



[정오표](#)

[소스 코드와 참고 자료](#)

지은이 : 이만우
정가 : 20,000원
360쪽 / 판형 : A5 / 1판
출간일 : 2009년 4월 1일
ISBN-13 : 978-89-91268-57-9

저자소개

이만우

숭실대학교 컴퓨터학부를 졸업하고, 삼성전자 소프트웨어 멤버십에서 리눅스 디바이스 드라이버, 임베디드 운영체제, 분산처리 검색엔진 등을 개발했다. 지금은 폐간된 『프로그램 세계』에 리눅스 관련 강좌를 다수 연재했고, 현재는 삼성전자 반도체 총괄에 근무한다.

책소개

이 책은, 불필요한 설명은 이론서에 맡기고, 담백하게 개발 위주로 설명한다. 시대의 흐름이 유비쿼터스와 모바일 중심으로 흘러가는 시점에, 임베디드 환경에서 동작하는 운영체제를 개발해 보는 것은 충분한 가치와 의미가 있다고 본다.

- 숭실대학교 컴퓨터학부 김명호 교수

이 책은 작고 간단한 임베디드 운영체제를 만들어 보면서 운영체제의 원리를 가르쳐준다. 어려운 전공 서적에나 나올 법한 설명들은 잠시 제쳐두고, 일단 코딩을 하면서 작동 방식을 파악하다 보면 이론 역시 쉽게 이해하게 될 것이다. 학교에서 배운 '운영체제 이론'만으로는 부족한 학생부터 운영체제가 어떻게 작동하는지 궁금한 사람, 임베디드 펌웨어를 개발해야 하는 개발자에 이르기까지, 이 책을 따라가다 보면 어느덧 작은 운영체제를 완성하게 될 것이다.

이 책에서 다루는 내용

- ARM 아키텍처의 기본
- 부트로더 재활용
- **exception** 핸들링
- 컨텍스트 스위칭
- 메모리 관리
- 외부 인터럽트 제어
- 시스템 콜
- 태스크 간 통신
- 동기화
- 디바이스 드라이버
- 에뮬레이터용 실습 코드

목차

추천의 글

지은이의 글

1장 임베디드 운영체제

1.1 운영체제

1.1.1 프로세스 관리

1.1.2 저장장치 관리

1.1.3 네트워킹

1.1.3 사용자 관리

1.1.5 디바이스 드라이버

1.2 임베디드 운영체제

1.3 나빌눅스

1.4 실습 : 임베디드 개발 환경 구성

1.4.1 목표 플랫폼 정하기

1.4.2 리눅스에서 크로스 컴파일 환경 설정

1.4.3 윈도우에서 임베디드 개발 환경 구성

1.5 정리

2장 부팅하기

2.1 개발보드 선정하기

2.1.1 EX-X5 보드

2.2 이지보드에 나빌눅스 이미지를 올리는 방법

- 2.3 에뮬레이터 환경 구성
 - 2.3.1 qemu
 - 2.3.2 u-boot 설치
- 2.4 실습 : 이지보드에서 **hello world**를 출력하자
 - 2.4.1 이지부트의 소스코드 재활용
 - 2.4.2 커널 이미지 부팅하기
 - 2.4.3 링커 스크립트 수정
- 2.5 실습 : 에뮬레이터에서 **hello world**를 출력하자
 - 2.5.1 UART 주소 수정
 - 2.5.2 에뮬레이터에서 부팅하기
 - 2.5.3 ulmage 만들기
 - 2.5.4 램 디스크 이미지 만들기
 - 2.5.5 플래시 이미지 만들어 부팅하기
- 2.6 실습 : 윈도 환경에서 에뮬레이터 실행시키기
 - 2.6.1 시그윈에서 플래시 이미지 만들기
 - 2.6.2 윈도용 에뮬레이터 실행
- 2.7 정리

3장 LED 켜기

- 3.1 부트로더 코드 재활용
- 3.2 실습 : 1초마다 LED를 켜 보자
 - 3.2.1 이지부트에서 LED 관련 코드 분석
 - 3.2.2 나빌룩스에 LED 점멸 코드 추가
- 3.3 정리

4장 exception vector table 구성하기

- 4.1 ARM의 exception과 프로세서 동작 모드
- 4.2 ARM의 exception vector table
- 4.3 실습 : 이지부트를 수정하여 exception 핸들링 하기
- 4.4 실습 : 에뮬레이터에서 실행 - u-boot를 수정하여 exception 핸들링 하기
- 4.5 정리

5장 Software Interrupt Handler 구현하기

- 5.1 스택을 이용한 ISR과 태스크 간의 컨텍스트 스위칭
 - 5.1.1 ISR
 - 5.1.2 태스크-ISR 간 컨텍스트 스위칭
- 5.2 ARM 프로세서의 레지스터
 - 5.2.1 스택 포인터
 - 5.2.2 링크 레지스터
 - 5.2.3 spsr
- 5.3 실습 : Software Interrupt Handling
 - 5.3.1 실제 프로그램은 레지스터들을 어떻게 사용하는가
 - 5.3.2 태스크-ISR 간 컨텍스트 스위칭 코드 구현
 - 5.3.3 main 함수의 수정
 - 5.3.4 시스템 콜 번호의 추출
- 5.4 정리

6장 IRQ 핸들러 구현 : OS 타이머 사용하기

6.1 PXA255의 인터럽트 컨트롤러 계층

6.1.1 OS 타이머

6.1.2 인터럽트 컨트롤러 계층

6.1.3 ICMR

6.1.4 ICLR

6.1.5 ICCR

6.1.6 ICFP, ICIP

6.1.7 ICPR

6.1.8 인터럽트의 종류

6.2 msleep() 함수 분석

6.3 PXA255의 OS 타이머 레지스터 계층

6.3.1 OSMR

6.3.2 OSCR

6.3.3 OIER

6.3.4 OSSR

6.4 실습 : IRQ 핸들러 구현 - OS 타이머

6.4.1 OS 타이머 초기화 함수 작성

6.4.2 OS 타이머 시작 함수 작성

6.4.3 커널 main 함수 수정

6.4.4 IRQ 핸들러 함수 수정

6.4.5 전체 작업 코드

6.4.6 태스크-ISR 간 컨텍스트 스위칭 코드 작성

6.4.7 ARM9 아키텍처의 파이프라인

6.4.8 exception 핸들러에서 복귀 주소의 결정

6.4.9 OS 타이머가 발생하는 순서

6.4.10 빌드와 테스트

6.5 정리

7장 메모리 맵 구성

7.1 나빌눅스의 메모리 맵

7.2 실습 : 나빌눅스 커널의 스택 주소 초기화

7.3 실습 : 스택 초기화 주소 확인하기

7.4 정리

8장 메모리 관리자 구현하기

8.1 임베디드 운영체제에서의 사용자 태스크

8.1.1 태스크

8.1.2 메모리 관리자

8.2 실습 : 메모리 관리자 정의

8.2.1 자유 메모리 블록 정의

8.2.2 메모리 관리자 함수 정의

8.3 실습 : 메모리 관리자 함수 구현

8.3.1 메모리 관리자 커널 전역 변수 선언

8.3.2 메모리 분할 크기 설정

8.3.3 mem_init() 함수 설명

8.3.4 mem_alloc() 함수 설명

8.3.5 navilinux.h 파일 수정

8.3.6 Makefile 수정

8.4 정리

9장 태스크 관리자 구현하기

9.1 태스크 컨트롤 블록

9.1.1 태스크 컨텍스트 정보

9.2 사용자 태스크

9.2.1 사용자 태스크의 등록과 로딩

9.3 실습 : 태스크 관리자 정의

9.3.1 태스크 컨트롤 블록 정의

9.3.2 사용자 태스크의 컨텍스트 자료형 크기

9.3.3 태스크 관리자 구조체 정의

9.4 실습 : 태스크 관리자 함수 구현

9.4.1 태스크 관리자 커널 전역 변수 선언

9.4.2 cpsr의 초기 값 설정

9.4.3 task_init() 함수

9.4.4 task_create() 함수

9.5 실습 : 사용자 태스크의 추가

9.5.1 사용자 태스크 함수의 추가

9.5.2 navilinux.h 파일 수정

9.5.3 navilinux.c 파일 수정 - navilinux_init() 함수 추가

9.5.4 main() 함수 수정

9.5.5 Makefile 수정

9.6 정리

10장 컨텍스트 스위칭 구현하기

10.1 컨텍스트 스위칭과 스케줄러

10.1.1 멀티태스킹

10.1.2 컨텍스트 스위칭

10.1.3 스케줄러

10.2 실습 : 컨텍스트 스위칭 구현

10.2.1 IRQ 핸들러 수정

10.2.2 태스크 컨텍스트 백업

10.2.3 IRQ 핸들러 함수에 진입

10.2.4 태스크 컨텍스트 복구

10.3 스케줄러 구현

10.3.1 다른 운영체제의 스케줄링 정책

10.3.2 가장 기본적인 스케줄러

10.3.3 라운드로빈 스케줄러 구현

10.3.4 스케줄러 초기화 코드 작성

10.3.5 커널 main() 함수 수정

10.3.6 OS 타이머 핸들러 수정

10.3.7 navilinux.c 전체 내용 다시 보기

10.3.8 사용자 태스크 수정

10.3.9 빌드와 테스트

10.4 실습 : 사용자 스택 할당 검증

10.5 정리

11장 외부 인터럽트

11.1 PXA255의 GPIO 레지스터 계층

11.1.1 대표적인 외부 인터럽트 : 입력 장치

11.1.2 GPIO

11.1.3 PXA255 칩의 GPIO 인터럽트 처리

11.1.4 Edge Detect

11.1.5 PXA255 칩에서 GPIO를 설정하는 레지스터들

11.1.6 GPDR

11.1.7 GFER과 GRER

11.1.8 GEDR

11.1.9 GAFR

11.1.10 버튼 회로 연결

11.2 GPIO 인터럽트 처리

11.2.1 GPIO 초기화 코드 작성

11.2.2 초기화 함수 추가

11.2.3 인터럽트 처리 코드 추가

11.2.4 수정된 전체 코드

11.2.5 테스트

11.3 정리

12장 시스템 콜 구현하기

12.1 리눅스의 시스템 콜

12.1.1 fork() 시스템 콜

12.2 실습 : 시스템 콜 계층 추가

12.2.1 시스템 콜 커널 함수 작성

12.2.2 시스템 콜 초기화 함수 호출

12.2.3 시스템 콜 관련 헤더 파일 작성

12.2.4 사용자 태스크 함수 수정

12.2.5 시스템 콜 래퍼 함수 작성

12.2.6 Software Interrupt의 ISR 수정

12.2.7 Makefile 수정

12.3 실습 : 시스템 콜 추가 절차

12.4 정리

13장 태스크 간 통신 구현하기

13.1 IPC(Inter-Process Communication)

13.1.1 파이프

13.1.2 FIFO

13.1.3 메시지 큐

13.1.4 공유 메모리

13.1.5 임베디드 운영체제의 ITC

13.2 컨텍스트 스위칭 시스템 콜 만들기

13.2.1 블로킹 상태

13.2.2 사용자 태스크에서 호출 가능한 컨텍스트 스위칭 시스템 콜 구현

13.2.3 스케줄러 시스템 콜 추가

13.2.4 entry.S 파일 수정

13.2.5 사용자 태스크에서 스케줄러 호출 테스트

13.3 실습 : 메시지 관리자 정의

- 13.3.1 navilnux_msg.h 파일 작성
- 13.3.2 자유 메시지 블록
- 13.3.3 메시지 관리자
- 13.3.4 메시지 관리자 제어 함수들
- 13.4 실습 : 메시지 관리자 함수 구현
- 13.4.1 msg_itc_send(), msg_itc_get() 함수 구현
- 13.4.2 navilnux.h 수정
- 13.4.3 navilnux_init() 함수 수정
- 13.5 실습 : 시스템 콜 계층에 ITC 함수 등록
- 13.5.1 시스템 콜 번호 추가
- 13.5.2 시스템 콜 함수 프로토타입 선언
- 13.5.3 시스템 콜 함수 본체 작성
- 13.5.4 시스템 콜 래퍼 함수 프로토타입 선언
- 13.5.5 시스템 콜 어셈블리어 래퍼 함수 작성
- 13.5.6 시스템 콜 C 래퍼 함수 프로토타입 선언
- 13.5.7 시스템 콜 C 래퍼 함수 본체 작성
- 13.5.8 ITC 테스트
- 13.6 정리

14 동기화 구현하기

- 14.1 세마포어
- 14.1.1 세마포어 구현하기
- 14.1.2 메시지 관리자 코드 수정
- 14.1.3 세마포어 함수 구현
- 14.1.4 새로운 시스템 콜 번호를 세마포어에 할당
- 14.1.5 시스템 콜 함수의 프로토타입 선언
- 14.1.6 시스템 콜 함수 작성
- 14.1.7 시스템 콜 래퍼 함수의 프로토타입 선언
- 14.1.8 시스템 콜 어셈블리어 래퍼 함수 작성
- 14.1.9 시스템 콜 C 언어 래퍼 함수 작성
- 14.1.10 사용자 태스크에서 세마포어 사용 테스트
- 14.2 뮉텍스
- 14.2.1 바이너리 세마포어와 뮉텍스의 차이
- 14.2.2 실습 : 뮉텍스 구현하기
- 14.2.3 메시지 관리자 수정
- 14.2.4 뮉텍스 함수 구현
- 14.2.5 뮉텍스에 시스템 콜 번호 할당
- 14.2.6 시스템 콜 함수 작성
- 14.2.7 시스템 콜 래퍼 함수 작성
- 14.2.8 사용자 태스크에서 뮉텍스 테스트
- 14.3 실습 : 시간 지연 함수 구현하기
- 14.3.1 커널 카운터 추가
- 14.3.2 sleep() 함수 구현
- 14.3.3 태스크 컨트롤 블록 수정
- 14.3.4 sleep() 함수 작성
- 14.3.5 수정된 sleep() 함수를 이용한 뮉텍스 테스트
- 14.4 정리

15 메모리 동적 할당 구현하기

15.1 메모리 동적 할당 설계

15.1.1 동적 할당에 사용할 메모리 영역

15.1.2 구현의 범위

15.1.3 메모리 풀

15.2 실습 : 메모리 동적 할당 구현

15.2.1 메모리 관리자 수정

15.2.2 동적 할당 전략

15.2.3 free() 함수 구현

15.2.4 malloc() 함수 구현

15.2.5 시스템 콜에 등록

15.2.6 메모리 동적 할당 테스트

15.3 정리

16 디바이스 드라이버 구현하기

16.1 디바이스 드라이버

16.1.1 리눅스 캐릭터 디바이스 드라이버 계층을 차용

16.2 실습 : 디바이스 드라이버 관리자 정의

16.2.1 fops 구조체

16.2.2 자유 디바이스 드라이버 블록

16.3 실습 : 디바이스 드라이버 관리자 구현

16.3.1 drv_init() 함수

16.3.2 drv_register_drv() 함수

16.3.3 시스템 콜에 등록

16.4 실습 : 디바이스 드라이버 추가하기

16.4.1 LED와 스위치를 디바이스 드라이버로 제어

16.4.2 IRQ 핸들러 벡터를 커널에 추가

16.4.3 read(), write() 함수 구현

16.4.4 IRQ 핸들러 함수

16.4.5 mydrv_open() 함수

16.4.6 mydrv_close() 함수

16.4.7 mydrv_read() 함수

16.4.8 mydrv_write() 함수

16.4.9 사용자 디바이스 드라이버를 커널에 등록

16.4.10 사용자 디바이스 드라이버를 테스트

16.5 정리

17 마치며

17.1 프로젝트 종료

17.2 나빌눅스의 파일 구성

17.2.1 entry.S, navilinux.c, navilinux.h

17.2.2 navilinux_memory.c, navilinux_memory.h

17.2.3 navilinux_task.c, navilinux_task.h

17.2.4 navilinux_user.c, navilinux_user.h

17.2.5 navilinux_sys.c, navilinux_sys.h, syscalltbl.h, navilinux_lib.S, navilinux_clib.c, navilinux_lib.h

17.2.6 navilinux_msg.c, navilinux_msg.h

17.2.7 navilinux_drv.c, navilinux_drv.h, mydrv.c

17.3 나빌눅스의 계층

17.4 맺음말

17.4.1 운영체제의 개념, 이론 그리고 구현

17.4.2 임베디드 개발 환경에 대한 경험

17.4.3 ARM 아키텍처에 대한 대략적 이해

17.4.4 마치며

찾아보기

약어표