논문 2017-12-23

Zephyr 커널에서 고정 시간 동기식 IPC 구현

(Fixed Time Synchronous IPC in Zephyr Kernel)

정 주 영, 김 은 영, 신 동 하*
(Jooyoung Jung, Eunyoung Kim, Dongha Shin)

Abstract: Linux Foundation has announced a real-time kernel, called Zephyr, for IoT applications recently. Zephyr kernel provides synchronous and asynchronous IPC for data communication between threads. Synchronous IPC is useful for programming multi-threads that need to be executed synchronously, since the sender thread is blocked until the data is delivered to the receiver thread and the completion of data transfer can be known to two threads. In general, 'IPC execution time' is defined as the time duration between the sender thread sends data and the receiver thread receives the data sent. Especially, it is important that 'IPC execution time' in the synchronous IPC should be fixed in real-time kernel like Zephyr. However, we have found that the execution time of the synchronous IPC in Zephyr kernel increases in proportion to the number of threads executing in the kernel. In this paper, we propose a method to implement a fixed time synchronous IPC in Zephyr kernel using Direct Thread Switching(DTS) technique. Using the technique, the receiver thread executes directly after the sender thread sends a data during the remaining time slice of the sender thread and we can archive a fixed IPC execution time even when the number of threads executing in the kernel increases. In this paper, we implemented synchronous IPC using DTS in the Zephyr kernel and found the IPC execution time of the IPC is always 389 cycle that is relatively small and fixed.

Keywords : IoT, Real-time kernel, Synchronous IPC, Direct thread switching, Zephyr

Ⅰ. 서 론

사물인터넷 (IoT) [1] 기기들은 대부분 PC나 모바일 기기보다 메모리와 자원이 한정적이다. 이렇게 한정적인 자원을 갖는 다양한 기기들이 정보를 주고받는 환경에서도 안정된 동작 및 응답성이 보장되어야 하는 사물인터넷용 경량 기기를 위한 실시간 커널의 필요성 또한 커지고 있다. 실시간 커널 [2]이란 주어진 시간 내에 어떤 동작의 완료를 보장하는 커널을 말한다. 이에 2016년 2월, 리눅스 재단은 사물인터넷용 기기를 위한 오픈소스 실시간 커널인 Zephyr [3]를 발표했다.

Zephyr 커널은 임베디드 환경의 자원이 한정된 시스템을 위하여 설계된 small-footprint 커널이며 아파치 2.0 라이센스를 사용한다. Zephyr 커널은 높은 성능과 멀티 쓰레드 수행 환경 및 실시간성을 제공하며 다양한 커널 설정 옵션을 제공함으로써 사용자가 필요한 기능을 선택할 수 있고 커널의 크기를 조정할 수 있다는 특징을 갖는다 [3].

Zephyr 커널은 쓰레드 간의 데이터 통신을 위하여 동기식 (synchronous) 및 비동기식 (asynchronous) IPC를 제공한다. 동기식 IPC는 데이터가 수신자 쓰레드에게 전달될 때까지 송신자 쓰레드가 블록되지만 비동기식 IPC는 데이터가 수신자 쓰레드에게 전달되지 않아도 송신자 쓰레드가 블록되지 않고 수행을 계속한다. Zephyr 커널은 비동기식 IPC로 FIFO, LIFO, 스택 및 메시지 큐를 제공하고 동기식 혹은 비동기식 IPC로 메일박스와 파이프를 제공한다 [3].

일반적으로 'IPC 수행 시간'은 송신자 쓰레드가

*Corresponding Author (dshin@smu.ac.kr)
Received: Mar. 14 2017, Revised: May 10 2017, Accepted: May 30 2017.
J. Jung, E. Kim, D. Shin: Sangmyung University
※ 본 논문은 상명대학교 교내연구비의 지원을 받아 수행된 연구입니다.

ⓒ IEMEK J. Embed. Sys. Appl. 2017 Aug. 12(4) 205-212
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2017.12.4.205

데이터를 보내는 시점에서 수신자 쓰레드가 데이터를 받는 시점까지 걸린 시간으로 정의되는데, 특히 Zephyr와 같은 실시간 커널의 동기식 IPC의 경우에는 'IPC 수행 시간'이 고정적이어야 한다는 것이 매우 중요하다. 하지만 현재 Zephyr 커널의 동기식 IPC인 메일박스와 파이프의 경우에는 IPC 수행 시간이 커널 내에 수행되는 쓰레드의 개수에 영향을 받아 증가한다는 문제점을 본 연구에서 발견하였다. IPC 수행 시간이 증가하는 이유는 커널 내에 송/수신자 쓰레드와 우선순위가 같거나 높으면서 준비 상태에 있는 쓰레드들이 있을 경우, 송/수신자 쓰레드의 수행 사이에 스케줄러에 의해 선택된 다른 많은 쓰레드들이 수행되기 때문이다.

이를 해결하기 위해 본 연구에서는 Zephyr 커널에 DTS (Direct Thread Switching) 기법 [4, 5]을 사용하여 고정 시간 동기식 IPC를 구현하였다. DTS 기법은 송신자 쓰레드가 수신자 쓰레드에게 데이터를 송신한 후 기다리지 않고 송신자 쓰레드의 남은 타임 슬라이스 동안 수신자 쓰레드를 직접 수행시켜 데이터를 전달하여 커널 내에서 수행되는 쓰레드의 개수가 증가하더라도 IPC가 고정 시간에 수행되도록 하는 방법이다.

본 논문의 구성은 다음과 같다. 본 논문의 2장에서는 사전 연구로서 동기식 IPC, Zephyr 커널의 동기식 IPC와 스케줄링 방식 그리고 DTS 기법에 대해 설명한다. 3장에서는 본 논문에서 제안한 고정 시간 동기식 IPC의 구현 방법에 대해 설명하고 4장에서는 Zephyr 커널의 동기식 IPC와 본 연구에서 구현한 고정 시간 동기식 IPC의 수행 시간을 측정한 결과를 비교 분석한다. 마지막으로 5장에서는 본 논문의 결론을 기술한다.

II. 사전 연구

이 장에서는 동기식 IPC, Zephyr 커널의 동기식 IPC와 스케줄링 방식 및 DTS 기법에 대하여 설명한다.

1. 동기식 IPC (Inter Process Communication)

IPC [6, 7]는 크게 동기식 IPC와 비동기식 IPC로 나눌 수 있다. 비동기식 IPC에서 송신자 쓰레드는 데이터를 전송하고 수신자 쓰레드가 데이터를 받지 못하더라도 블록되지 않고 수행을 계속하는 반면, 동기식 IPC에서 송신자 쓰레드는 전송한 데이터가 수신자 쓰레드에게 전달될 때까지 블록된다 [8]. 따라서 동기식 IPC에서 송신자 쓰레드는 데이터의 완료 시점을 알 수 있어 두 쓰레드가 동기식으로 협조하여 수행해야 하는 작업에 유용하게 사용된다. 또한, 커널은 수신되지 못한 데이터를 저장하기 위한 큐가 필요하지 않기 때문에 메모리를 아낄 수 있다는 장점이 있다. 하지만 프로그래밍 오류에 의해 송/수신자 쓰레드가 무한정 블록될 수 있다는 문제점이 있으나 이는 일반적으로 timeout 기능을 사용하여 해결한다.

2. Zephyr 커널의 동기식 IPC

Zephyr 커널은 IPC로 FIFO, LIFO, 스택, 메시지 큐, 메일박스 그리고 파이프를 제공한다. 이 중 FIFO, LIFO, 스택 및 메시지 큐는 비동기식 IPC이고 메일박스와 파이프는 동기식 및 비동기식 IPC를 모두 제공한다. Zephyr 커널의 동기식 IPC인 메일박스와 파이프에 대한 설명은 다음과 같다.

- 메일박스: 메일박스는 다양한 크기의 메시지를 정해진 메시지 형식에 따라 동기식으로 전달한다. 메일박스는 송/수신자 쓰레드를 지정할 수 있으며 데이터 복사는 송신자 쓰레드 영역에서 수신자 쓰레드 영역으로 직접 복사되기 때문에 1회 일어난다.
- 파이프: 파이프는 두 쓰레드 사이에 바이트 스트림을 동기식으로 전달한다. 파이프에서 데이터 복사는 송신자 쓰레드 영역에서 수신자 쓰레드 영역으로 직접 복사되는 경우 1회 혹은 송신자 쓰레드 영역에서 파이프의 버퍼를 거쳐 수신자 쓰레드 영역으로 복사되는 경우 2회 일어난다.

Zephyr 커널의 동기식 IPC인 메일박스와 파이프에서는 송신자 쓰레드가 데이터를 송신한 후에 수신자 쓰레드가 아닌 스케줄러가 선택한 쓰레드가 수행되어 IPC 수행 시간이 커널 내에 준비 상태인 쓰레드의 개수에 영향을 받아 일정하지 않다는 문제점이 있다.

3. Zephyr 커널의 스케줄링

Zephyr 커널은 우선순위 기반 라운드 로빈 스케줄링 방식을 사용하여 레디 큐에서 다음에 수행할 쓰레드를 선택한다. 레디 큐는 현재 수행 중 혹은 준비 상태인 쓰레드가 연결되는 비트맵 기반 이중 원형 연결 리스트로, 각 우선순위에 해당하는 쓰레드가 연결되는 우선순위 큐와 각 우선순위 큐의 쓰

레드 유무를 표시하는 비트맵으로 구성된다. 이때 우선순위 큐는 쓰레드 우선순위 개수만큼 있으며 우선순위의 범위는 커널 설정 옵션으로 설정이 가능하며, 값이 작을수록 높은 우선순위를 갖는다. Zephyr 커널의 레디 큐는 구조체 _ready_q로 표현되며 구조체 _ready_q는 아래와 같은 세 개의 필드로 구성된다.

- cache: 레디 큐에서 다음에 수행할 쓰레드를 가리키며 스케줄링이 일어나면 스케줄러는 가장 먼저 이 필드를 검사하여 이 필드가 가리키는 쓰레드를 수행시킨다. 이 필드 값이 NULL일 경우 스케줄러는 레디 큐에서 다음에 수행할 쓰레드를 찾는다.
- prio_bmap[]: 각 우선순위 큐의 쓰레드 유무를 비트로 표시하는 비트맵 배열로, 한 개의 우선순위 비트맵은 32개의 우선순위를 나타낼 수 있으며 0번 인덱스의 LSB가 가장 높은 우선순위를 나타낸다.
- q[]: 각 우선순위에 대한 0개 이상의 쓰레드를 이중 원형 연결 리스트로 연결하기 위한 필드로, 각 큐에 있는 첫 번째와 마지막 쓰레드의 주소를 저장한다.

4. DTS (Direct Thread Switching)

마이크로 커널 [9] 기반 운영체제에서는 커널 서비스 프로그램들이 사용자 모드 주소공간에서 수행되기 때문에 IPC를 통해 통신한다. 마이크로 커널에서 IPC는 매우 자주 사용되기 때문에 그 성능이 매우 중요했으나 초기 마이크로 커널의 IPC는 성능이 좋지 않았다. 이에 1993년 독일의 컴퓨터 과학자 Liedtke는 IPC의 응답성을 높이기 위해 송신자 쓰레드 수행 후, 송신자 쓰레드의 남은 타임 슬라이스 동안 수신자 쓰레드를 수행시키는 direct process switch 기법 [10]을 고안했다. Direct process switch 기법은 L4 계열의 커널인 seL4, Fiasco.OC, NOVA 등에서 IPC에 실시간성을 제공하기 위해 사용되고 있다 [11, 12].

본 연구에서 제안한 DTS 기법 [4, 5]은 direct process switch 기법 [10]을 모놀리식 커널 구조의 실시간 커널인 Zephyr에 적용한 기법이다. 일반적으로 IPC 수행 시 송신자 쓰레드가 데이터를 송신하고 나면 커널은 스케줄러를 동작시켜 다음에 수행할 쓰레드를 찾고 그 쓰레드를 수행시킨다. 이때, 커널 내에 송/수신자 쓰레드와 우선순위가 같으면서 준비 상태에 있는 쓰레드가 있을 경우 스케줄러는 수신자 쓰레드가 아닌 다른 쓰레드를 선택할 수 있고 이 쓰레드의 수행이 끝나야 수신자 쓰레드가 수행될 수 있다. 따라서 커널 내에 준비 상태에 있는 쓰레드의 개수가 증가할 경우 IPC 수행 시간 또한 비례하여 증가한다.

그러나 DTS를 사용한 IPC는 송신자 쓰레드가 데이터를 송신한 후 송신자 쓰레드의 남은 타임 슬라이스 동안 스케줄러를 동작시키지 않고 바로 수신자 쓰레드를 수행시킴으로써 송/수신자 쓰레드의 수행 사이에 다른 쓰레드의 수행을 막아 IPC 수행 시간이 일정하다. 이때, 송신자 쓰레드의 남은 타임 슬라이스 동안 수신자 쓰레드를 수행시키기 때문에 스케줄러와 다른 쓰레드의 수행 순서에는 영향을 미치지 않는다. 또한 본 연구팀은 이전 연구에서 비동기식 IPC만을 제공하는 Zephyr 커널 v1.5.0의 나노커널에 DTS를 사용한 효율적인 동기식 IPC를 구현하여 Zephyr 나노커널의 비동기식 IPC인 나노커널 FIFO와 수행 시간을 비교했을 때, 더 짧은 시간 내에 수행됨을 확인한 바 있다 [13].

Ⅲ. 구 현

이 장에서는 본 연구에서 제안한 DTS를 사용한 IPC (IPC-DTS)의 구현 방법에 대해서 설명한다. 본 연구에서 구현한 IPC는 주어진 두 쓰레드 사이에 32 bit로 고정된 크기를 갖는 데이터를 동기식으로 전달하며, IPC가 고정 시간에 수행되도록 하기 위하여 사전연구에서 설명한 DTS 기법을 사용하였다. 다음은 Zephyr 커널에 고정 시간 동기식 IPC를 구현하기 위해 Zephyr 커널의 TCS (Thread Control Structure)인 구조체 k_thread를 수정한 내용과 IPC 송신 함수 및 수신 함수 그리고 DTS 수행 함수에 대한 설명이다.

1. TCS (Thread Control Structure) 수정

Zephyr 커널에서 하나의 쓰레드를 표현하는 TCS는 구조체 k_thread이며 쓰레드의 정보 (상태, 우선순위, 타임아웃 등)을 저장한다. 구조체 k_thread는 커널 내에 생성된 쓰레드 개수만큼 있으며 레디 큐의 구성요소로 사용된다. 또한 어떤 쓰레드의 구조체 k_thread의 주소는 그 쓰레드의 tid (thread id)와 같다. 본 연구에서는 고정 시간 동기식 IPC를 위해 구조체 k_thread에 다음과 같은 세 개의 필드를 추가하였다.

- message: 송/수신자 쓰레드 사이에서 IPC를 통해 주고 받을 데이터가 저장되는 필드이다. unsigned int 형의 32 bit 크기를 갖는 데이터가 저장될 수 있다.
- rbtid (receiver blocked by tid): 필드 rbtid는 수신자 쓰레드가 데이터 수신 요청을 했으나 수신할 데이터가 없을 경우, 수신자 쓰레드가 블록되면서 어떤 쓰레드로 부터 데이터를 기다리고 있는지를 표시하기 위한 필드이다. 송신자 쓰레드가 데이터를 송신하기 전에 수신자 쓰레드가 데이터 수신을 요청한 경우, 수신자 쓰레드가 블록되면서 자신의 TCS의 이 필드에 송신자 쓰레드의 tid를 저장한다. 후에 송신자 쓰레드가 데이터를 송신할 때, 송신자 쓰레드는 수신자 쓰레드의 필드 rbtid 값과 자신의 tid 값을 비교하여 이전에 데이터 수신 요청이 있었는지를 확인할 수 있다.
- sbtid (sender blocked by tid): 필드 sbtid는 송신자 쓰레드가 데이터 송신 요청을 했으나 데이터를 수신할 수신자 쓰레드가 없을 경우, 송신자 쓰레드가 블록되면서 어떤 쓰레드에게 데이터를 송신했는지를 표시하기 위한 필드이다. 수신자 쓰레드가 데이터 수신 요청을 하기 전에 송신자 쓰레드가 데이터를 송신한 경우, 송신자 쓰레드가 블록되면서 자신의 TCS의 이 필드에 수신자 쓰레드의 tid를 저장한다. 후에 수신자 쓰레드가 데이터 수신해 갈 때, 수신자 쓰레드는 송신자 쓰레드의 필드 sbtid 값과 자신의 tid 값을 비교하여 이전에 데이터 송신이 있었는지를 확인할 수 있다.

2. IPC 송신 함수

IPC 송신 함수는 현재 쓰레드 (송신자 쓰레드)가 다른 쓰레드 (수신자 쓰레드)에게 데이터를 송신하기 위해 사용된다. 그림 1은 IPC 송신 함수의 의사코드이다.

그림 1의 의사코드의 1번 줄에서 IPC 송신 함수는 인수로 수신자 쓰레드의 tid와 송신할 데이터를 전달받고 3번 줄에서 인수로 전달받은 수신자 쓰레드의 TCS의 필드 rbtid에 저장된 송신자 쓰레드의 tid와 현재 쓰레드의 tid를 비교하여 이전에 수신자 쓰레드가 현재 쓰레드에게 데이터를 요청한 적이 있는지 검사한다.

만약 수신자 쓰레드가 이전에 IPC 수신 함수를 호출하여 현재 쓰레드에게 데이터를 요청하고 블록되어 있을 경우, 4-8번 줄의 코드를 수행하여 수신자 쓰레드의 TCS에 데이터를 복사하고 수신자 쓰레드를 레디 큐에 추가한 후, 자신을 우선순위 큐의 맨 뒤로 옮긴다. 그 후, 본 장의 4절에서 설명하는 DTS 수행 함수를 호출하여 바로 수신자 쓰레드를 수행시켜 데이터 전달이 완료되도록 한다.

```
1  ipc_send_dts(receiver's tid, msg)
2  {
3    if (receiver's tcs.rbtid == my tid) {
4      receiver's tcs.message = msg;
5      receiver's tcs.rbtid = NULL;
6      add receiver to ready queue;
7      move me to end of priority queue;
8      call DTS_function;
9    } else {
10     my tcs.sbtid = receiver's tid;
11     my tcs.message = msg;
12     remove me from ready queue;
13     call scheduler;
14   }
15 }
```

그림 1. IPC 송신 함수의 의사코드
Fig. 1 Pseudo code for IPC send function

```
1  ipc_recv(sender's tid, *msg)
2  {
3    if (sender's tcs.sbtid == my tid) {
4      *msg = sender's tcs.message;
5      sender's tcs.sbtid = NULL;
6      add sender to ready queue;
7    } else {
8      my tcs.rbtid = sender's tid;
9      remove me from ready queue;
10     call scheduler;
11     *msg = my tcs.message;
12   }
13 }
```

그림 2. IPC 수신 함수의 의사코드
Fig. 2 Pseudo code for IPC receive function

만약 수신자 쓰레드가 이전에 현재 쓰레드에게 데이터 요청을 한 적이 없다면, 현재 쓰레드는 10-13번 줄의 코드를 수행하여 자신의 TCS에 수신자 쓰레드의 tid와 데이터를 저장한 후에 자신을 레디 큐에서 제거하여 블록시킨 후, 스케줄러를 호출한다.

3. IPC 수신 함수

IPC 수신 함수는 현재 쓰레드 (수신자 쓰레드)가 다른 쓰레드 (송신자 쓰레드)로 부터 데이터를 수신하기 위해 사용된다. 위 그림 2는 IPC 수신 함수의 의사코드이다.

그림 2의 의사코드의 1번 줄에서 IPC 수신 함수는 인수로 송신자 쓰레드의 tid와 수신한 데이터를 저장할 메모리의 주소를 전달받고 3번 줄에서 인수로 전달받은 송신자 쓰레드의 TCS의 필드 sbtid에 저장된 수신자 쓰레드의 tid와 현재 쓰레드의 tid를 비교하여 이전에 송신자 쓰레드가 현재 쓰레드에게 데이터를 송신한 적이 있는지 검사한다.

만약 송신자 쓰레드가 이전에 IPC 수신 함수를 호출하여 현재 쓰레드에게 데이터를 송신하고 블록되어 있을 경우, 4-6번 줄의 코드를 수행하여 송신자 쓰레드의 TCS에 저장된 데이터를 복사하고 송신자 쓰레드를 레디 큐에 추가한다.

송신자 쓰레드가 데이터를 송신한 적이 없다면, 현재 쓰레드는 8-10번 줄의 코드를 수행하여 현재 쓰레드의 TCS에 송신자 쓰레드의 tid를 저장하고 자신을 레디 큐에서 제거하여 블록시킨 후, 스케줄러를 호출한다. 후에 송신자 쓰레드가 IPC 송신 함수를 호출하여 수신자 쓰레드가 수행을 재개하면, 11번 줄의 코드를 수행하여 자신의 TCS에서 데이터를 메모리로 복사한다.

4. DTS 수행 함수

이 함수는 IPC 송신 함수에서 호출되며 수신자 쓰레드를 우선순위에 관계없이 송신자 쓰레드의 남은 타임 슬라이스 동안 바로 수행시킨다. Zephyr 커널은 스케줄링 시, 레디 큐 구조체 _ready_q의 필드 cache가 가리키는 쓰레드를 수행시키기 때문에 이 필드에 수신자 쓰레드를 저장하고 스케줄러를 호출하면 수신자 쓰레드를 바로 수행시킬 수 있다.

Ⅳ. 성능 측정

이 장에서는 Zephyr 커널의 동기식 IPC인 메일박스와 파이프와 본 연구에서 구현한 IPC의 수행 시간을 측정하여 비교 분석한다. 측정을 위해 Zephyr v1.6.0 [3]을 ARM Cortex-M4 프로세서 [14, 15]가 탑재된 NXP의 FRDM-K64F [16]에 수행시켰으며, 본 연구실에서 개발한 성능 측정 도구인 pcounter 프로그램 [17]을 사용하였다. 다음은 측정 방법과 측정 도구 그리고 측정 결과에 대한 설명이다.

1. 측정 방법

본 논문에서 측정한 Zephyr 커널의 메일박스와 파이프 그리고 본 연구에서 구현한 동기식 IPC인 IPC-DTS의 수행 시간의 측정 범위는 송신자 쓰레드가 IPC 송신 함수를 호출한 순간부터 수신자 쓰레드가 IPC 수신 함수를 반환한 순간까지이다. 일반적으로 커널에서는 여러 개의 쓰레드가 동시에 수행되기 때문에 송/수신자 쓰레드와 n개의 기타 쓰레드를 모두 같은 우선순위로 생성 및 수행시켰다. 송/수신자 쓰레드는 각각 unsigned int형의 4 byte 크기를 갖는 데이터를 IPC를 통해 송/수신하는 동작을 반복하여 수행하며, 기타 쓰레드는 약 5,000 사이클 동안 busy wait을 수행하고 프로세서를 다음 쓰레드에게 넘기는 작업을 반복한다.

2. 측정 도구

본 논문에서는 IPC 수행 시간을 측정하기 위해 본 연구실에서 개발한 pcounter 프로그램 [17]을 사용하였다. Pcounter 프로그램 [17]은 ARM Cortex-M 프로세서 [14, 15]가 제공하는 DWT (Data Watchpoint and Trace) 유닛의 DWT_CYCCNT (DWT Current PC Sampler Cycle Count) 레지스터 [18]를 읽어 프로그램의 수행 시간을 클럭 사이클 단위로 측정하는 도구이다. DWT_CYCCNT 레지스터 [18]는 프로세서의 클럭 사이클을 세며 수행 시간을 측정하기 위한 레지스터이다. Pcounter 프로그램은 측정하고자 하는 프로그램의 시작과 끝에서 이 레지스터 값을 읽고 두 값의 차에서 레지스터를 읽는데 걸린 오버헤드를 뺀 값을 계산하여 프로그램을 수행하는데 소요되는 클럭 사이클을 측정한다.

3. 측정결과

표 1과 그림 3은 Zephyr 커널의 메일박스, 파이프 및 본 연구에서 구현한 IPC-DTS의 수행 시간을 정리 및 비교한 표와 그래프이다.

표 1에서 쓰레드 개수 (n)는 커널 내에서 송/수신자 쓰레드 외의 기타 쓰레드 개수를 의미한다. 측정값은 100,000회 측정한 결과의 최솟값이며 이 값이 클수록 IPC 수행에 시간이 오래 걸림을 의미한다. 측정값으로 최솟값을 사용한 이유는 IPC 송신 함수를 시작 직전과 IPC 수신 함수의 반환 직후에 pcounter 프로그램을 사용하여 DWT_CYCCNT 레지스터를 읽는 도중에 타이머 인터럽트가 발생하여 다른 쓰레드로 수행이 넘어가면 측정 시작 및 끝 지점이 IPC 송신 함수 시작 및 IPC 수신 함수 반환 지점이 아닌 다른 쓰레드의 수행 지점이 될

표 1. IPC 수행 시간 측정 결과
Table 1. The result of IPC execution time measurement

No. of threads (n)	Mailbox	Pipe	IPC-DTS
0	951	1,229	394
10	54,261	54,559	389
20	107,561	107,909	389
30	160,861	161,259	389
40	214,161	214,609	389
50	267,461	267,959	389
60	320,761	321,309	389
70	374,061	374,659	389
80	427,361	428,009	389
90	480,661	481,359	389
100	533,961	534,709	389

clock cycles

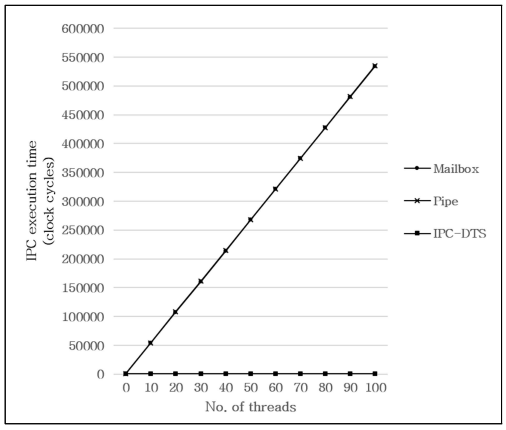

그림 3. 메일박스, 파이프 및 IPC-DTS의 IPC 수행 시간 비교
Fig. 3 Comparison of IPC execution time in mailbox, pipe and IPC-DTS

수 있어 평균값이나 최대값을 사용할 경우 측정값이 정확하지 않을 수 있기 때문이다.

표 1의 측정 결과에 의하면 송/수신자 쓰레드 외의 기타 쓰레드 개수가 0일 때에도 메일박스와 파이프의 수행 시간이 IPC-DTS에 비해 더 오래 걸리는 것을 확인할 수 있다. 본 연구에서는 그 이유를 메일박스와 파이프는 동기식 뿐만 아니라 비동기식으로도 동작할 수 있기 때문에 송/수신자 쓰레드가 블록될 경우 연결되는 웨이트 큐가 있어 이 웨이트 큐에 쓰레드를 추가하고 삭제하는 작업을 수행하는데 소요되는 오버헤드가 있기 때문으로 분석하였다. 그러나 IPC-DTS에서는 송/수신자 쓰레드가 블록될 때에도 웨이트 큐를 사용하지 않기 때문에 IPC 수행 시간이 메일박스와 파이프에 비해 짧게 걸린다.

또한 메일박스와 파이프에서는 커널 내의 쓰레드 수가 증가함에 따라 IPC 수행 시간이 비례하여 크게 증가하지만 IPC-DTS의 경우 389 사이클로 고정적임을 확인할 수 있다. 이는 그림 3의 그래프에서도 확인할 수 있다. 메일박스와 파이프에서 커널 내의 쓰레드 수가 증가할수록 IPC 수행 시간이 증가하는 이유는 송/수신자 쓰레드의 수행 사이에 스케줄러에 의해 수행되는 다른 쓰레드 수가 증가하기 때문이다. 반면에 IPC-DTS에서는 DTS 기법을 사용했기 때문에 송/수신자 쓰레드 사이에 다른 쓰레드가 수행되지 않아 IPC 수행 시간이 고정적이다.

V. 결 론

실시간 커널은 정해진 마감시간 내에 동작해야 하기 때문에 고정 시간 동기식 IPC를 제공해야 한다. 하지만 현재 Zephyr 커널의 동기식 IPC인 메일박스와 파이프에서는 IPC 수행 시간이 커널 내에 수행되는 쓰레드의 개수에 영향을 받아 그 수가 증가할수록 IPC 수행 시간 또한 증가한다는 문제점이 있다. 이를 해결하기 위하여 본 연구에서는 Zephyr v1.6.0에 DTS 기법을 사용하여 수행 시간이 커널 내에 수행되는 쓰레드의 개수에 영향을 받지 않고 일정한 고정 시간 동기식 IPC를 구현하였다. 본 논문에서 구현한 고정 시간 동기식 IPC에서는 송/수신자 쓰레드 수행 사이에 다른 쓰레드가 수행되지 않기 때문에 커널 내에 수행되는 쓰레드 개수가 증가해도 IPC 수행 시간이 고정적이며 이를 측정을 통해 확인하였다.

References

[1] E. Baccelli, O. Hahm, M. Wahlisch, M. Gunes, T. C. Schmidt, “RIOT: One OS to Rule Them All in the IoT,” INRIA Research Report No. RR-8176, 2012.

[2] P. A. Laplante, S. J. Ovaska, Real-Time Systems Design and Analysis: Tools for the Practitioner, 4th Edition, John Wiley & Sons, Inc. 2012.

[3] Zephyr Project Documentation, 2016.

https://www.zephyrproject.org/doc/1.6.0

[4] E. Kim, D. Shin, “IPC Implementation and Analysis Using Direct Thread Switching on Operating System Kernel,” Proceedings of the Conference of Institute of Korean Engineering of Korea, Vol. 1, No. 1, pp. 398-401, 2015 (in Korean).

[5] D. Shin, S. Son, E. Kim, “An Implementation of IPC Using Direct Thread Switching,” International Journal Of Engineering And Computer Science, Vol. 5, No, 1, pp. 15454-15457, 2016.

[6] A. Silberchatz, P. B. Galvin, G. Gagne, Operating System Concepts, Chapter 3 Processes, 9th Edition, John Wiley & Soons, Inc., 2012.

[7] W. Stallings, Operating Systems Internals and Design Principles, Chapter 5 Concurrency: Mutual Exclusion and Synchronization, 8th Edition, Pearson Education, Inc., 2014.

[8] T. Jaeger, J. E. Tidswell, A. Gefflaut, Y. Park, K. Elphinstone, J. Liedtke, “Synchronous IPC Over Transparent Monitors,” Proceedings of the 9th ACM SIGOPS European Workshop, pp. 189-194, 2000.

[9] J. Liedtke, “On Micro-kernel Construction,” Proceedings of the 15th ACM Symposium on Operating Systems Principles, Vol. 29, No. 5, pp. 237-250, 1995.

[10] J. Lidetke, “Improving IPC by Kernel Design,” Proceeding of the 14th ACM Symposium on Operating Systems Principles, Vol. 27, No. 5, pp. 175-188, 1993.

[11] K. Elphinstone, G. Heiser, “From L3 to seL4: What Have We Learnt in 20 Years of L4 Microkernel?,” Proceeding of the 24th ACM Symposium on Operating Systems Principles, pp. 133-150, 2013.

[12] A. Lyons, G. Heiser, “It's Time: OS Mechanisms for Enforcing Asymmetric Temporal Integrity,” ArXiv:1606.00111, 2016.

[13] J. Jung, E. Kim, J. Lim, D. Shin, “Efficient Synchronous IPC in Zephyr Nanokernel,” The Conference of Institute of Korean Engineering of Korea 2016, Vol. 1, No.1, pp. 442-445, 2016 (in Korean).

[14] ARM, ARM Cortex-M4 Processor Technical Reference Manual, Revision: r0p1, ARM 100166_0001_00_en, 2015.

[15] J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3rd Edition, Elsevier Inc., 2014.

[16] NXP, FRDM-K64F Freedom Module User's Guide Rev 1, FRDMK64FUG, 2016.

[17] E. Kim, J. Jung, J. Kim, D. Shin, “Improving Fiber Scheduling in Zephyr Kernel,” The Conference of Institute of Korean Engineering of Korea, Vol. 1, No.1, pp. 438-441, 2016 (in Korean).

[18] ARM Limited, ARMv7-M Architecture Reference Manual, ARM DDI 0403E.b, Part C Debug Architecture, 2014.

Jooyoung Jung (정 주 영)

She received B.E. degree in Computer Science from Sangmyung University in 2015. She is currently a M.S. student of Department of Computer Science at Graduate School of Sangmyung University. Her research interests include real-time kernels, inter thread communication, inter processor communication and thread scheduling.
Email: 201538007@sangmyung.kr

Eunyoung Kim (김 은 영)

She received B.E. degree in Computer Science from Sangmyung University in 2016. She is currently a M.S. student of Department of Computer Science at Graduate School of Sangmyung University. Her research interests include real-time kernels, inter thread communication and inter processor communication.
Email: 201631060@sangmyung.kr

Dongha Shin (신 동 하)

He received the B.S. degree in Computer Engineering from Kyung-pook National University in 1980, the M.S. degree in Computer Engineering from Seoul National University in 1982 and the Ph.D. degree in Computer Science from University of South Carolina in 1994. During 1982-1996, he stayed in ETRI as a technical staff to study expert systems, word processing systems, file systems and language processing systems. During 1997-current, he is a professor of Computer Science Department in Sangmyung University. His research interests include real-time kernels, inter thread communication, inter processor communication, embedded computing and compiler implementation.
Email: dshin@smu.ac.kr