실시간 운영체제에서 효율적인 메모리 사용을 위한 printf() 함수 설계 및 구현

이재규°, 성영락*, 이철훈
충남대학교 컴퓨터공학과, *국민대학교 전자정보통신공학부
{jglee°,chlee}@ce.cnu.ac.kr, *yeong@mail.kookmin.ac.kr

Design and Implementation of printf() for Efficient Memory Use in Real-Time Operating System

Jae-Gyu Lee°, Yeong Rak Seong*, and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National Univ.
*School of Electrical Engineering, Kookmin Univ.

요 약

실시간 운영 체제(Real-Time Operating System)는 시스템 동작이 논리적 정확성뿐만 아니라 시간적 정확성에도 좌우되는 운영 체제이다. 또한 실시간 운영체제는 멀티태스킹(Multitasking) 과 ITC(Inter Task Communication)을 제공한다는 점에서 일반 운영 체제인 Windows, Linux, Unix등과 같지만 시간적 정확성을 보장해야 한다는 점에서 일반 운영 체제와 다르다. 이러한 실시간 운영 체제를 포함하는 내장형 시스템(Embedded System)은 각각의 목적에 맞도록 모든 것이 최적화되어야 하므로 실행 이미지의 크기도 작아야 하고 사용 가능한 메모리에도 제한이 있다. 본 논문에서는 실시간 운영 체제에서 이러한 조건들을 고려하여 효율적인 메모리 사용을 위한 printf() 함수를 설계하고 구현한 내용에 대해서 설명한다.

1. 서 론

실시간 운영 체제는 내장형 시스템에 사용되는 대표적인 운영 체제이다. 내장형 시스템은 프로세서의 처리 속도, 메모리, 전원 등의 자원이 제한된 환경에서 미리 정해진 특정한 기능을 수행하도록 설계된 시스템이다. 다시 말해 특정한 기기에 주어진 작업을 수행하도록 구동시키는 시스템이라고 할 수 있다. 따라서 이러한 내장형 시스템을 설계하거나 개발할 때 고려 사항들로는 첫째, 최악의 상황에서도 정해진 시간내에 동작해야 하고 둘째, 작은 기기의 용량에 맞도록 설계되어야 하므로 가볍고 효율적이어야 하며 셋째, 위험성을 내포하고 있는 불안정한 상황에서도 오류없이 안정적으로 동작되도록 해야 하며 넷째, 저가의 비용으로 높은 효율을 발휘하도록 해야 한다는 점등이 있다. 실시간 운영 체제의 이러한 특징 때문에 설계자는 발생 가능한 여러 가지 시나리오를 모두 인지한 상태에서 시스템 개발을 해야 한다.

기존의 일반 운영 체제의 라이브러리 함수인 printf()를 실시간 운영 체제에서 stdio.h를 포함해서 그대로 사용할 경우 메모리 사용에 있어서의 비효율성과 커널의 실행 이미지가 커지게 되고 최악의 경우 stack overflow가 발생할 수도 있게 된다. 따라서 실시간 운영 체제에 알맞은 printf() 함수가 필요하게 되어 설계하고 구현한 내용을 설명한다. 본 논문에서는 2장에서 관련된 연구를, 3장에서 printf() 함수의 설계 및 구현을, 4장에서 실시간 운영체제 iRTOS에서의 테스트 및 결과를, 5장에서 결론 및 향후 연구 과제를 기술한다.

2. 관련 연구

2.1 실시간 운영 체제 개요

실시간 운영 체제는 시간 결정성과 작은 크기의 실행 이미지를 특징으로 한다. 멀티태스킹(Multitasking)을 지원하고 태스크에 우선순위를 할당하여 높은 우선순위의 태스크가 CPU를 선점하는 스케줄링 방식을 사용한다. 또한, ITC(Inter Task Communication) 환경은 태스크간 동기화 및 통신을 위하여 세마포어(Semaphore), 메시지 메일박스(Message Mailbox), 메시지 큐(Message Queue) 등을 지원한다.

2.2 실시간 운영 체제의 메모리 관리

① 정적 할당(Static allocation)

정적 할당 방법은 컴파일 시간에 프로그램에서 사용될 메모리를 전역으로 선언하는 방법이다. 이 방법의 장점은 링크 타임에 프로그램 실행을 위한 메모리를 결정할 수 있기 때문에 시간 결정성을 지킬 수 있고, 메모리와 관련된 버그들(leaks, failures, dangling pointers)이 존

재하지 않는다는 것이다. 단점은 프로그램에서 사용될 양보다 불필요하게 더 많은 메모리가 할당되는 점이다.

② 동적 할당(Dynamic allocation)

동적 할당 방법은 실행 시간에 메모리 힙(Heap)영역으로부터 사용자가 요청한 크기만큼의 메모리를 할당받는 방법이다.(ex. malloc(), free()) 이 방법의 장점은 사용자가 필요한 만큼의 메모리를 할당받기 때문에 낭비가 없다는 것과 사용이 편리하다는 점이다. 단점은 힙에서 사용 가능한 메모리블록을 찾기 위한 시간이 많이 걸려서 시간 결정성을 저해 할 수 있고, 외부단편화(External fragmentation)가 발생해서 No-Memory-Condition이 발생한다는 점이다. 이것을 해결하기 위한 방안으로 메모리 풀(Memory Pool)로부터 고정된 크기의 메모리 블록을 할당받는 방법을 사용한다.

2.3 iRTOS

iRTOS는 (주)아이네스테크에서 개발한 실시간 운영 체제로써 다음과 같은 특징이 있다.

① 멀티태스킹 지원
② 우선순위 기반의 선점형 스케줄링 사용
③ ITC를 위한 세마포어, 메시지 메일박스, 메시지 큐 지원
④ 동적 할당을 위한 힙과 메모리 풀 지원
⑤ Small Foot Print : 약 23KB

3. 설계 및 구현

3.1 Standard Library를 사용한 printf() 함수의 문제점

기존의 printf() 함수는 고정된 크기의 배열을 사용해서 버퍼로 사용하는 방법을 사용하게 된다 [그림 1].

```
printf(char *fmt,...){
        ...중략
        char string[256]; //버퍼
        vsprintf(string, fmt, ap); //실제 printf()
        ...생략
}
```

[그림 1] Standard Library 의 printf() 함수

이러한 방법을 사용하면 다음과 같은 문제점들이 발생할 수 있다.

① printf() 함수를 실행중인 태스크가 다른 태스크로 문맥 교환이 일어났을 때 계속적인 printf() 함수의 사용으로 태스크의 스택에 printf() 함수에서 사용하는 고정된 크기의 배열이 쌓임으로 인해서 stack overflow가 발생할 수도 있다.

② 고정된 크기의 배열을 사용하기 때문에 출력할 문자열이 적은 경우 메모리의 낭비가 생기게 되고, 출력할 문자열이 선언된 배열보다 크기가 많으면 오류가 발생하게 된다.

③ 실시간 운영체제인 iRTOS의 커널 실행 이미지는 약 23KB정도인데, 일반 C의 Standard Library인 stdio.h를 포함하게 되면 실행 이미지가 43KB를 넘게 된다.

3.2 MK_Printf() 함수의 설계 및 구현

iRTOS에서 기존 printf() 함수의 문제점을 해결하기 위해서 고정된 크기의 배열을 선언하지 않고 컴파일 타임에 출력 문자열의 크기만큼 메모리를 할당받아 사용하게 하였다 [그림 2].

```
MK_Printf(char *fmt, int args){
        int putc();
        doprint(fmt, &args, putc, CONSOLE);
}
```

[그림 2] MK_Printf() 함수

실제적으로 printf() 함수의 동작을 하는 doprint() 함수의 원형과 전달 인자들은 다음과 같다 [표 1].

void doprint(char *fmt, int *args, int (*func)(), int farg)	
char *fmt	MK_Printf() 함수를 위한 문자열
int *args	MK_Printf() 함수로의 전달 인자들
int (*func)()	한 문자를 put하는 함수
int farg	func() 함수로의 전달 인자

[표 1] doprint() 함수 원형

doprint() 함수는 fmt 문자열에서 '%' 또는 문자열의 마지막에 도달하기 전까지 한 문자씩 출력하도록 하였다 [그림 3].

```
for(;;){
        while( c=*fmt++ != '%') {
                if(c== '\0') return;
                (*func)(farg,c);
        }
        ...생략
}
```

[그림 3] doprint() 함수 1

다음으로 fmt 문자열이 '%-'인 경우 필드 내에서 왼쪽 정렬인지 오른쪽 정렬인지 판단하고, 필드의 빈 부분을 '0'으로 채우는 '%0'문자가 있으면 처리한다. fmt 문자열이 '%*' 이면 임의의 값을 인자 필드로부터 받아서 필드 폭과 정밀도를 처리해 준다. fmt 문자열에서 '%s'의 최

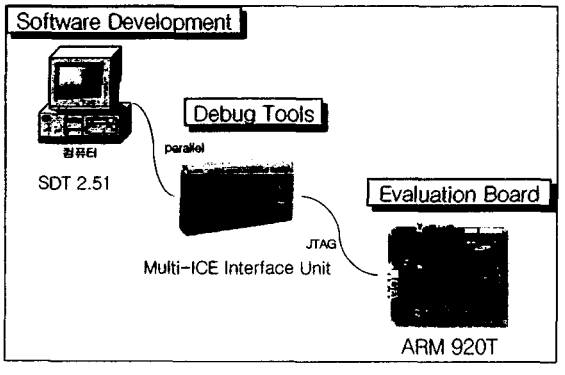
대 문자열 넓이를 지정해 주고 '!' 옵션에 대한 플래그를 처리를 한다다. 다음에는 '%c, %s, %d, %u, %o, %x'에 대해서 인자별로 문자, 문자열, 10진법, 8진법, 16진법의 처리를 한다. 이 부분은 기존 printf() 함수의 처리부분을 사용하였으나 printf() 함수의 가변 인자들을 출력 문자열로 만들어 주는 부분을 putc() 함수를 호출해서 한 문자씩 출력하도록 구현하였다 [그림 4].

```
if( (f = *fmt++) == '₩0' ) {
        ...중략
        return;
}
switch(f) {
        case 'c' :
                string[0] = (char) *args;
                string[1] = '₩0';
                fmax = 0;
                fill = ' ';
                break;
        case 's' :
                ...중략
        default :
                (*func)(farg f);
                break;
}
```

[그림 4] doprint() 함수 2

4. 실시간 운영체제 iRTOS에서의 테스트 및 결과

본 논문에서 구현한 MK_Printf() 함수를 iRTOS의 Kernel Library에 포함시켜서 SDT 2.51로 컴파일한 실행 이미지는 약 23KB 정도이다. ARM 920T기반의 S3C2400 Evaluation Board에 다운로드하여 테스트하였다 [그림 5].



[그림 5] 개발환경

테스트 결과 C Standard Library에서 제공되는 printf() 함수의 기능들을 모두 잘 수행할 수 있었다.

5. 결론 및 향후 연구과제

실시간 운영 체제 iRTOS를 위한 MK_Printf() 함수를 구현함으로써 메모리 관리를 더 안정적으로 할 수 있었고, 커널 실행 이미지의 크기를 줄여서 실시간 운영 체제의 기본 개념에 적합하게 할 수 있었다.

향후 사용자 라이브러리로 MK_Printf() 함수 뿐만 아니라 파일로 출력을 할 수 있게 하는 MK_FPrintf() 함수나, 커널 내부에만 사용할 수 있게 하는 MK_KPrintf() 함수에 대한 부분은 앞으로 더 연구할 계획이다.

참고문헌

[1] http://www.inestech.com

[2] D. Comer, *Operation System Design VOL 1 : THE XINU APPROACH*, PRENTICE HALL, 1988.

[3] Jean, J. Labrosse, *uC/OS The Real-Time Kernel*, R&D Publications, 1992.

[4] BRETT HAMMOND, "Software Quality Vs. Dynamic Memory Allocation", Embedded Systems Programming, 1995.

[5] NIALL MURPHY, "Safe Memory Utilization", Embedded Systems Programming, 2000.

[6] 박희상, 안희중, 김용희, 이철훈, "Design and implementation of Memory Management Facility for Real-Time Operating System", iRTOS™, 정보과학회 춘계학술발표논문집, 제 29권 1호, 2002.