실시간 리눅스 기반의 회전익 무인항공기 제어 소프트웨어 개발

박기석*, 박중희*, 위영준*, 박정근*, 문창주*
*건국대학교 항공우주정보시스템공학과
e-mail : katsu12@konkuk.ac.kr

Real-time Linux based Rotor UAV Control Software Development

Kiseok Park*, Joong Hee Park*, Young Jun Wie*
Jungkeun, Park* Chang Joo Moon*
*Dept. of Aerospace Information Engineering, Konkuk University

요 약

본 논문은 실시간 운영체제인 Xenomai 를 사용하여, 회전익 무인항공기 소프트웨어 개발에 대한 내용을 설명하고 있다. 실시간 운영체제 사용하여 고정 순위 우선 스케줄링을 채택함으로써 데드라인의 타이밍(Timming) 결정성을 보장하였고, 이기종 시스템과의 호환성과 확장성을 고려하여 POSIX API 를 사용하여 멀티 쓰레드를 구현하였다. 또한 실시간 드라이버 모델(RTDM : Real-Time Driver Model)을 사용하여 획득한 데이터를 실시간 전송이 가능하도록 하였다. 본 논문은 실시간 운영체제를 무인항공기에 적용하고 구현된 비행제어 컴퓨터와 제어 소프트웨어를 비율 단조 스케줄링을 적용하여 무인항공기의 쓰레드들의 응답 속도 및 안정성을 보장하는 방안을 제시하였다.

1. 서론

최근 무인항공기는 군사정찰, 감시 등을 비롯하여 기상관측, 통신, 해양탐사, 항공촬영 등 그 활용범위가 다양하여 많은 연구개발이 이루어지고 있다.[1] 무인항공기의 제어 소프트웨어는 모든 임무를 실시간으로 처리해야 함으로 통상적으로 실시간 운영체제(RTOS: Real-Time Operating System)를 사용한다. 상용 RTOS 는 상대적으로 안정된 기능과 체계적인 기술지원을 제공한다. 그러나 탑재 소프트웨어가 특정 RTOS 에 종속이 되고 향후 확장성에 제한을 받게 된다. 특히 무인항공기의 경우 대부분이 군용으로 장기간에 걸쳐 개발이 되고 사용됨으로 상용 RTOS 의 사용은 유지보수 측면에서 문제점이 된다. 이러한 이유 때문에 무인기 탑재 소프트웨어를 공개 소프트웨어인 리눅스를 이용한 개방형 시스템으로 구축하려는 시도들이 이루어지고 있다.[1]

실시간 임베디드 리눅스는 실시간성 제공을 위해 리눅스 커널을 최적화한 비 상용 운영체제이다. 임베디드 리눅스의 가장 뛰어난 특징은 확장성, 범용성이다. 또한 리눅스는 공개 소스이므로 전세계의 개발자들에 의해 지속적으로 발전하고 있어 최신의 컴퓨터 기술을 반영할 수 있다. 리눅스의 이러한 장점들은 무인항공기 관련 기술 개발과 유지보수 측면에서 매우 긍정적이라 할 수 있다.[3]

본 논문에서는 실시간 임베디드 운영체제인 Xenomai 기반으로 소형 회전익 무인항공기 제어 소프트웨어 설계 및 구현을 하였다. 임베디드 리눅스에 실시간 개발 프레임워크인 Xenomai 패치 및 포팅 통하여, 무인항공기에 필요한 센서 데이터들을 실시간으로 수집하고, RTDM(Real-Time Driver Model)을 통하여 실시간 데이터 전송 하였다.

본 논문의 구성은 다음과 같다. 2 장 배경연구에서는 논문을 이해하는데 필요한 Xenomai 와 POSIX(Portable Operating System Interface)에 대해서 언급한다. 3 장에서는 무인항공기 제어 시스템 구성, 탑재 소프트웨어 설계 및 구현에 대해 설명한다. 4 장에서는 Xenomai 패치 및 쓰레드 구현에 대하여 언급하고, 5 장에서 결론 및 향후 과제을 언급한다.

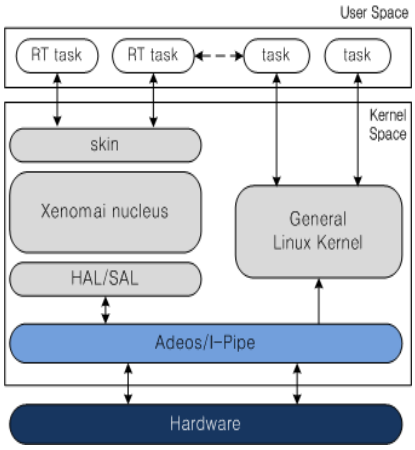
2. 배경연구

2.1 Xenomai

Xenomai 는 실시간 보장을 위해 리눅스 커널과 융합되는 실시간 개발 프레임워크로 리눅스 커널에 리

얼 타임 응용프로그램, 다양한 인터페이스, GNU/LINUX 환경을 제공한다. Xenomai 는 기본적으로 Native API 와 VxWorks, pSOS+ VTRX, uITRON, 그리고 POSIX API 등을 제공한다.[2][7]

(그림 1)은 Xenomai 의 아키텍쳐를 나타내고 있다. Adeos/I-Pipe(Interrupt Pipeline)은 리눅스 커널과 관계없이 Xenomai 에게 하드웨어 레벨의 인터럽트를 빠르게 전달하고 HAL(Hardware Abstraction Layer)은 기계 의존적인 코드의 의존성을 해결하며, 특정 Xenomai 포트를 구현하는데 필요한 모든 CPU 및 플랫폼 정보를 수집한다. 또한 뉴클리우스(Nucleus) 및 스킨을 간편하게 만들 수 있게 시스템 SAL(System Abstraction Layer)과 결합시키는 역할도 한다.[7][8]



(그림 1) Xenomai 구조

2.2 POSIX

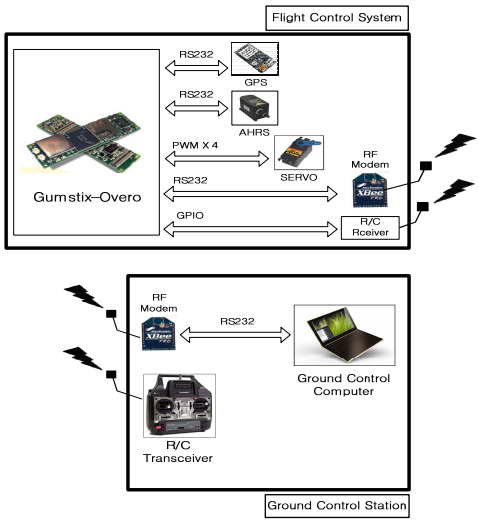
POSIX 는 서로 다른 유닉스 운영체제의 공통 API 를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE 가 책정한 애플리케이션 인터페이스 규격이다. 이러한 표준 인터페이스 규격은 개발된 소프트웨어의 호환성과 재사용성을 향상시킨다. 쓰레드를 사용하여 작업을 분리해 놓으면, 반응을 예측 가능하도록 구현하기가 수월하기 때문에 쓰레드는 모든 종류의 실시간 프로그래밍에 적합하다. 또한 유닉스 시스템이 지원하지 않는 실시간 스케줄링 옵션을 지원하며, 옵션을 바꾸어 시스템에 맞게 적용할 수 있다. POSIX 를 사용함으로서 메모리 낭비나 시스템부하를 줄일 수 있고, 프로그램 내의 전역변수나 디스크립터 등을 공유 할 수 있다. 본 논문에서 사용하는 POSIX API 는 쓰레드 관리, 메모리 힙(heap), 뮤텍스(mutex), 인터럽트 관리, 쓰레드 스케줄링, 메세지 큐 서비스등을 제공한다.

3. 비행제어 시스템 개발 및 구현

3.1 시스템 구성도

무인항공기 제어 시스템은 크게 비행제어 시스템과 지상관제 시스템으로 구분된다.[4][5] (그림 2)는 회전익 무인항공기의 시스템 구성도를 나타낸다. 비행 제어 시스템은 비행제어 컴퓨터, 항법센서 (AHRS), 위치센서 (GPS), 서보 모터, 무인항공기와 지상제어 시스템 간의 데이터 송수신을 위한 RF(Raiod Frequency) 모뎀, 무선영상 모듈, 외부 인터페이스, 내부 통신 채널 등으로 구성되어있다.

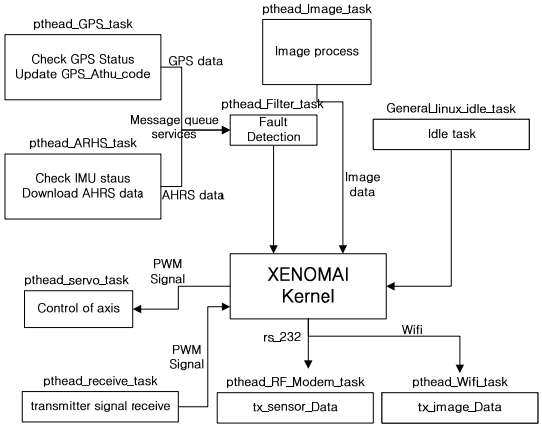
비행제어 컴퓨터의 프로세서는 텍사스 인스트루먼트 사의 Gumstix-Overo 프로세서이다. 항법센서와 위치센서는 비행제어 컴퓨터의 RS-232 시리얼 통신 채널을 통하여 데이터를 받는다. 위치센서 GPS(Global Position System)는 절대 좌표값을 받아 무인항공기의 현재위치를 결정하고, 항법센서 AHRS(Attitude and heading reference System)는 헤딩값(heading) 및 롤(roll), 피치(pitch), 요(yaw) 3 축 값을 제어하여 항공기의 자세를 제어한다.[5] 조종면(actuator)을 구동하기 위해서 3 개의 서보 모터가 사용되며, PWM(Pulse Width Modulation)신호 채널로 서보 모터를 제어하게 된다. 또한 지상의 R/C 전송기의 신호를 무인항공기의 R/C 수신기를 통해 수신 후, PWM 신호 채널을 통하여 서보모터를 제어할 수 있다.



(그림 2) 회전익 무인항공기시스템 구성도

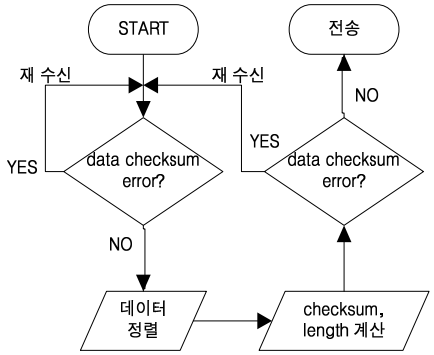
3.2 소프트웨어 구조

(그림 3)은 Xenomai 의 POSIX API 를 이용하여 나타낸 RUAV 소프트웨어 구조이다.



(그림 3) 회전익 무인기 소프트웨어 구조

시스템내의 모든 데이터 획득 및 처리과정은 실시간 쓰레드화되어 처리된다. 각각의 쓰레드는 우선 순위에 따라 스케줄링 된다. GPS, ARHS 데이터 획득 쓰레드는 필터(Filter) 쓰레드를 거쳐 Xenomai 로 전달된다. 필터 쓰레드는 수신된 데이터의 체크섬을 계산하고 지상국 전송 데이터 구조로 재 배열한다. 재 배열된 데이터 구조의 체크섬을 계산 후, 오류가 없으면 지상국으로 전송한다. (그림 4)는 필터 쓰레드의 오류검출 알고리즘을 설명하고 있다. PWM 쓰레드는 타이머를 통하여 신호를 전달 및 처리한다.



(그림 4) 필터 쓰레드 오류검출 알고리즘

1) 실시간 멀티 쓰레드 구조

POSIX API 는 사용자 공간(User space)에서 쓰레드 구현을 가능하게 한다. (그림 5)는 멀티 쓰레드방식의 GPS 쓰레드 기본 구조를 나타낸다. 사용자 공간에서 쓰레드를 사용하기 위해서는 pthread.h 를 포함하고, 우선순위를 선언하여야 한다. 쓰레드의 일이 종료되면 다음 주기까지 대기상태로 들어가며, 다른 쓰레드가 실행되어진다. 해당 쓰레드의 주기가 돌아왔을때, 실행되는 쓰레드가 없을 경우 실행상태로 전이되어 일을 수행한다. 하지만 우선 순위가 높은 쓰레드가 실행상태일 경우에는 블럭(block)되고, 실행 쓰레드가 종료 됐을때 다시 실행되어진다.[3]

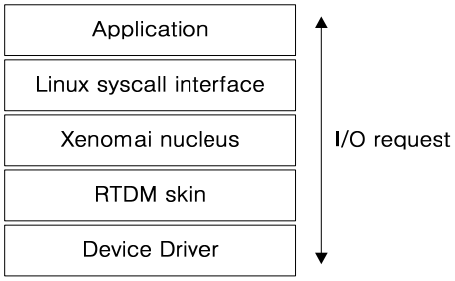
```
void *thread_gps() {
      int fd;
      int gps_flag,checksum_end=0;

      while(1) {
            /*gps receive*/
            /*gps parcing*/
            /*send to filter thread*/
            sleep(1)
      }
int main() {
      pthread_t p_thread[3];
      int thr_id, status = 0;

      thr_id = pthread_create(&p_thread[0], NULL, thread_gps, NULL);
      pthread_join(p_thread[0], (void **)&status);

      return 0;
}
```

(그림 5) GPS pthread 구조

2) RTDM(Real-Time Driver Model)

RTDM 을 통한 실시간 드라이버(Real-Time Driver)의 사용은 실시간 시리얼 통신이 가능하게 해준다. 커널 환경 셋팅시 RTDM 을 선택 하여, 실시간 파일 디스크립터를 사용할 수 있다.[2][6]

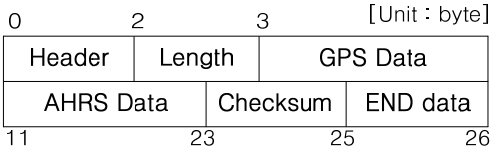
RTDM 은 CAN(Control Area Network), 실시간 IPC(Inter-Process Communication), 시리얼, A/D(Analog to Digital) 드라이버를 제공하며 본 논문에서는 RT_COM 을 이용하여 시리얼 드라이버를 사용하였다. RT_COM 은 보(baud)레이트, 패리티등을 설정 할수 있으며, FIFO 방식으로 데이터 전송을 하였다. (그림 6)는 RTDM 의 계층을 보여주고 있다. RTDM 은 Xenomai 커널과 디바이스 드라이버 중간에서 실시간 I/O 요청 처리 및 인터페이스 역할을 한다.



(그림 6) RTDM 계층도

3) 지상국 전송 데이터 구조

필터 쓰레드를 통과한 데이터들은 지정된 데이터 프레임에 따라 재 정비 된다. GCS(Ground Control System)와 통신을 위한 RUAV 전송 데이터 구조는 (그림 7)과 같다. RUAV 전송 데이터 구조는 Header 2byte, 데이터 length, GPS, AHRS, Checksum, END data 순서로 지상국으로 전송이 된다.



(그림 7) RUAV Data Structure

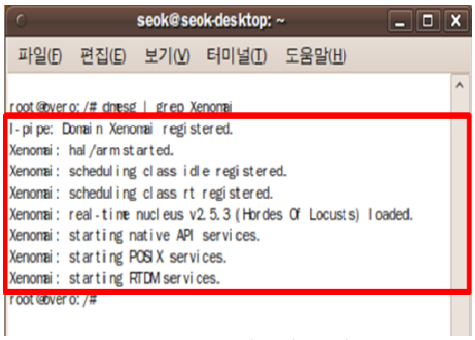
4. 구현

4.1. Xenomai 패치 및 커널 포팅

본 논문에서는 리눅스 커널 2.6.28 에 대하여 Xenomai 패치를 실행 하였다. 개발 환경은 우분투 9.10 이고, 교차 컴파일러는 arm-angstrom-linux-gnueabi-gcc 를 사용하였다. Xenomai 패치 후 커널 config 에서 Real-Time sub-system 을 선택하여 Xenomai 를 사용가능하게 한다. 또한 Pervasive real-time support in user-space 를 체크하여 사용자 공간에서도 실시간을 지원하도록 설정한다. 소프트웨어의 호환성 및 확장성을 극대화 하기 위해 POSIX API 및 실시간 전송을 위한 RTDM 을 선택하여 실험 하였다.

(그림 8)은 Xenomai 가 정상적으로 패치된 커널의 부팅메시지를 보여준다. Xenomai 가 작동을 하기 위해선 타겟 보드를 위한 Xenomai 공유 라이브러리가 필

요하다. 리눅스 시스템에서 생성한 Xenomai 공유 라이브러리를 타겟 보드의 루트파일시스템에 복사하여 Xenomai API 가 실행될 수 있도록 한다.



(그림 8) Xenomai 가 적용된 Kernel

4.2 Xenomai 쓰레드 구현

무인항공기의 멀티 쓰레드는 (표 1)과 같이 구현하였다. 쓰레드는 고정 우선 순위 스케줄링으로 구현하였으며, 필터 및 전송 쓰레드를 우선 순위를 높게 지정하였다. GPS 쓰레드가 AHRS 쓰레드 보다 상대적으로 주기가 낮기때문에 센서 획득 쓰레드의 동기화를 위해 RF 쓰레드의 주기를 GPS 와 같게 하였다. GPS, AHRS 쓰레드의 데이터는 메시지 큐(message queue)를 통하여 filter 쓰레드로 전송하였다.

쓰레드	우선순위	주기(Hz)	수행시간(㎲)
GPS	4	4	1648
AHRS	5	100	1785
Filter	1	50	961
RF	2	4	1054
PWM	3	50	1373

(표 1) Xenomai 쓰레드 내용

(표 1)을 기반으로 쓰레드의 스케줄링 가능성 및 응답성은 비율단조(RM : Rate-Monotonic) 스케줄링을 사용하였다. 비율단조 스케줄링은 프로세스에 우선순위의 변동이 없는, 정적인 스케줄링이며 상당히 효율적이며 널리 사용되고 있다.[3] 비율단조 스케줄링의 스케줄링 가능성 공식은 (그림 9)와 같다.

$$CPU\text{사용률}(U) = \sum_{i=1}^{n} \frac{C_i}{T_i} < n(\sqrt[n]{2} - 1)$$

(C: 수행시간, T: 주기, n: 쓰레드 개수)

(그림 9) 스케줄링 가능성 계산 공식

(그림 9)를 이용하여 CPU 사용률(Utilization Factor)은 0.303, 상한 CPU 사용률 0.743 를 얻었다. 0.303 < 0.743 이므로 Xenomai 를 이용한 멀티 쓰레드가 스케줄링이 가능하다는 결론을 얻었다.

5. 결론 및 향후 과제

본 논문에서는 실시간 운영체제인 Xenomai 기반의 임베디드 리눅스를 사용하여 무인항공기에 적합한 멀티 쓰레드의 개발 과정을 설명하고, 무인 항공기 쓰레드들의 응답속도 및 안정성을 보장할 수 있는 방안을 제시 하였다. 기존 단일 프로세스의 비 실시간성 시스템의 한계를 극복하고자 Xenomai 를 사용하여 실시간 성능을 보장하였고, 고정 우선 순위 스케줄링을 통하여 스케줄링이 가능함을 보여 시스템 성능을 향상 시켰다. 또한 Xenomai 의 RTDM 을 사용하여 획득한 데이터를 지상국으로 실시간 전송을 하였고, POSIX API 를 사용하여 범용성 및 확장성을 증대 하였다. 하지만 Xenomai 뿐만 아니라 비 상용 실시간 운영체제의 경우 타겟 보드의 선정이 어렵고, 타겟 보드에 따른 하드웨어 설정이 힘들다는 단점이 있다.

향후 고정 우선 순위 기반이 아닌 EFD(Eearly First Deadline)나 TMO 등 발전된 스케줄러를 이용하면 실시간 성능을 극대화 시킬 수 있을 것으로 기대한다. 또한 무인항공기가 복잡해지고 쓰레드가 늘어남에 따라 우선 순위 역전 현상이 발생 할 수 있으므로, 이에따른 뮤택스(Mutex) 및 세마포어를 사용하여 미연에 방지 할 수 있어야 한다.

참고 문헌

[1]안성진, “군용 항공전자시스템 운영체제로서의 리눅스 적용성 분석”, 국방과학연구소, 한국항공우주 학회, 2009

[2]최병욱 , “지능형 로봇을 위한 이중 커널 구조의 제어시스템 구현 및 실시간 제어 성능 분석”, 서울산업대, 로봇 시스템학회, 2008

[3]박한솔, “실시간 객체 모델을 이용한 내장형 무인항공기 제어시스템의설계 및 구현”, 건국대, 한국정보과학회, 2006

[4]이기영, ”소형 무인기용 센서통합형 비행제어 컴퓨터 개발”, 국방기술품질원, 한국항공우주학회, 2006

[5]배종윤, “실시간 UML 을 이용한 회전익 UAV 제어 소프트웨어 설계 및 구현”, 건국대, 한국정보과학회, 2009

[6]J.Kiszka, “The Real-Time Driver Model and First Applications”, IEEE 2004

[7]XENOMAI Home Page / www.xenomai.org

[8]XENOMAI, “Real-Time Linux and Xenomai system.pdf”

[9]Real-Time Technology for Embedded Linux : The MontaVista Advatage / http://www.mvista.com/real_time_linux.php

[10] KARIM YAGHMOUR, JON MASTERS “Building Embedded LINUX SYSTEMS”, p365 – 386