함수를 호출해 그 함수 포인터를 가지고 실제 함수를 호출하게 된다.

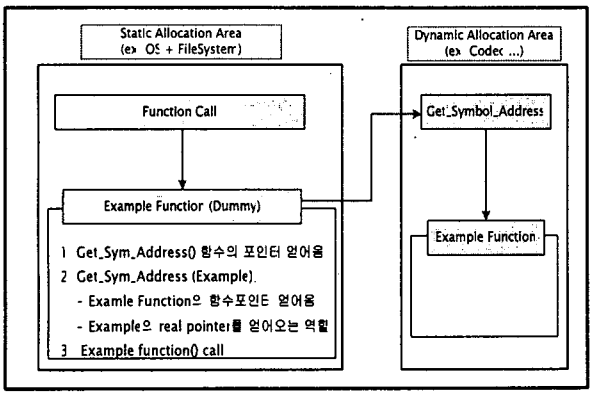
3.2 Dynamic Allocation Area

동적으로 적재되는 이미지의 엔트리 포인트는 심볼을 얻는 함수인 Get_Symbol_Address()함수로 한다. 이렇게 함으로써 라이브러리가 적재된 후, 첫번째 주소는 Get_Symbol_Address()함수의 시작주소가 된다.

이것으로 동적으로 로딩하여 수행하고자 하는 함수나 변수의 포인터를 얻어와 해당하는 심볼에 접근하게 된다.

이와 같이 동적으로 적재되는 라이브러리는 호출될

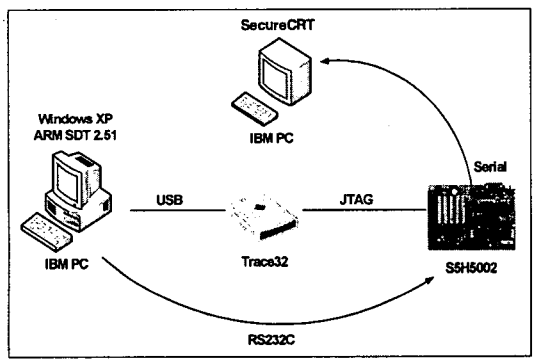
수 있는 API 함수들을 리스트로 관리하여 해당 함수의 함수 포인터 주소를 반환할 수 있는 Get_Symbol_Address()를 포함하여 컴파일 한다.



[그림 3] Simple Loader 의 심볼 주소 얻는 과정

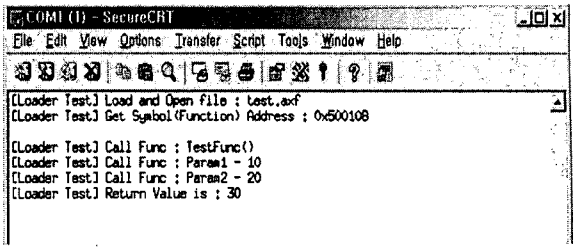
4. 테스트 환경 및 결과

본 논문은 ARM940T 기반의 S5H5002 32-Bit RISC MicroProcessor 에 *i*RTOS™를 탑재하여 테스트 하였다. 컴파일러는 ARM SDT 2.51 을 사용하였고 디버거는 Lauterbach 사의 Trace32 를 사용하였다. 동적 라이브러리로딩을 테스트하기 위해 Memory Stick 을 사용하였다.



[그림 4] 개발 환경

동적으로 메모리에 적재하는 테스트를 위해 두 개의 입력값을 받아 합을 구한 후 반환하는 함수를 라이브러리로 하여 메모리 스틱에 저장한 후, 파일 시스템으로 해당 라이브러리를 메모리에 적재하여 동적으로 구동시키는 테스트를 하였다.



[그림 5] Simple Loader 동작 테스트

5. 결론 및 향후 연구 과제

본 논문에서는 한정된 메모리를 효율적으로 사용하기 위해 동적 라이브러리 심플 로더를 설계 및 구현하였다. 향후 연구 과제로는 현재 ARM 플랫폼 기반의 로더를 다른 CPU 플랫폼에서 구현하는 것과 현재 고정된 메모리에 로딩하는 방식에서 범용 OS 에서 채택하고 있는 동적 로더와 같은 표준 ELF 파일 포맷인 shared-object file 형식을 준수하는 로더를 구현하는 것이다.

6. 참고 문헌

[1] http://www.aijisystem.com
[2] Stefan Beyer 외 2 명, *"Dynamic Configuration of Embedded Operating Systems"*,
[3] *"Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification"* TIS Committee, May 1995
[4] *"ARM ELF"*, ARM, June 2001
[5] David A. Wheeler, *"Program-Library-HOWTO"*, LDP(http://www.linuxdoc.org), 2000