실시간 운영체제 UbiFOS™에서의 POSIX 적용*

송예진 0, 이철훈
충남대학교 컴퓨터공학과
{syj3565 0, clee}@cnu.ac.kr

The Application of POSIX on UbiFOS™ Real-Time Operating System

Ye-Jin Song 0, Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University

요 약

임베디드 시스템은 그 용도에 따라서 다양한 기능을 수행하도록 설계된 전용 컴퓨팅 시스템으로 최근 임베디드 시스템의 보급이 많아지고 다양화됨에 따라 응용 프로그램의 재사용성 향상을 위해 운영체제의 개발 시 표준 API 를 사용하여 표준화를 해야 하는 필요성이 대두되고 있다.

따라서 본 논문에서는 실시간 운영체제인 UbiFOS™에 개방형 운영체제 인터페이스인 POSIX 의 API 를 적용시켜 설계 및 구현하였다.

1. 서 론

특정 목적을 수행하기 위해서 하드웨어와 소프트웨어를 밀접하게 통합시킨 컴퓨팅 장치를 임베디드 시스템이라 하며 최근 임베디드 시스템은 그 특정 목적에 따라서 다양한 형태로 개발되고 발전되어 가고 있다.

임베디드 시스템은 하드웨어 자원이 매우 한정적인데, 이와 같은 한정적인 하드웨어 자원을 효율적으로 관리하기 위하여 실시간 운영체제(Real-Time Operating System)를 많이 사용하고 있다.

실시간 운영체제는 시스템이 정확한 계산 결과를 출력해야 한다는 기능정확성과 그 결과를 주어진 시간 안에 얻을 수 있어야 한다는 시간정확성에 의존적이다 [2].

최근 임베디드 시스템의 보급이 많아지고 다양화되고 있다. 여기에 탑재되는 응용 프로그램도 장치에 맞게 다양화되고 있으며, 따라서 재사용성이 뛰어난 응용 프로그램이 개발이 절실히 필요하다. 응용 프로그램의 재사용성 향상을 위해서 개발 시 표준 API 를 사용하는 것이 바람직하며 기반이 되는 운영체제는 당연히 표준 API 를 제공해야 한다.

본 논문에서는 실시간 운영체제인 UbiFOS™에 개방형 운영체제 표준 인터페이스인 POSIX API 를 적용시켰다.

본 논문의 구성은 2 장에서 관련연구로 실시간 운영체제 UbiFOS™의 전반적인 구성과 POSIX 에 대하여 기술하고, 3 장에서는 실시간 운영체제 UbiFOS™에서의 POSIX 적용에 대한 내용을 기술한다. 4 장에서는 테스트 환경 및 결과를 기술하고, 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다.

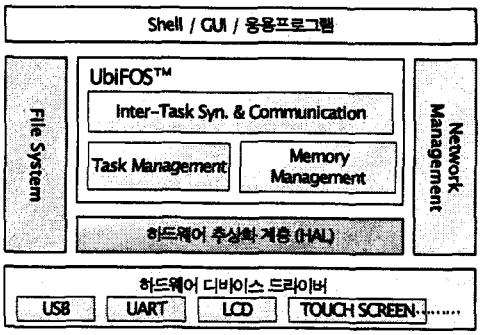
2. 관련 연구

2.1 UbiFOS™

실시간 운영체제 UbiFOS™ 은 멀티태스킹을 지원하기 위해 0 부터 255 까지 256 단계의 우선순위를 제공하는 선점형 스케줄러로 동일 우선순위에 대해 FIFO 와 Round-Roubin 기법을 제공한다 [1][3].

이 밖에도 Task 간의 통신을 위해 Message Mailbox, Message Queue, Message Port, Task Port, Signal 을 제공하고 Task 간의 동기화를 위해 Semaphore 와 Event Flag 를 제공하고 있다.

* 본 논문은 국방과학연구소의 실시간 운영체제 인터페이스용 미들웨어 연구과제의 수행결과임



[그림 2-1] UbiFOS™ 전체 구성도

2.2 POSIX

POSIX (Portable Operating System Interface for UNIX)란 개방형 운영체제 인터페이스로 서로 다른 UNIX OS 의 공통되는 API 를 정리하여 이식성이 높은 UNIX Application 을 개발하기 위한 목적으로 IEEE 가 책정한 일련의 Application Interface 규격이다.

표준화에 관한 필요성은 컴퓨터를 사용하고 있는 기업들이 다시 코딩 하지 않고서도 다른 컴퓨터 회사가 만든 컴퓨터 시스템에도 운영할 수 있도록 호환성이 있는 프로그램을 개발하기 원하는 데에서 기인했다.

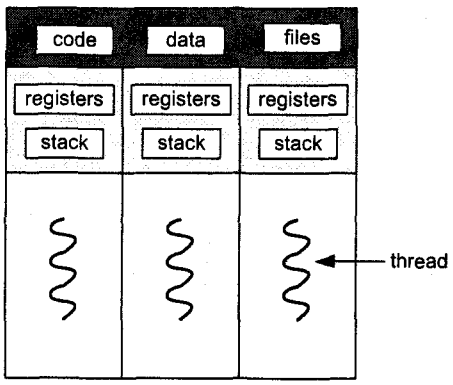
규격의 내용은 커널로의 C 언어 인터페이스인 System call 뿐만 아니라, Process Environment, File and Directory, System Database 등 다양한 분야를 아우른다 [4][5].

3. 실시간 운영체제 UbiFOS™에서 POSIX 적용의 설계 및 구현

POSIX 는 [그림 3-1]에서와 같이 하나의 Process 안에 여러 개의 Thread 가 존재하는 Multithreaded Processes 구조로 되어 있고, 하나의 Thread 마다 자신만의 문맥을 저장하는 메모리 공간인 registers 와 stack 영역을 가지며 하나의 Process 안에 존재하는 Thread 들은 code, data, files 영역을 공유한다.

그리고 UbiFOS™은 Multitasking 구조로 되어있고, Task 는 흔히 Thread 와 같은 의미로 사용한다. Multitasking 이란 하나의 CPU 에서 여러 개의 Task 가 동시에 수행될 수 있도록 하는 방식이며, 각 Task 는 자신의 TCB(Task control block)에 PC(Program Counter), register, stack, function call, 우선순위, Signal Handler 등을 저장한다.

본 논문에서는 POSIX 에서 제공하는 Thread 관련 API 를 UbiFOS™의 Task API 에 맞춰주었고 POSIX 와 UbiFOS™의 Semaphore, Mutex, Message Queue, Signal, Timer 의 구조체 제어블럭을 제어하여 UbiFOS™의 각 모듈 별 API 를 POSIX API 와 호환 가능하도록 설계 및 구현하였다.



[그림 3-1] Multithreaded Process 구조

3.1 Thread

Thread 는 세미 Process 라고 불리며 POSIX 의 Theard 는 흔히 pthread 라고 한다. Thread 는 커널에 의해 시간이 분할되어 실행 되고, 같은 메모리 공간을 공유한다.

UbiFOS™에서 Task 의 구조체 제어블럭을 Thread 의 구조체 제어블럭으로 제어하여 유사한 기능을 수행하는 함수들을 사용하여 POSIX 와 호환될 수 있도록 구현하였다.

UbiFOS™에서는 우선순위 기반으로 Task 를 생성하도록 되어있는 반면 POSIX 에서는 같은 우선순위의 Thread 를 생성하여 Round-Roubin 스케줄링을 하도록 되어있다.

MK_CreatePthread 함수에서는 POSIX 의 pthread_create 함수의 attr 값을 NULL 로 우선순위 7 인 Task 를 생성하여 Round-Roubin 또는 FIFO 방식으로 수행되도록 하였고, MK_CreateTask 함수를 통해 생성한 Task 를 MK_Resume 함수를 통해 ReadyList 에 넣어주고 MK_Start 함수를 통해 컨텍스트 스위치(Context Switch)가 가능하게 하여 해당 함수를 수행할 수 있도록 설계하였다.

[표 3-1] Thread 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
pthread_create	MK_CreatePthread
⇒ Thread 를 생성하는 함수	
pthread_join	MK_Join
⇒ 해당 Thread 가 종료할 때까지 기다리고 자원을 해제	
pthread_detach	MK_Detach
⇒ Process 에서 생성된 Thread 를 분리시키는 함수	
pthread_exit	MK_DeleteTask
⇒ Thread 를 종료하는 함수	
pthread_self	MK_GetCurrentTask
⇒ 현재 수행중인 Thread 의 식별자를 얻는 함수	

3.1.1 Thread 의 attribute

다음은 pthread 의 attribute 객체 변수인 attr 과 관련된 함수들이다. pthread_attr_setstackaddr 와 pthread_attr_setstacksize 함수는 MK_CreateTask 함수의 인자 값을 control 하여 사용될 수 있도록 구현하였다.

[표 3-2] Thread 의 attribute 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
pthread_attr_init	MK_AttrInit
⇒ attr 를 디폴트 값으로 초기화 하는 함수	
pthread_attr_destroy	MK_AttrDestroy
⇒ attr 을 제거해주는 함수	
pthread_attr_getscope	MK_AttrGetScope
⇒ 어떤 영역에서 다뤄지고 있는지 얻어오는 함수 (PTHREAD_SCOPE_SYSTEM, PTHREAD_SCOPE_PROCESS)	
pthread_attr_setscope	MK_AttrSetScope
⇒ 어떤 영역에서 작동하게 할 것인지 결정하는 함수 (PTHREAD_SCOPE_SYSTEM, PTHREAD_SCOPE_PROCESS)	
pthread_attr_setschedpolicy	MK_SetSchedulePolicy
⇒ 스케줄 정책을 설정하는 함수	
pthread_attr_getschedpolicy	MK_GetSchedulePolicy
⇒ 스케줄 정책을 얻어오는 함수	
pthread_attr_setstackaddr	MK_CreateTask
⇒ 스택 주소를 설정하는 함수	
pthread_attr_setstacksize	MK_CreateTask
⇒ 스택의 크기를 설정하는 함수	

3.1.2 Thread 의 Mutex

Mutex 는 여러 개의 Thread 가 공유하는 데이터를 보호하기 위해 사용되고 보호하고자 하는 데이터를 다루는 코드 영역을 단지 한번에 하나의 Thread 만 실행 가능하도록 하는 방법으로 공유 데이터를 보호한다.

POSIX 에서는 Mutex 객체를 선언하여 생성 후 초기화를 거쳐 사용하도록 되어있다.

UbiFOS™에서는 별도의 Mutex 함수를 제공하지 않는데 Semaphore 의 count 값을 1 로 셋팅하여 Binary Semaphore 를 통해 동기화와 상호배제를 제공하도록 설계하였다.

pthread_mutex_lock 과 pthread_mutex_trylock 함수는 MK_SemaphorePend 함수에 옵션을 사용하여 같은 기능을 수행할 수 있도록 구현하였다.

[표 3-3] mutex 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
pthread_mutex_init	MK_CreateSemaphore
⇒ pthread 의 Mutex 를 생성하는 함수	
pthread_mutex_destroy	MK_DeleteSemaphore
⇒ pthread 의 Mutex 를 제거하는 함수	
pthread_mutex_lock	MK_SemaphorePend
⇒ 임계 영역에 진입하는 함수	
pthread_mutex_trylock	MK_SemaphorePend
⇒ 임계 영역에 진입하는 함수	
pthread_mutex_unlock	MK_SemaphorePost
⇒ 임계 영역을 마치고 빠져나오는 함수	

3.1.3 Thread 의 취소

Thread 의 취소는 하나의 Process 안에서 실행 중인 다른 Thread 를 종료시키기 위한 목적으로 사용된다. 취소 요청을 받은 Thread 는 설정에 따라서 곧 바로 종료할 수도 있고 취소 지점 (cancellation point)을 벗어난 후 종료 할 수도 있다.

다음은 pthread 의 취소와 관련된 함수들이다.

[표 3-4] Thread 의 취소 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
pthread_cancel	MK_DeleteTask
⇒ 해당 Thread 에게 종료 요청을 보내는 함수	
pthread_setcancelstate	MK_SetCancelState
⇒ 취소 상태를 활성화, 비활성화로 변경시키는 함수 (PTHREAD_CANCEL_ENABLE, PTHREAD_CANCEL_DISABLE)	
pthread_setcanceltype	MK_SetCancelType
⇒ 취소 타입을 결정하는 함수 (PTHREAD_CANCEL_ASYNCHRONOUS, PTHREAD_CANCEL_DEFERRED)	

3.2 Semaphore

Semaphore 는 상호배제와 thread 간 동기화라는 두 가지 요구사항을 충족시키는 실시간 운영체제의 매우 중요한 자원이다.

다음은 Semaphore 와 관련된 함수들이다.

sem_wait 과 sem_trywait 함수는 MK_SemaphorePend 함수에 옵션을 사용하여 같은 기능을 수행할 수 있도록 구현하였다.

[표 3-5] Semaphore 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
sem_init	MK_CreateSemaphore
⇒ Semaphore 를 생성하는 함수	
sem_destroy	MK_DeleteSemaphore
⇒ Semaphore 를 제거하는 함수	
sem_post	MK_SemaphorePost
⇒ Semaphore 에 자원을 반납하는 함수	
sem_getvalue	MK_GetSemaphoreCount
⇒ 현재 Semaphore 의 count 값을 얻어오는 함수	
sem_wait	MK_SemaphorePend
⇒ Semaphore 에서 자원을 요구하는 함수	
sem_trywait	MK_SemaphorePend
⇒ Semaphore 에서 자원을 요구하는 함수 (블록되지 않고 바로 에러 Message 를 출력)	

3.3 Message Queue

Message Queue 는 thread 간 통신을 위한 기능 중 하나로, thread 에서 다른 thread 로 여러 개의 Message 를 전달하려 할 때 사용한다.

다음은 Message Queue 와 관련된 함수들이다.

[표 3-6] Message Queue 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
mq_open	MK_CreateMsgQueue
⇒ Message Queue 를 생성하는 함수	
mq_close	MK_DeletrMsgQueue
⇒ Message Queue 를 제거하는 함수	
mq_receive	MK_MsgQueuePost
⇒ Message Queue 로부터 Message 를 받는 함수	
mq_send	MK_MsgQueuePend
⇒ Message Queue 에 Message 를 전달하는 함수	

3.4 Signal

Signal 은 하나 이상의 thread 에게 event 를 알리기 위해서 사용된다. POSIX 의 signal 은 UbiFOS™의 Signal 이 아닌 Event Flag 와 기능이 유사하여 Event Flag 의 API 와 매핑되도록 설계하였다.

다음은 Signal 과 관련된 함수들이다.

pthread_sigmask 와 pthread_kill 함수는 MK_SignalSend 함수에 옵션을 사용하여 같은 기능을 수행할 수 있도록 구현하였다.

[표 3-7] Signal 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
pthread_sigmmask	MK_SetEvent
⇒ 특정 Thread 만 Signal 을 받도록 하는 함수	
pthread_kill	MK_SetEvent
⇒ Thread 로 해당 번호의 Signal 을 전달하는 함수	
sigwait	MK_EventPend
⇒ Signal 의 전달을 동기적으로 기다리는 함수	

3.5 Timer

실시간 운영체제는 서론에서 언급한 것과 같이 기능 정확성과 시간정확성에 의존적이기 때문에 Timer 는 실시간 운영체제에서 중요도가 매우 높은 모듈이라 할 수 있다.

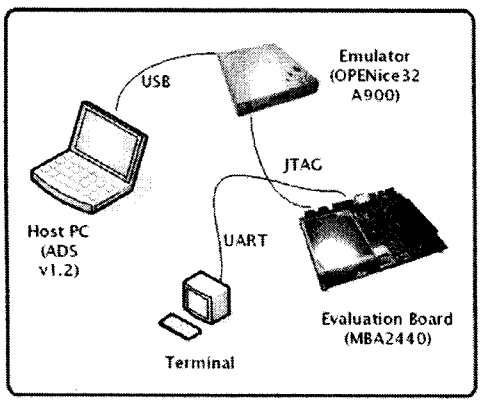
다음은 Timer 와 관련된 함수들이다.

[표 3-8] Timer 관련 함수의 매핑

POSIX 함수	UbiFOS™ 함수
timer_create	MK_CreateTimer
⇒ Timer 를 생성하는 함수	
timer_delete	MK_DeleteTimer
⇒ Timer 를 제거하는 함수	
timer_gettime	MK_GetRemainingTime
⇒ 현재 Timer 의 값을 얻어오는 함수	
sleep	MK_Sleeptick
⇒ 주어진 시간만큼 thread 를 지연시키는 함수	

4. 테스트 환경 및 결과

본 논문에서는 실시간 운영체제인 UbiFOS™에 개방형 운영체제 표준 인터페이스인 POSIX API 를 적용시켰다. 테스트 환경으로는 UbiFOS™의 커널이 탑재된 ARM920T 기반의 MBA2440 보드에서 수행하였고, 컴파일러는 ARM ADS 1.2 를 사용하였다.



[그림 4-1] 실험 환경

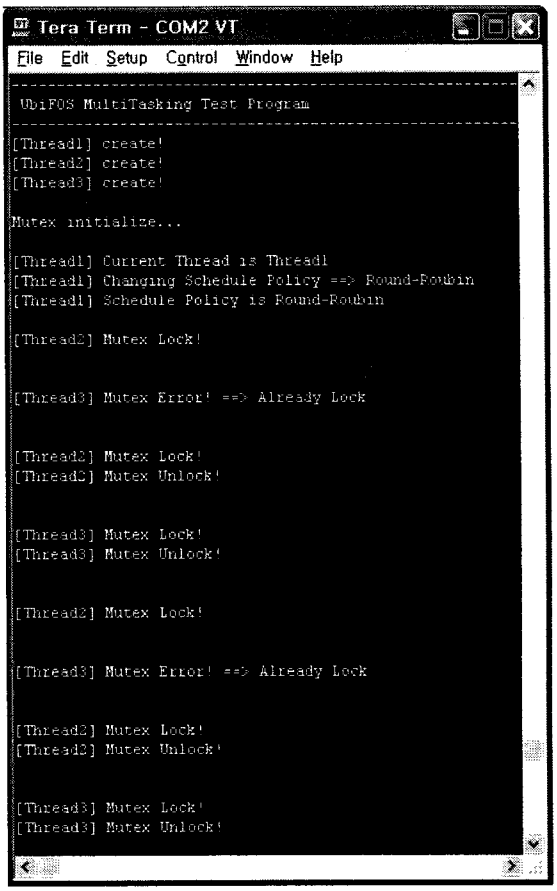
[그림 4-2]는 UbiFOS™에 POSIX API 를 적용시킨 결과로 main 프로세서에서 pthread_create 함수를 통해 Thread1, Thread2, Thread3 를 생성하고 Mutex initialize 를 수행한 후 pthread_join 함수를 통해 Thread1, Thread2, Thread3 를 해제한다.

Thread1 에서는 pthread_self 함수를 호출하여 현재 수행중인 Thread 를 리턴하고 pthread_attr_setschedpolicy 함수와 pthread_attr_getschedpolicy 함수를 호출하여 스케줄 정책을 Round-Roubin 으로 설정한 후, 설정한 스케줄 정책을 얻어서 출력한다.

Thread2 와 Thread3 은 Mutex 를 통해 서로 동기화를 제공하는데, Thread2 에서는 무한루프를 통해 lock 또는 lock, unlock 을 교대로 수행한다.

Thread3 에서는 무한루프를 통해 Thread2 에서 lock 을 수행했을 경우 에러 Message 를 출력하고 lock, unlock 을 수행했을 경우에는 Thread3 도 정상적으로 lock, unlock 과정을 수행한다.

Thread1 은 한번 수행 후 join 함수를 통해 해제되게 되며, Thread2, Thread3 은 Round-Roubin 을 통해 교대로 수행하게 된다.



[그림 4-2] 테스트 결과

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제인 UbiFOS™에 개방형 운영체제 표준 인터페이스인 POSIX API 를 적용시켰다. 논문의 주 목적이 호환성이 있는 프로그램을 개발하여 실시간 운영체제를 표준화 하는데 있기 때문에 향후 과제로는 POSIX 와 UbiFOS™의 좀더 견고한 호환이 필요하다.

참고 문헌

[1] Embedded System & RTOS, http://www.aijisystem.com

[2] David Stepner 외 2 명, "Embedded Application Design Using a Real-Time OS", IEEE, 1999

[3] 박희상, 정명조, 조희남, 이철훈, "Design of Open-Architecture Real-Time OS Kernel, 한국정보과학회, 제 31 권 제 1 호, p.163-165, 2002.04.

[4] David R. Butenhof, "Programming with POSIX Threads"

[5] Bill O. Gallmeister, "POSIX.4 Programming for the Real World"