네트워크 상의 종단 호스트에서의 실시간 데이터 처리 기법

김경산, 김성조[1]
중앙대학교 컴퓨터 공학부
{mountain, sjkim}@konan.cse.cau.ac.kr

A Mechanism of Real-time Data Processing at End-Host on Network

Kyung San Kim Sung Jo Kim
School of Computer Science and Engineering, Chung-Ang University

요 약

최근 네트워크의 전송 대역이 넓어지고 네트워크 전송 기술이 발전함에 따라 네트워크 통신의 실시간성을 만족하기 위한 요구사항 중 종단 호스트가 중요한 병목 지점으로 부각되고 있다. 종단 호스트에서의 처리 과정에서 발생할 수 있는 병목 현상은 사용자 공간과 커널 공간의 전환, 커널의 네트워크 모듈 처리, 네트워크 모듈에서 디바이스 드라이버로 데이터의 전달 과정에서 발생할 수 있다. 본 논문에서는 이 중 첫 번째 병목지점에 대한 개선 방안으로서 커널과 사용자 공간 사이에 복사를 없애는 기법과, 세 번째 병목지점에 대한 개선 방안으로서 새로운 패킷 처리 구조를 제안 한다. 이를 통해 종단 호스트에서 데이터 처리 시간을 단축함으로써 실시간성을 지원하고, 부가적으로 시스템 리소스와 네트워크 대역사용을 줄일 수 있다. 또한 시스템의 자원이 충분치 않은 임베디드 시스템에 적용 시 더 큰 효과를 얻을 수 있을 것으로 예상된다.

1. 서 론

실시간 시스템은 애플리케이션 작업이 올바르게 수행할 뿐 아니라 정해진 시간 내에 작업을 완료할 것을 요구하는 특징을 가지며 자동차, 항공기, 네트워크 장비, 센서 네트워크 그리고 모바일 장비 등 다양한 응용 분야에서 사용되고 있다. 최근 네트워크 전송 대역이 급속도로 넓어지고, 효율적인 라우팅 기법 및 패킷 스케줄링 알고리즘 대한 연구가 활발히 진행되고 있으며, IPv6와 같은 차세대 프로토콜에서도 라우터에서 패킷 처리를 단순화하기 위한 철학을 프로토콜 설계에 반영하고 있다[1,2]. 네트워크를 통한 단말 호스트간의 I/O를 수행하는 네트워크 통신의 구조상, 네트워크 중간 경로에서의 연구와 더불어 통신이 수행되는 종단 호스트 내에서의 데이터 처리기술이 통신 전체의 실시간성 만족에 중요한 요소가 된다[3]. 종단 호스트에서 네트워크 통신을 위해 수행되는 과정은 1)사용자 공간과 커널 공간 사이에서의 전환 및 데이터 복사, 2)커널 내의 네트워크 모듈 경유, 3)네트워크 모듈과 네트워크 인터페이스 사이에서의 처리와 같이 세가지 과정으로 설명할 수 있다[그림 1]. 때문에 종단 호스트 내에서 실시간 성을 지원하기 위해서는 위의 세가지 부분에 대한 연구가 진행되어야 한다. 본 논문의 2장에서는 1)에 대한 개선 방안으로서 커널과 사용자 공간 사이에 데이터 복사를 제거하는 기법을 설계 및 구현하였고 이에대해 설명한다. 3장에서는 3)에 대한 개선방안으로서 커널과 네트워크 인터페이스 사이에 우선순위를 고려한 패킷 처리 기법에 대해 제안한다. 본 연구실에서는 2)에 대한 커널 내의 네트워크 모듈에서 실시간성을 지원하기 위한 방안이 연구 된 바 있다[4].

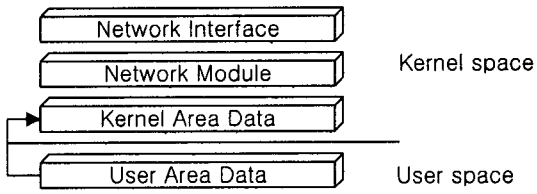

그림 1.네트워크 통신에서 수행되는 세가지 단계

2. 제로카피(Zero-copy)

가상메모리 관리 시스템을 지원하는 리눅스 커널에서는 시스템의 보호를 위해 프로세스의 선형 주소공간이 고정된 크기의 커널 공간과 사용자 공간으로 구분되어있다. 두 주소 공간 사이에는 자유롭게 읽기 쓰기 연산을 할 수 없고 copy_to_user()나, copy_from_user() 같은 특수한 커널 API를 통해서 두 공간사이에 복사를 수행해 데이터를 참조 할 수 있다. 네트워크 I/O가 수행될 때 사용자 영역의 애플리케이션에서 송신하는 패킷 데이터는 커널의 네트워크 모듈을 지나서 네트워크로 전송 되게 되며, 네트워크에서 수신하는 패킷도 네트워크 모듈을 지나 사용자 영역의 애플리케이션에 도착하게 된다. 이러한 송/수신의 과정 모두에서 커널과 사용자 영역의 문맥전환이 일어나게 되고,

[1] 본 연구는 정보통신부 ITRC 프로그램의 지원을 받아 중앙대학교 HNRC(Home Network Research Center)에서 수행되었음.

이에 따라 송/수신되는 패킷은 커널 API를 통하여 두 공간 사이에서 복사되어야 한다[5]. 제로카피는 커널과 사용자 영역간에 데이터를 주고 받을 때 두 영역 사이에 보호를 위해 요구되는 데이터 복사를 없애는 기법 을 의미한다. 즉 커널과 사용자 영역 사이에서 복사 없이 일반 애플리케이션 내에서처럼 데이터의 포인터만으로 데이터를 조작할 수 있는 방법이다.
본 논문에서는 하나의 물리 메모리 페이지프레임에 커널 가상 주소 공간과 사용자 가상 주소 공간을 각각 매핑 시키는 방법으로 제로카피를 구현하였다

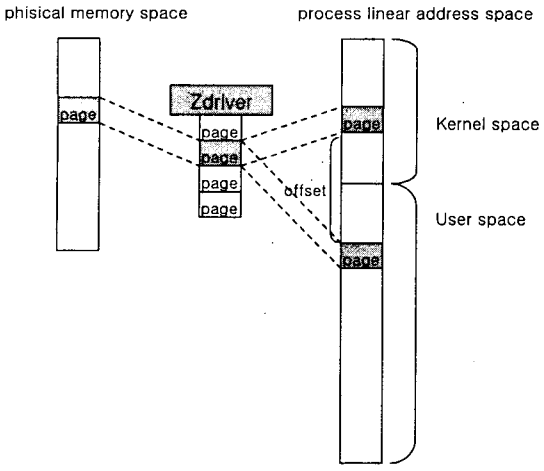

그림 2.메모리 매핑 인터페이스를 통한 제로카피

zdrivr는 제로카피를 위해 사용될 물리 메모리 영역의 할당 및 해제, 물리 주소 공간과 커널 주소공간 및 사용자 주소 공간의 매핑 시키는 기능을 제공하는 디바이스 드라이버 이다. zdriver는 커널 공간에서 동작하며 사용자 공간에서는 파일 공통 연산에서 제공되는 인터페이스(struct file_operation)를 통해 접근한다. zdriver는 공유영역으로 사용할 메모리 공간의 할당 시 연속된 물리메모리 페이지들에 대한 커널 주소를 할당 받도록 하였다. 이러한 이유는 연속된 물리 메모리 영역은 커널 주소공간 오프셋(0xC0000000)만을 차감하는 것만으로 물리주소를 확보할 수 있기 때문에 메모리 공간 접근시 커널 내에서 빠른 주소계산을 통해 물리 메모리에 접근할 수 있다.
할당 받은 연속된 물리 메모리 영역을 공유 공간으로 사용하기 위해서는 커널 주소 공간과 사용자 주소 공간으로 물리페이지를 매핑 해야 한다. 디바이스 드라이버는 커널 공간에서 동작하기 때문에 zdriver가 연속된 물리 메모리 영역을 할당 받을 때 이미 커널 주소 공간에 매핑 된 주소를 리턴 한다. 때문에 커널 공간에 대한 별도의 매핑 작업은 필요 없다. 사용자 주소 공간으로의 매핑을 위해 애플리케이션은 mmap 시스템 콜을 호출한다. mmap 시스템 콜에 대응되는 zdriver의 mmap 함수 내부에서는 remap_page_range 커널 함수를 사용하여 사용자 영역 주소 공간에 할당 받은 물리 메모리 페이지들을 매핑 시킨다. 사용자 공간에 대한 매핑이 완료되면 사용자 애플리케이션은 zdriver가 커널,사용자 두 공간 사이에 매핑된 주소의 offset을 알 수 있도록 ioctl 시스템 콜을 사용하여 zdriver에 사용자 공간에 매핑된 주소를 알린다. 이 offset은 두 공간 사이에서 값을 읽어오기 위한 포인터 위치를 계산하기 위한 기준주소로 사용된다.

zdriver가 할당한 공간의 사용상태를 관리하기 위한 매커니즘이 추가적으로 필요한데 본 구현에서는 리눅스 커널에서 사용하고 있는 비트맵 관리 기법을 사용하였다[그림 3][6].

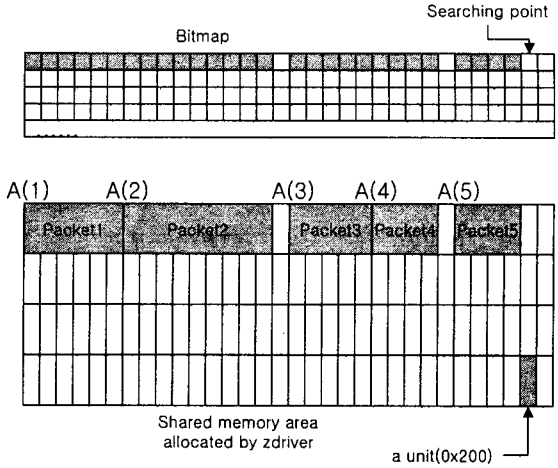

그림 1.비트맵을 공유 메모리 공간의 관리

zdriver를 이용한 zero-copy 시나리오에 대해 설명한다.
패킷이 종단 호스트에 도착하면 커널은 zdriver에서 할당받은 공유 메모리 공간을 bitmap 탐색을 통해 가용공간을 찾은 후 패킷을 저장한다. 이후 애플리케이션은 read 시스템 콜을 사용하여 읽기를 시도한다. 이에 대응되는 zdriver의 read 함수에서는 현재 읽어야 하는 패킷에 대한 사용자 공간에 매핑 되어 있는 주소 계산하여 리턴한다[그림 4]. 이를 통해 사용자 공간에서는 복사 없이 이 공간에서 값을 읽을 수 있다.

```
userAddr = baseAddr -  offset + A(n)

userAddr : 사용자 공간에서 데이터를 읽을 주소
baseAddr : 커널 공간에 매핑된 시작 주소
offset    : 매핑된 사용자 공간과 커널공간의 주소차
A(n)      : n 번째 패킷의 읽기 위한 시작 주소
```

그림 4.데이터를 읽을 사용자 주소공간 계산식

2장에서 설명한 zdriver 디바이스 드라이버를 사용하여 커널과 사용자 공간에서는 데이터의 복사 없이 전송될 데이터들을 공유할 수 있다.

3. 우선순위기반 멀티레벨 패킷 큐

멀티 태스킹 OS에서 프로세스들은 제한된 시스템 자원인 CPU에 대하여 경쟁한다. 리눅스 커널 2.6 에서는 선점 형 커널을 지원하고 시스템의 응답 성을 개선함으로써 프로세스가 요구하는 실시간성 처리를 강화하였다. 하지만 커널은 프로세스 단위로 스케줄링 하기 때문에 커널 에서 이루어지는 패킷 수준의 정밀한 제어를 통한 실시간성의 지원은 기대 할 수 없다. 리눅스 커널은 종단 호스트에 패킷이 도착하면 Top-Half 루틴이 패킷을 커널의 하나의 처리 패킷 큐에 복사한 후 우선 종료한다. 이후 CPU를 다시 할당 받았을 때 Bottom-Half 루틴이 패킷을 큐에서 꺼내어 상위 프로토콜 스택으로 전달하는 잔여작업을 수행하게 된다[5]. 이러한 단일 큐 구조에 기반한 패킷 처리는 패킷을 사용하는 프로세스의 우선순위가 높다 하더라도 패킷의 우선적인 처리를 보장할 수가 없는 단점을 가지고 있다.

본 논문에서 제안하는 패킷 처리 구조는 패킷을 저장하기 위한 큐를 프로세스의 우선순위 혹은, 프로세스 그룹의 우선순위에 기반한 멀티 패킷 큐 구조를 제안한다[그림 5]. 패킷 큐는 우선순위 순서로 연결된 리스트로 관리된다.

Top priority queue는 가장 우선순위가 높은 큐인 리스트의 헤드를 나타낸다. 이 포인터는 최고 우선순위를 지정한 프로세스의 생성과 종료에 따라 가리키는 큐를 수정하게 된다. 우선순위 큐가 하나도 없는 경우에는 Top priority queue는 우선순위 가지정되지 않은 일반 프로세스들의 패킷이 저장되는 큐인 No priority queue를 가리키게 된다.

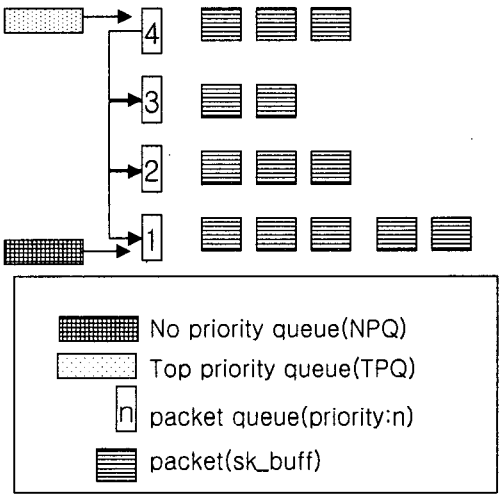

그림 5.우선순위 멀티 큐 구조

프로세스가 실행 시 자신만의 우선순위를 지정하여 패킷 큐를 생성 할 수 있는 시스템 콜을 제공한다. 멀티 패킷 큐 구조에서의 패킷 처리구조는 [그림 6]과 같다. 패킷이 호스트의 종단에 도착하면 Top-Half루틴인 인터럽트 핸들러가 수행된다. 핸들러는 패킷을 룩업 하여 목적 프로세스를 알아 낸 후 해시테이블에서 프로세스의 큐 포인터를 찾아 해당되는 큐에 패킷을 삽입한다. 자신의 우선순위 큐를 가지지 않은 프로세스인 경우 가장 낮은 순위의 입력 큐(NPQ)에 패킷을 삽입 하여 가장 낮은 순위로 처리되게 된다(그림 6의 1,2).

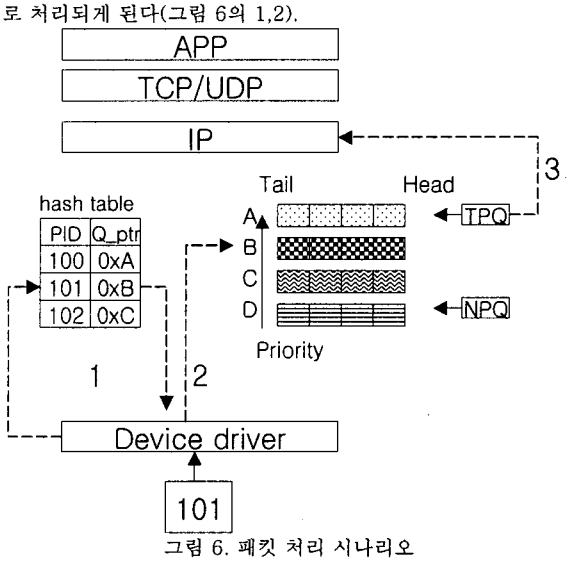

그림 6. 패킷 처리 시나리오

이후 Bottom-Half루틴은 TPQ에서 패킷을 꺼내서 상위 네트워크 모듈으로 전달한다. 패킷을 꺼낸 후 큐에 더 이상 패킷이 없는 경우 TPQ를 다음 우선순위의 큐를 가리키도록 수정한다(그림 6의 3). 안된 우선 순위 기반 멀티 큐를 사용할 경우 부가적으로 TCP 혼잡 회피 알고리즘에서 재전송 횟수를 줄이는 효과를 얻을 수 있다. ACK 패킷 역시 다른 패킷과 동일하게 단일 큐에서 FIFO 방식으로 처리되기 때문에 단말에 도착한 시점에는 시간제약이 유효했던 ACK 패킷이 실제 처리되는 TCP 계층에 전달되기까지 큐에서 대기하는 동안 허용된 시간을 초과하여 재전송을 요청하는 경우가 발생 할 수 있다. 제안된 구조를 사용하여 이러한 불필요한 재전송에 소모되는 대역 낭비 및 처리 지연을 줄일 수 있다. 멀티 패킷 큐 구조의 장점과 함께 유지시 발생하는 오버헤드에 대해 고려하여야 한다. 프로세스의 생성과 종료시 발생하는 큐 리스트의 갱신은 빈번한 연산이 아니므로 성능에 큰 영향을 미치지 않을 것이다. 그림 5의 과정은 네트워크 인터페이스에 도착하는 모든 패킷이 수행하는 과정이기 때문에 실시간성 지원과 네트워크 통신의 병목지점이 된다. 그림 5의 2와 3은 단일 큐 구조에서도 동일한 과정이나 그림 5의 1 과정인 패킷 삽입 큐를 검색하기 위한 해시 테이블 참조에 소모되는 오버헤드는 제안된 구조에서 추가된 오버헤드로 최적화 하기위한 연구가 추가적으로 필요하다.

4. 결론 및 향후 연구 과제

본 논문에서는 종단호스트에서 실시간 성을 지원하기 위하여 커널과 사용자 공간 사이에 복사에 소모되는 시간을 제거하는 제로카피 디바이스 드라이버를 구현하였고 이의 구조 및 활용에 대해 설명하였다. 또한 패킷 수준의 정밀한 제어를 통해 네트워크 I/O를 수행하는 실시간 성을 요구하는 프로세스에 패킷을 우선적으로 전달할 수 있는 우선순위 기반 멀티 큐 구조에 대해 제안하였다. 이 두 기법을 통하여 종단 단말에서의 실시간성 지원 및 부가적인 효과로서 시스템 리소스와 네트워크 대역의 절약효과가 기대되며, 시스템의 자원이 충분치 않은 임베디드 시스템에 이 두 기법을 적용시 더 큰 효과를 얻을 수 있을 것으로 예상된다. 향후 연구과제로서 제로카피 디바이스 드라이버를 커널의 네트워크 모듈에서 사용하는 메모리 공간으로서 연동하고, 커널에 멀티 패킷 큐 구조를 적용하여 성능 개선 측정에 대한 연구가 진행될 것이다. 또한 두 기법의 안정성을 위한 동기화 기법에 대한 지원 역시 연구될 것이다.

참고 문헌

[1] J. Borje, H. Lund and A. Wirkestrand, "Realtime Routers for Wireless Networks", Ericsson Review, No. 4, Ch1, 1999.

[2] Silvia Hagen, *IPv6 Essentials*, O`REILLY, No2, Ch2.3.2, 2002

[3]Delivery.acm.org, "http://portal.acm.org/ft_gateway.cfm?id=603774&type=html&coll=Portal&dl=GUIDE&CFID=42436492&CFTOKEN=53200265"

[4] 선동국, 김태웅, 김성조, "내장형 시스템의 원활한 멀티미디어 서비스 지원을 위한 커널 수준의 RTP", 한국 정보 과학회 2004년 추계 학술대회, 제10권 제6호, pp. 460-471, Dec. 2004

[5] 이귀영,*Add-on Linux Kernel*, No1, Ch5, 글로벌

[6] Uresh Vahalia, *Unix internals*, 2001, 흥릉과학 출판사