부록(PDF) 심화 학습

* 부록은 http://www.hanbit.co.kr/src/4467에서 PDF로만 제공합니다.

SECTION 01 컴퓨터의 사칙연산과 논리 게이트

SECTION 02 버스 종류와 캐시 구조

SECTION 03 인터럽트와 저장 장치의 계층 구조

SECTION 04 삽입정렬과 버킷정렬

SECTION 05 LAN 토폴로지와 오류 코드

SECTION 01

컴퓨터의 사칙연산과 논리 게이트

본문 연계: 3장

1. 헤르츠 사용

헤르츠는 주파수를 나타내는 단위다. 100Hz는 1초에 100번을 진동한다는 의미다. 따라서 헤르츠는 주기적으로 반복하는 모든 단위에 사용한다. 예를 들어 시계 초침은 1초에 한 번 움직이므로 1Hz다.

인간의 가청주파수는 20Hz에서 20KHz까지다. 여기서 20Hz는 1초에 소리 변화가 20번 나타나는 음으로, 인간이 들을 수 있는 가장 낮은 저음이다. 20KHz는 1초에 소리 변화가 2만 번 나타나는 음으로, 인간이 들을 수 있는 가장 높은 음이다. 인간은 들을 수 없지만 높은 주파수를 통신에도 사용한다. 예를 들어 여러분이 집에서 사용하는 무선공유기(무선 WiFi)는 2.4GHz대 전파를 사용하여 데이터를 전송하는 통신기기다.

전기에도 헤르츠를 사용한다. 우리나라는 220V에 60Hz 규격의 전기를 사용한다. 여기서 220V는 전압으로 전기 세기를 나타낸다. 일본, 미국, 캐나다는 110V를 사용하기 때문에 해당 국가에서 구매한 가전제품은 승압 변압기를 사용해야 한다. 그렇다면 60Hz는 무엇을 의미할까? 보통 가전제품에는 +극과 −극이 필요하며, 이렇게 +와 −가 결정되어 있는 전류를 직류라고 한다. 그런데 가정까지 배달되는 전기는 +극과 −극이 계속 교차하는 교류다. 가전제품은 어댑터로 이 교류를 직류로 바꾸어 사용한다. 교류에서 1초 동안 +극과 −극이 바뀌는 횟수를 나타낸 것이 Hz다. 다시 말해 우리나라 전기는 1초 동안 60번 +와 −가 바뀌는 교류라는 의미다.

중국은 220V에 50Hz 규격의 전기를 사용한다. 그래서 중국 내수용 제품을 우리나라에서 사용해도 전압이 똑같기 때문에 작동하는 데 이상이 없다. 그러나 교류 규격이 달라 가전제품이 쉽게 망가질 가능성이 있다. 따라서 중국 제품을 살 때는 프리볼트(free volt) 제품, 다시 말해 전 세계 어떤 전기를 써도 다 맞는 제품을 구매하는 것이 좋다.

2. 컴퓨터의 사칙연산

컴퓨터는 2진법을 사용하여 단순화함으로써 계산을 빠르게 한다고 설명했다. 덧셈(+), 뺄셈(−), 곱셈(×), 나눗셈(/)의 사칙연산도 마찬가지다.

덧셈 연산과 뺄셈 연산

컴퓨터는 덧셈은 할 수 있어도 뺄셈은 못한다. 그 대신 보수를 만들어 더함으로써 뺄셈을 한다. 예를 들어 7−4는 7+(−4)로 계산하는 것이다. 2의 보수에서 7은 (0111)이고 −4는 (1100)이므로 이를 더하면 (10011)이다. 여기서 맨 앞자리의 넘치는 값(오버플로)을 버리면 (0011)이 되고 계산 결과는 3이다.

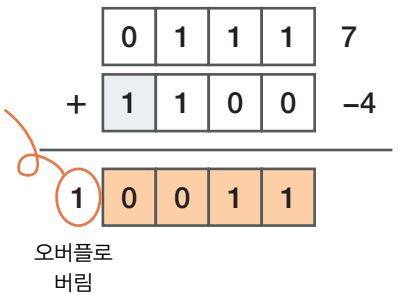

그림 1 컴퓨터의 뺄셈 연산

곱셈 연산

컴퓨터는 곱셈이나 나눗셈도 단순하게 처리한다. 곱셈은 여러 번 더하면 해결된다. 예를 들어 3×4는 3을 네 번 더하면 된다. 특히 2의 배수를 곱하거나 나눌 때는 자리이동(시프트, shift)을 이용하면 쉽게 처리할 수 있다. 예를 들어 12×2의 경우, 12의 전체 자릿수를 오른쪽으로 한 칸 이동하면 16이 된다. 또 12의 전체 자릿수를 두 칸 오른쪽으로 이동했다는 것은 12에 2^2을 곱한 것, 즉 12×4를 한 것이다. 다시 말해 2의 지수만큼 오른쪽으로 이동시키면 곱셈이 된다.

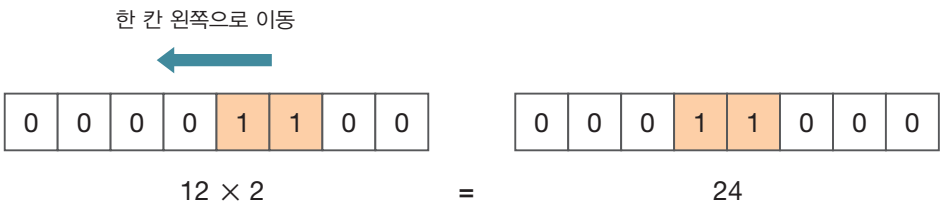

그림 2 컴퓨터의 곱셈 연산

나눗셈 연산

나눗셈은 2의 지수만큼 왼쪽으로 이동하면 된다. 예를 들어 12/4는 $12/2^2$이기 때문에 왼쪽으로 두 칸 이동시키면 되고, 결과 값은 3이 된다. 물론 2의 지수승이 아닐 경우 복잡한 전처리를 해야 하지만, 2의 배수는 자리이동을 이용하면 간단하게 곱셈과 나눗셈이 가능하다.

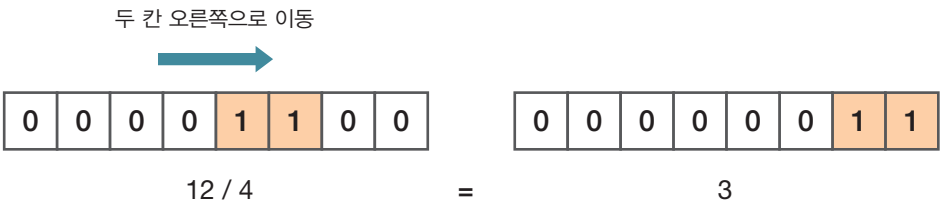

그림 3 나눗셈 연산

3. 정수와 실수 표현

앞서 정수와 실수를 배웠다. 정수와 실수는 비슷해 보이지만 특성이 다르다. 정수는 크기 문제가 있고, 실수는 정확도 문제가 있다. 예를 들어 가계부 프로그램을 짜는데, 2바이트 크기의 정수가 있다고 가정해 보자. 2^{16}이 표현할 수 있는 최댓값은 65,536이다. 양수와 음수가 있으므로 2바이트가 표현할 수 있는 최대 양수 값은 32,768이다. 다시 말해 2바이트로 된 정수로는 3만 3,000원을 표현할 수 없다.

그렇다면 가계부 프로그램을 짜려면 어떻게 해야 할까? 더 큰 크기를 표현할 수 있도록 정수를 담을 수 있는 크기를 늘려야 한다. 4바이트로 구성된 정수를 사용하면 $2^{32}/2$가 최대 양수 값이므로, 최댓값은 2,147,483,648이 된다. 약 21억 원이 되므로 가계부 프로그램으로 사용하기에 충분한 크기다. 그보다 더 돈이 많다면 8바이트로 구성된 정수를 사용하면 된다.

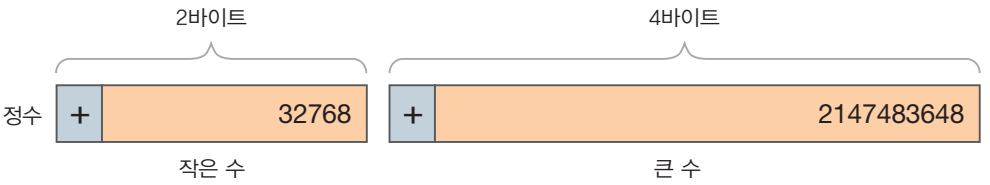

그림 4 정수의 크기와 최댓값

정수와 달리 실수는 큰 수를 표현하기에 적합한 구성이다. 예를 들어 21억을 실수로 표현하면 0.21×10^{10}이 되고, 10과 21을 보관하면 된다. 2조 1,000억 원을 표현하고 싶다면 0.21×10^{13}으로 표현하면 된다. 그렇다면 4바이트로 구성된 실수를 8바이트로 구성하면 어떤 이점이 생길까? 정밀도에 차이가 생긴다.

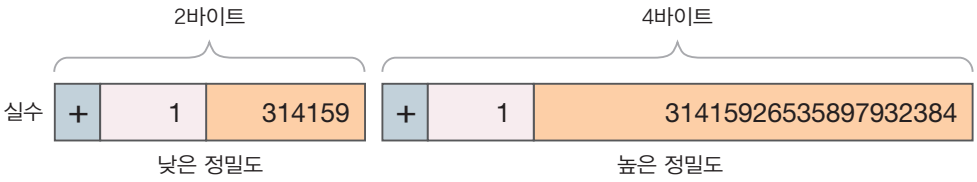

그림 5 실수의 크기와 정밀도

예를 들어 원주율은 3.14159265358979323846264 3이다. 이를 4바이트로 구성된 실수에 보관하면 다 들어가지 못하고 3.14159 정도만 들어간다. 3.14159265358979323846264 3을 다 보관하고 싶다면 8바이트로 구성된 실수에 넣어야 한다. 결론적으로 큰 크기로 구성된 실수는 작은 크기의 실수보다 정밀한 값을 사용할 수 있게 한다. 따라서 정밀한 값을 다루는 프로그램을 짠다면 원래 크기보다 2배 더 큰 실수인 배정도 실수(double float)를 사용해야 한다.

4. 논리 게이트

논리 연산은 컴퓨터의 논리 회로를 만드는 데 사용하는데, 이를 논리 게이트라고 한다. AND 게이트는 입력 값 2개에 전기가 동시에 들어와야 전기가 흐르는 회로다. OR 게이트는 둘 중 1개의 선에만 전기가 흘러도 전기가 통하는 회로다. XOR 게이트는 둘 중 1개에만 전기가 흐를 때 전기가 통하는 회로다.

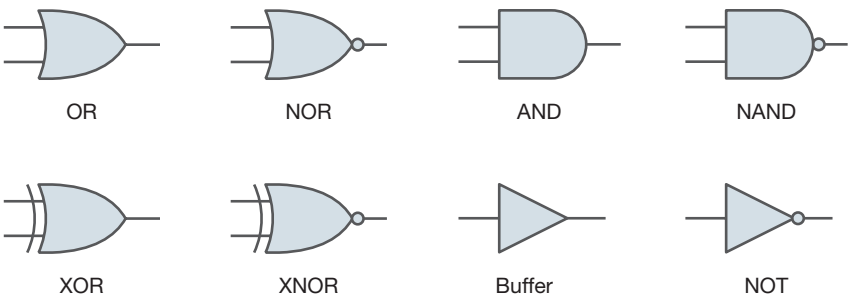

그림 6 논리 게이트 심벌

[그림 6]은 논리 게이트의 심벌을 나타낸다. 회로 설계를 할 때 AND나 OR 대신에 그림의 심벌을 사용하여 설계도를 그린다. 심볼에서 특징적인 것은 AND, OR, XOR 논리 게이트의 NOT 버전인 NAND, NOR, XNOR이다. NAND는 AND의 반대 값, NOR은 OR의 반대 값, XNOR은 XOR의 반대 값이 출력된다. 각 진리표는 [표 1]과 같다.

표 1 논리 연산 진리표

NAND 연산			NOR 연산			XNOR 연산		
입력		출력	입력		출력	입력		출력
T	T	F	T	T	F	T	T	T
T	F	T	T	F	F	T	F	F
F	T	T	F	T	F	F	T	F
F	F	T	F	F	T	F	F	T

논리 게이트를 사용하면 우리가 생각하는 여러 가지 연산을 쉽게 할 수 있다. 예를 들어 NOT 게이트를 사용하면 쉽게 1의 보수를 만들 수 있다. 앞서 숫자의 0과 1을 바꾸어 주면 1의 보수가 된다고 설명했다. 따라서 입력되는 숫자에 NOT 게이트를 연결하여 출력하면 1의 보수를 만들 수 있고, 이 출력 값 중 음수 값에만 +1을 더하면 2의 보수가 된다.

SECTION 02

버스 종류와 캐시 구조

본문 연계: 4장

1. 버스 종류

메인보드에 있는 버스를 시스템 버스 혹은 전면 버스(FSB)라고 한다. 시스템 버스는 앞서 설명한 대로 메모리와 하드디스크, 프린터 같은 주변 장치를 연결하는 버스다. CPU 내부에 있는 장치들도 버스로 연결하데, 이를 후면 버스(Back Side Bus, BSB)라고 한다. 후면 버스의 속도는 CPU의 클록과 같다. 보통 후면 버스가 시스템 버스보다 훨씬 빠르다. 전면 버스에는 메모리가 연결되어 있기 때문에 메모리도 전면 버스 속도로 작동된다. CPU는 후면 버스 속도로 작동하고, 메모리는 전면 버스 속도로 작동하기 때문에 속도 차이로 작업이 지연된다. 이러한 속도 차이로 일어난 문제를 완화시켜 주는 장치가 캐시(cache)다.

지금까지는 버스를 한 줄로 그렸으나, 시스템 버스나 후면 버스에는 세 종류의 데이터가 왕래한다. 그래서 세 종류의 버스가 존재한다.

- **제어 버스**: 제어 버스(control bus)는 제어 장치가 명령을 내릴 때 사용하는 버스다. 제어 버스는 제어 장치가 메모리에서 데이터를 가져오거나, CPU에 있는 데이터를 내보내거나, 하드디스크에서 데이터를 가져오거나, 소리를 사운드 카드로 보내는 등 모든 동작에 사용한다. 제어 버스는 명령이 전달되는 버스다.
- **데이터 버스**: 명령이 전달되면 실제로 데이터가 이동하는데, 이때 사용하는 버스가 데이터 버스(data bus)다. 데이터 버스는 CPU와 메모뿐 아니라 모든 장치 간에 데이터가 이동하는 통로다. 우리가 지금까지 배운 버스는 데이터 버스다.
- **주소 버스**: 주소 버스(address bus)는 데이터를 가져오거나 내보낼 때 어디로 보내야 하는지 결정하는 버스다. 메모리의 100번지와 120번지에 있는 숫자를 가져오거나 결과를 160번지에 저장할 때 100번지, 120번지, 160번지의 주소는 주소 버스로 전달된다. 메모리뿐 아니라 하드디스크에도 주소 버스가 사용된다. 하드디스크의 몇 번째 섹터에서 데이터를 가져올지 혹은 몇 번째 섹터에 데이터를 저장할지 결정하는 버스가 주소 버스다.

버스 3개와 주변 장치를 같이 그리면 [그림 7]과 같다.

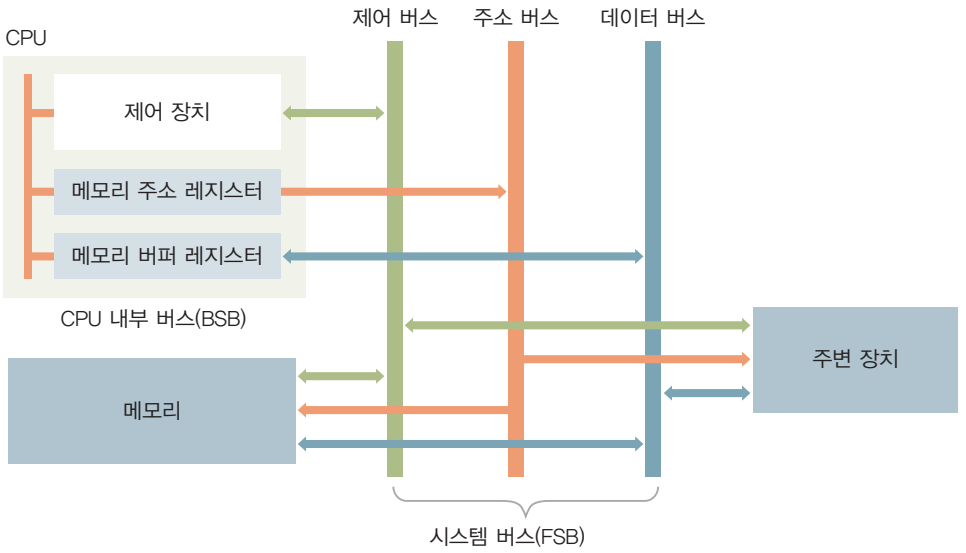

그림 7 버스의 종류와 구조

제어 버스는 다음에 어떤 작업을 할지 지시하는 제어 신호가 오간다. 메모리에서 데이터를 가져올지 아니면 처리한 데이터를 옮겨 놓을지에 대한 지시정보가 오가는데, 메모리에서 데이터를 가져올 때는 읽기 신호를 보내고 처리한 데이터를 메모리로 옮겨 놓을 때는 쓰기 신호를 보낸다. 주변 장치도 마찬가지로 하드디스크에 저장 명령을 내리거나 사운드 카드에 소리를 내라는 명령을 내릴 때도 제어 버스로 전달된다. [그림 7]을 보면 제어 버스는 CPU의 제어 장치와 연결되어 있다. 메모리에서 오류가 발생하거나 네트워크 카드에 데이터가 모두 도착했다는 신호는 모두 제어 버스로 CPU에 전달된다. 제어 버스의 신호는 CPU, 메모리, 주변 장치와 양방향으로 오간다.

주소 버스는 메모리의 데이터를 읽거나 쓸 때 어느 위치에 작업할지 알려 주는 위치 정보(주소)가 오간다. 주변 장치도 마찬가지로 하드디스크의 어느 위치에서 데이터를 읽어 올지 혹은 어느 위치에 저장할지에 대한 위치 정보를 주소 버스로 전달한다. 주소 버스는 메모리 주소 레지스터(Memory Address Register, MAR)와 연결되어 있다. 메모리 주소 레지스터는 주소만 다루는 특수한 레지스터다. 다른 버스와 달리 주소 버스는 단방향이다. CPU에서 메모리나 주변 장치로 나가는 주소 정보는 있지만, 주소 버스를 통해 CPU로 전달되는 정보는 없다. 제어 버스가 다음에 어떤 작업을 할지 신호를 보내고, 주소 버스가 위치정보를 전달하면 데이터는 데이터 버스에 실려 목적지까지 이동한다. 데이터 버스는 양방향 버스이며 메모리 버퍼 레지스터(Memory Buffer Register, MBR)와 연결된다. 메모리 버퍼 레지스터는 가져온 데이터 혹은 내보낼 데이터가 임시로 저장되는 레지스터다.

표 2 버스의 종류와 특징

버스	특징
제어 버스	제어 장치와 연결된 버스로, CPU가 메모리 및 주변 장치에 제어 신호를 보내는 데 사용한다. 메모리 및 주변 장치에서도 작업이 완료되거나 오류가 발생하면 제어 신호를 보내기 때문에 양방향이다.
주소 버스	메모리나 주변 장치에 데이터를 읽거나 쓸 때, 직업 위치 정보를 보내는 데 사용한다. 단방향이다.
데이터 버스	데이터가 오고 가며, 데이터 이동은 양방향이다.

2. 캐시 구조

앞서 설명했듯이, CPU에 비하여 메모리는 느린 장치다. 캐시는 빠른 속도로 작동하는 CPU와 느린 속도로 작동하는 메모리 사이에서 두 장치의 속도 차이를 완화시켜 준다. CPU에 있는 캐시를 정의하면 매우 빠른 속도로 작동하는 메모리라고 할 수 있다.

캐시가 어떻게 작동하는지 [그림 8]에 나타냈다. 캐시는 메모리 내용 중 일부를 미리 가져온다. CPU가 메모리에 접근해야 할 경우, 캐시를 먼저 방문하여 원하는 데이터가 있는지 찾아본다. 캐시에서 원하는 데이터를 찾았다면 캐시 적중(cache hit)이고, 해당 데이터를 바로 사용한다. 원하는 내용이 캐시에 없다면 메모리로 가서 데이터를 찾는다. 이를 캐시 실패(cache miss)라고 한다. 캐시가 적중하는 비율을 캐시 적중률(cache hit ratio)이라고 하는데, 일반적인 컴퓨터의 캐시 적중률은 약 90%다.

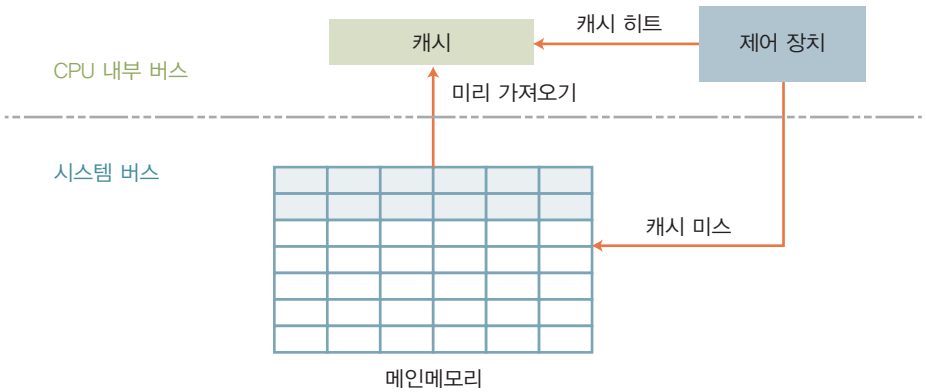

그림 8 캐시 구조

컴퓨터 성능을 향상시키려면 캐시 적중률이 높아야 한다. 캐시 적중률을 높이는 방법은 캐시 크기를 늘리는 것이다. 캐시 크기가 늘어나면, 미리 가져올 수 있는 데이터가 많아져 적중률이 높아진다. 클록이 같은 CPU의 경우, 저가형과 고가형 차이는 캐시 크기 차이다. 예를 들어 인텔의 저가 i7은 4MB인 반면에 고가의 i7은 8MB 이상의 캐시 메모리를 가진다. 캐시는 가격이 비싸기 때문에 크기를 늘리는 데 한계가 있어 수 MB 정도만 사용한다.

지금까지 캐시에 데이터를 가져오는 문제만 언급했는데, 캐시에 있는 데이터를 변경할 때 이를 반영하는 문제도 있다. 캐시는 메모리에 있는 값을 임시로 가져온 것이다. 따라서 캐시에 있는 데이터가 변경되면 메모리에 있는 원래 값을 변경시켜야 한다. 캐시에서 변경된 데이터를 메모리에 반영하는 방법에는 즉시 쓰기 방식과 지연 쓰기 방식이 있다.

즉시 쓰기(write through)는 캐시에 있는 데이터가 변경되면 이를 즉시 메모리에 반영하는 방식이다. 메모리와 빈번하게 데이터를 전송하므로 성능이 느려지는 단점이 있으나, 언제나 메모리에 최신 값이 유지되기 때문에 급작스러운 정전에도 데이터를 잃어버리지 않는다는 장점이 있다.

지연 쓰기(write back)는 카피 백(copy back) 방식이라고도 하는데, 메모리에 즉시 반영하는 것이 아니라 주기적으로 변경된 내용을 모아 메모리에 반영하는 방식이다. 메모리와 데이터를 전송하는 횟수가 줄어 시스템 성능은 높일 수 있으나, 메모리와 캐시된 데이터 사이에 불일치가 발생할 수 있는 단점이 있다.

3. 부팅

컴퓨터 전원을 켜 운영체제를 메모리로 올리는 과정을 부팅(booting)이라고 한다. 전원을 켜면 [그림 9]와 같이 윈도우 로고가 뜨고, 막대기가 좌우로 움직이는 화면을 보았을 것이다. 컴퓨터의 각 하드웨어를 점검하고, 윈도우를 실행할 수 있는 상태까지의 과정이 부팅이다.

그림 9 윈도우 부팅 화면

부팅에서는 다양한 작업이 일어난다. 사용자가 컴퓨터 전원을 켜면 바이오스(Basic Input/Output System, BIOS)가 실행된다. 바이오스는 메인보드에 있는 롬에 담겨 있다. 바이오스 역할은 CPU, 메모리, 하드디스크, 키보드, 마우스 등 주요 하드웨어가 제대로 동작하는지 점검하는 것이다. 하드웨어에 이상이 있다면 '삐~' 소리와 함께 오류 메시지를 출력하고, 컴퓨터는 더 이상 작동하지 않는다.

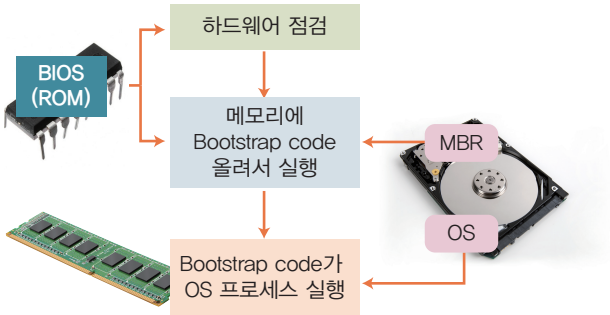

그림 10 부팅 과정

바이오스 검사가 통과되면, 하드디스크의 마스터 부트 레코드(Master Boot Record, MBR)에 저장된 작은 프로그램을 메모리로 가져와 실행한다. 마스터 부트 레코드는 하드디스크의 가장 첫 번째 섹터를 가리키는 단어다. 이곳에 운영체제를 실행하는 코드인 부트스트랩(bootstrap)이 저장된다. 부트스트랩 코드는 운영체제를 메모리로 가져와 실행시키는 역할을 하는 작은 프로그램이다. 예를 들어 유닉스용 부트스트랩이 실행되면 유닉스 운영체제가 메모리에 올라오고, 윈도우용 부트스트랩이 실행되면 윈도우 운영체제가 메모리에 올라온다. 요즘은 USB 드라이브에도 마스터 부트 레코드를 탑재할 수 있다. 흔히 듣는 '부팅 USB 만들기'나 '부팅 DVD 만들기'는 USB나 DVD에 마스터 부트 레코드 영역을 만들고 필요한 부트스트랩 코드를 마스터 부트 레코드에 설치하는 것이다. 마스터 부트 레코드에 있는 부트스트랩이 메모리에 올라오면 하드디스크에 저장된 운영체제를 메모리로 불러온다. 따라서 마스터 부트 레코드가 손상되면 운영체제를 실행할 수 없다.

부트스트랩을 경험하려면 한 컴퓨터에 운영체제 2개를 설치해 보면 된다. 한 컴퓨터에 윈도우 7과 윈도우 10을 같이 설치하거나 리눅스와 윈도우 운영체제를 같이 설치하는 것이다. 이 경우 바이오스가 하드웨어 점검을 마치고 마스터 부트 레코드에서 부트스트랩을 가져와 실행하면, 어떤 운영체제로 사용할지 묻는 화면이 나타난다. 이 화면이 부트스트랩 코드가 실행된 화면이다. 운영체제 1개가 설치된 컴퓨터의 부트스트랩은 이러한 과정 없이 바로 운영체제를 메모리에 올린다.

SECTION 03

인터럽트와 저장 장치의 계층 구조

본문 연계: 6장

1. 프로세스 우선순위

컴퓨터 내에는 사용자가 실행한 프로세스만 있는 것이 아니다. 운영체제도 프로그램이기 때문에 프로세스 형태로 실행해야 한다. 컴퓨터에는 일반 사용자의 프로세스와 운영체제의 커널 프로세스가 섞여서 실행된다.

레스토랑이 꽉 차서 자기 차례를 기다리는 손님들을 상상해 보자. 우선순위가 있다는 것은 프로세스 중요도가 다르다는 의미다. 레스토랑에서도 예약 손님은 일반 손님보다 우선순위가 높다. 예약을 했는데도 일반 손님처럼 줄을 서야 한다면 굳이 예약을 할 필요가 없을 것이다. 마찬가지로 컴퓨터 내 프로세스들은 각기 다른 우선순위를 가진다.

프로세스는 크게 커널 프로세스와 일반 프로세스로 나뉜다. CPU 스케줄러는 각 프로세스에 우선순위를 부여하는데, 커널 프로세스의 우선순위가 일반 프로세스보다 높다. 커널 프로세스와 일반 프로세스를 같은 우선순위로 처리한다면 커널과 관련한 중요한 프로세스 작업을 제대로 할 수 없을 것이다.

우선순위가 높다는 것은 더 빨리 자주 실행한다는 의미다. 준비 상태의 커널 프로세스와 일반 프로세스가 하나씩 있다면, 커널 프로세스의 우선순위가 더 높기 때문에 커널 프로세스를 먼저 실행하며 작업이 끝날 때까지 계속 CPU를 사용한다. 또 같은 커널 프로세스라고 하더라도 더 중요한 커널 프로세스는 우선순위가 높고, 덜 중요한 프로세스는 우선순위가 낮다.

일반 프로세스도 중요도가 각각 다르기 때문에 우선순위가 서로 다르다. 예를 들어 워드프로세서와 비디오 플레이어 중에서 비디오 플레이어의 우선순위가 더 높다. 워드프로세서는 사람이 타이핑하는 속도가 CPU 연산 속도보다 느리기 때문에 천천히 실행하여도 문자 입력 처리가 가능하다. 그러나 비디오 플레이어는 실시간으로 데이터를 읽어 와서 영상과 소리를 출력해야 하기 때문에 자주 실행하지 않으면 화면이 끊긴다.

일반 프로세스의 우선순위는 사용자가 조절할 수 있다. [그림 11]은 윈도우 운영체제에서 비디오 플레이어 중 하나인 팟플레이어의 우선순위를 조절하는 화면이다. 작업 우선권 항목을 조절하면 우선순위를 높이거나 낮출 수 있다. 우선순위를 조절할 때는 해당 프로세스뿐 아니라 다른 프로세스의 실행 속도에도 영향을 미친다는 것을 주의해야 한다.

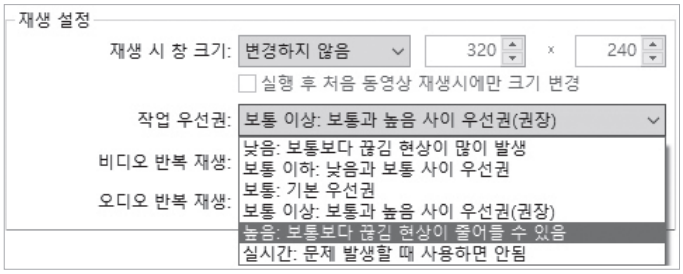

그림 11 프로세스의 우선순위 변경

비디오 플레이어의 우선순위를 너무 높이면 CPU를 거의 비디오 플레이어에만 사용하기 때문에 다른 프로세스 작업이 원활하지 않다.

워드프로세서를 사용하면서 큰 파일을 압축하는 경우를 생각해 보자. 워드프로세서는 사용자에게 키보드로 입력을 받아야 하기 때문에 전면 프로세스로 실행되고, 압축 프로그램은 압축이 끝날 때까지 사용자 입력이 필요 없기 때문에 후면 프로세스로 실행된다. 전면 프로세스는 GUI를 사용하는 운영체제에서 화면의 맨 앞에 놓인 프로세스를 의미한다. 현재 입력과 출력을 사용하는 프로세스이며, 사용자와 상호 작용이 가능하여 상호 작용 프로세스라고도 한다. 후면 프로세스는 사용자와 상호 작용이 없는 프로세스다. 압축 프로그램처럼 사용자 입력 없이 작동하기 때문에 일괄 작업 프로세스라고도 한다.

[그림 12]는 윈도우에서 전면 프로세스와 후면 프로세스의 차이를 보여 준다. 윈도우에서 웹 브라우저와 아래아한글 프로그램이 실행 중인데, 현재 운영체제에서 키보드와 화면 사용 허가를 받은 맨 앞의 프로세스는 웹 브라우저다. 즉, 웹 브라우저는 전면 프로세스고 아래아한글은 후면 프로세스다. 전면 프로세스와 후면 프로세스 간 전환은 화면 아래 작업 표시줄의 아이콘을 클릭하면 된다.

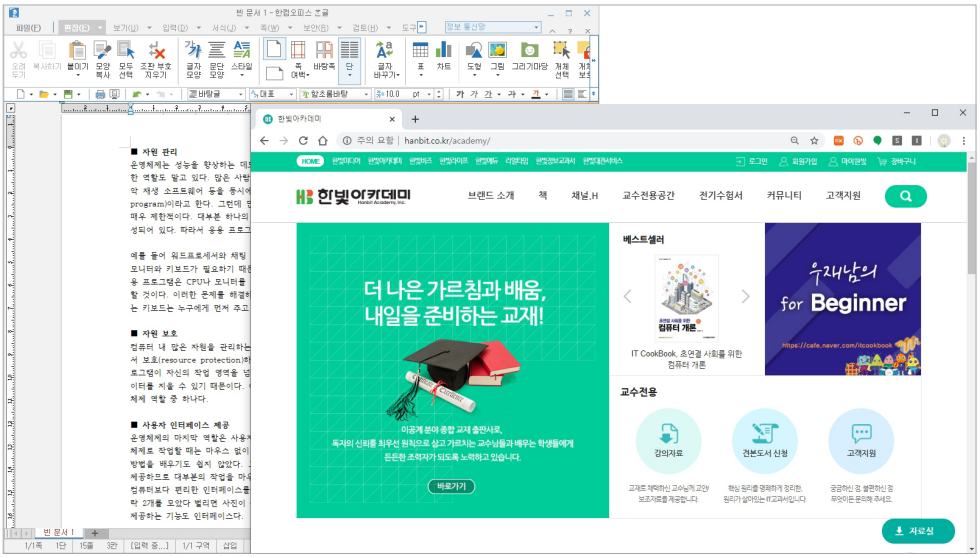

그림 12 전면 프로세스와 후면 프로세스

전면 프로세스는 사용자 요구에 즉각 반응해야 하지만, 후면 프로세스는 상호 작용이 없다. 따라서 전면 프로세스의 우선순위가 후면 프로세스보다 높다. 그만큼 후면 프로세스는 전면 프로세스보다 CPU를 할당받을 확률이 적다.

[그림 13]은 지금까지 설명한 우선순위에서 고려할 사항을 정리한 것이다. 작업 중요도가 높은 프로세스, 즉 우선순위가 높은 프로세스는 커널 프로세스, 전면 프로세스, 대화형 프로세스다. 반대로 우선순위가 낮은 프로세스는 일반 프로세스, 후면 프로세스, 일괄 작업 프로세스다.

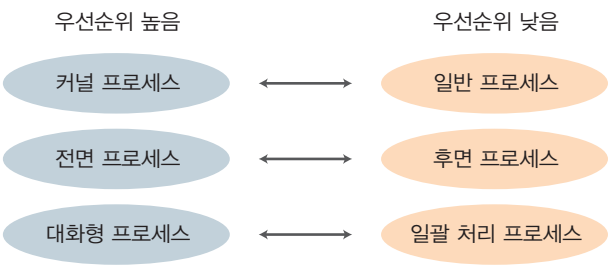

그림 13 프로세스의 우선순위

2. 인터럽트

초기 컴퓨터 시스템에는 주변 장치가 많지 않았다. 당시에는 CPU가 직접 입출력 장치에서 데이터를 가져오거나 내보냈는데, 폴링(polling) 방식이라고 한다. 이를 요리사 모형에 비유하면 요리사가 요리를 하다 재료가 필요하면 보관 창고에서 직접 가져오는 것과 같다. 폴링 방식에서는 CPU가 입출력 장치의 상태를 주기적으로 검사하여 일정한 조건을 만족할 때 데이터를 처리한다. CPU가 명령어 해석과 실행이라는 본래 역할 외에 모든 입출력까지 관여해야 하므로 작업 효율이 떨어진다.

그림 14 폴링 방식

오늘날 컴퓨터에는 많은 주변 장치가 있기 때문에 CPU가 모든 입출력에 관여하면 작업 효율이 현저하게 떨어진다. 이러한 문제를 해결하려고 등장한 것이 인터럽트(interrupt) 방식이다. 인터럽트 방식을 요리사 모형에 비유하면 요리사 옆에 주방 보조를 두는 것과 같다.

그림 15 인터럽트 방식

요리사는 필요한 재료를 가져오도록 주방 보조에게 지시하고 자신은 계속 요리를 하며, 지시를 받은 주방 보조는 재료를 가져다 도마에 올려놓고 재료가 준비되었다는 것을 요리사에게 알려 준다. 인터럽트 방식은 CPU 작업과 저장 장치의 데이터 이동을 독립적으로 운영함으로써 시스템 효율을 높인다. 즉, 데이터의 입출력이 실행되는 동안 CPU가 다른 작업을 할 수 있다. 이때 입출력 관리자 또는 장치 관리자는 주방 보조에 해당한다.

인터럽트 방식의 동작 과정을 자세히 살펴보자.

❶ CPU가 입출력 관리자에게 입출력 명령을 보낸다.
❷ 입출력 관리자는 명령받은 데이터를 메모리에 가져다 놓거나 메모리에 있는 데이터를 저장 장치로 옮긴다.
❸ 데이터 전송이 완료되면 입출력 관리자는 완료 신호를 CPU에 보낸다.

입출력 관리자가 CPU에 보내는 완료 신호를 인터럽트라고 한다. CPU는 입출력 관리자에게 작업 지시를 내리고, 다른 일을 하다가 완료 신호를 받으면 하던 일을 중단하고 옮긴 데이터를 처리한다. 이처럼 하던 작업을 중단하고 처리해야 하는 신호라는 의미에서 인터럽트라고 불렀다.

컴퓨터에는 하드디스크뿐 아니라 마우스, 키보드, 프린터 등 다양한 입출력 장치가 있다. 하드디스크가 여러 개 장착된 경우도 있고 USB 드라이버 같은 외부 저장 장치를 사용하는 경우도 있다. 인터럽트 방식에서는 많은 주변 장치 중 어떤 것의 작업을 끝냈는지 CPU에 알려주고자 인터럽트 번호(interrupt number)를 사용한다. 인터럽트 번호는 완료 신호를 보낼 때 장치 이름 대신 사용하는 장치의 고유번호로, 운영체제마다 다르다. 윈도우 운영체제는 인터럽트 번호를 IRQ(Interrupt ReQuest)라고 하며, 키보드의 IRQ는 1번, 마우스의 IRQ는 12번, 첫 번째 하드디스크의 IRQ는 14번처럼 구분해서 사용한다.

3. 메모리 관련 이슈

배치 작업을 생각해 보자. 컴퓨터에서 동작하는 프로세스 크기는 제각각이다. 따라서 메모리를 어떤 크기로 나눌 것인지 하는 문제가 있다. 메모리를 나누는 방식에는 크게 가변분할 방식(variable-size partitioning)과 고정분할 방식(fixed-size partitioning)이 있다. 가변분할 방식은 프로세스 크기에 비례하여 메모리를 나누는 것이며, 고정분할 방식은 프로세스 크기와 상관없이 메모리를 같은 크기로 나누는 것이다.

식당에 의자를 배치하는 경우를 생각해 보자. 손님 개개인의 신체 크기는 다르므로 의자를 크기별로 준비했다가 제공하는 방법, 손님의 신체 크기와 상관없이 크기가 같은 의자를 준비했다가 제공하는 방법이 있을 것이다. 손님의 신체 크기에 맞게 의자를 준비하는 것은 가변분할 방식에 해당하고, 손님의 신체 크기와 상관없이 크기가 같은 의자를 준비하는 것은 고정분할 방식에 해당한다.

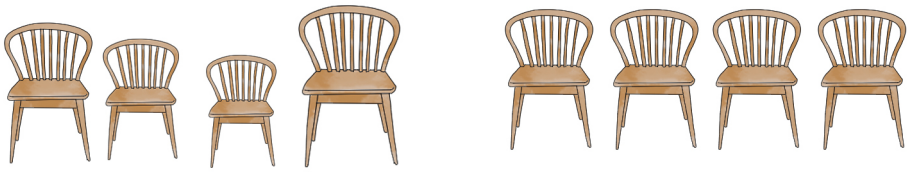

그림 16 가변분할 방식과 고정분할 방식

가변분할 방식은 손님에게 맞는 의자를 제공한다는 측면에서는 바람직하지만 관리 측면에서는 매우 불편하다. 크기가 다른 의자를 쌓아서 보관하기 어려울 뿐더러, 어떤 날은 몸집이 큰 손님만 올 수도 있고 어떤 날은 꼬마 손님만 올 수도 있으니 손님을 모두 만족시키려면 의자를 많이 준비해야 할 것이다. 그러나 고정분할 방식은 모든 의자의 크기가 같기 때문에 쌓아서 보관하기 수월하다. 또 손님의 신체 크기를 고려하지는 않았지만 체구가 큰 손님이 오면 의자 2개를 붙여 주면 된다.

가변분할 방식의 메모리 관리를 세그먼테이션(segmentation)이라고 하며, 고정분할 방식의 메모리 관리를 페이징(paging)이라고 한다. 책의 모든 페이지는 크기가 똑같기 때문에 고정분할 방식의 이름이 페이징이다. 가변분할 방식은 메모리를 관리하기 어려우므로 대부분의 운영체제는 페이징을 사용한다.

[그림 17]은 가변분할 방식과 고정분할 방식의 차이점을 보여 준다. 가변분할 방식은 사용자 메모리를 프로세스 크기별로 잘라서 사용한다. 의자 예에서 설명했듯이, 프로세스가 끝나고 나면 남는 공간의 크기는 제각각이므로 빈 공간을 관리하기 어렵다는 단점이 있다. 고정분할 방식은 프로세스 크기에 상관없이 같은 크기로 메모리를 자른다. 큰 프로세스는 메모리를 여러 개 사용하게 된다. 그림에서 프로세스 A는 고정분할 영역 2개를 사용한다. 고정분할 방식은 메모리를 같은 크기로 잘랐기 때문에 빈 공간을 관리하기 편하다는 장점이 있다.

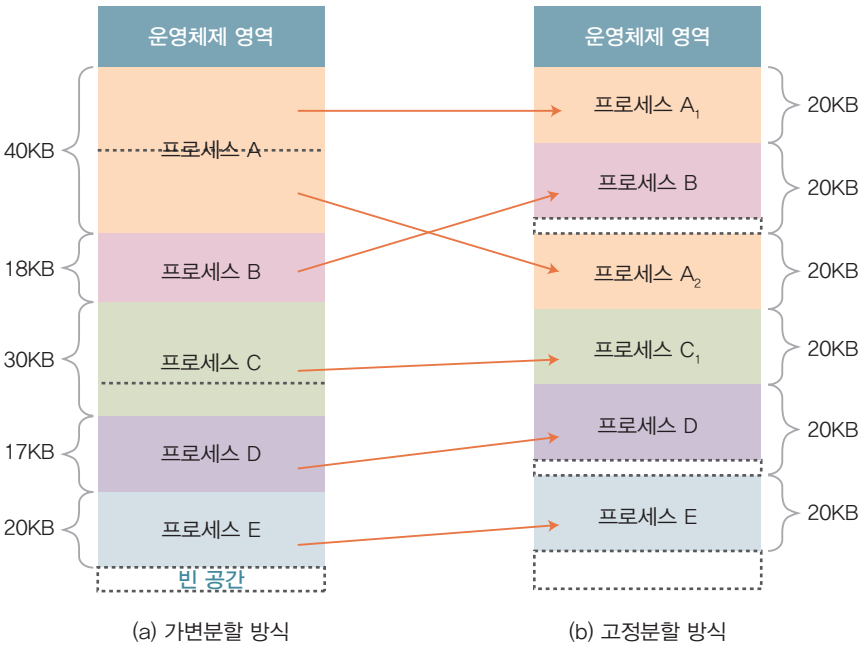

그림 17 가변분할 방식과 고정분할 방식 차이

가상 메모리 시스템은 자신이 가진 메모리보다 큰 프로그램을 실행시킬 수는 있지만, 충분히 큰 메모리를 확보하지 못하면 시스템이 느려진다. 실제 메모리 크기가 작으면 새로운 프로세스를 실행할 때마다 메모리에 있는 프로세스를 스왑으로 옮겨 놓은 후 가져와야 한다. 이것은 컴퓨터 성능을 떨어뜨린다. 그래서 실제 메모리 크기가 작아 컴퓨터 속도가 느릴 때 메모리를 늘리면 성능이 올라가는 것이다.

[그림 18]은 동시에 실행되는 프로세스 개수(멀티 프로그램 개수)와 CPU 사용률을 나타낸 그래프다. 처음 프로세스 1개를 실행할 때는 메모리가 넉넉하기 때문에 큰 무리 없이 동작한다. 프로세스를 계속 실행시켜 보자. 프로세스 개수가 늘어날수록 메모리를 점점 채워 갈 것이다. 그러다가 하드디스크는 계속 작동하는데 프로세스는 동작을 못하는 순간이 온다. 이때는 메모리가 꽉 차서 새로운 프로세스를 가져오려면 기존 프로세스는 스왑으로 쫓아내야 하는 순간이다. 메모리에 있는 프로세스가 나가고 새로운 프로세스가 메모리에 들어오려고 느린 하드디스크에 계속 접근하므로 프로세스가 작업을 거의 못하게 된다. 이러한 순간을 스레싱(threshing)이 발생했다고 한다. 스레싱은 빈번한 스왑으로 하드디스크에 접근이 많아 작업을 거의 못하는 상태를 가리킨다.

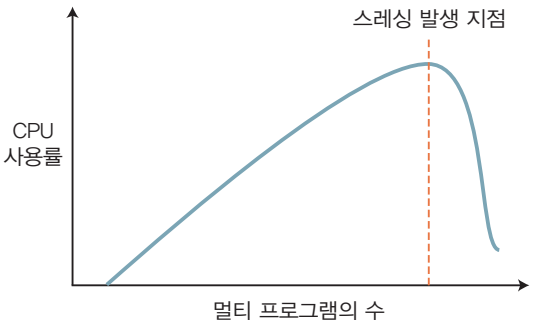

그림 18 스레싱 개념

4. 저장 장치의 계층 구조

최고 성능을 내는 컴퓨터를 구성하고 싶다면 다음과 같이 여러 방법을 시도할 수 있다. 우선 저장 장치로 느린 하드디스크 대신 SSD 같은 빠른 플래시 메모리를 사용하는 것이다. 아니면 메모리를 하드디스크 크기만큼 확장하는 것이다. 다시 말해 메모리는 1TB로 구성하여 하드디스크를 통째로 가져와 작업하면 컴퓨터 속도는 빨라질 것이다. 더 나아가 캐시를 크게 늘려 4GB 캐시를 사용하는 방법도 있다. 이 경우 메모리의 모든 데이터를 캐시로 가져옴으로써 메모리로 가지 않아도 CPU 내부에서만 작업이 가능해진다.

그러나 이러한 방법들은 돈이 많이 든다. 하드디스크 1TB는 약 4만 원이지만, SSD 1TB는 40만 원이다. 1MB에 1만 원하는 캐시로 메모리 4GB를 구성하려면 약 4,000만 원이 필요하다.

일반인에게 자동차보다 비싼 컴퓨터는 필요 없다. 따라서 가격과 컴퓨터 성능 사이의 타협점이 필요하다. 이러한 타협점이 저장 장치의 계층 구조다. 저장 장치의 계층 구조(storage hierarchy)는 속도가 빠르고 값이 비싼 저장 장치는 CPU와 가까운 쪽에 두고, 값이 싸고 용량이 큰 장치는 반대쪽에 배치하여 적당한 가격으로 높은 속도와 큰 용량을 동시에 얻는 방법이다. [그림 19]는 저장 장치의 계층 구조를 나타낸 것이다.

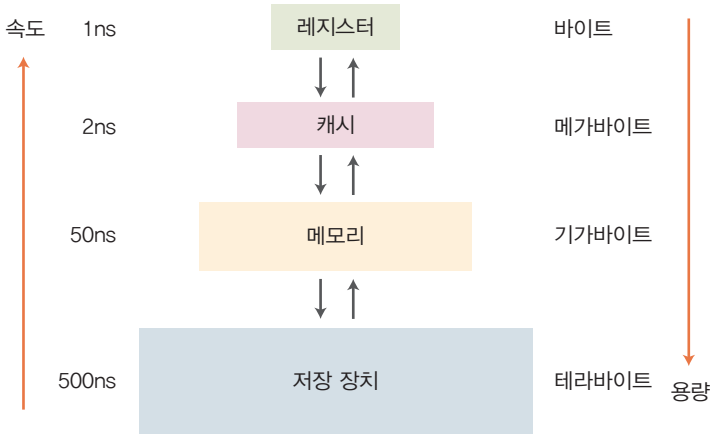

그림 19 저장 장치의 계층 구조

컴퓨터는 CPU와 메모리가 서로 협업하여 작업한다. 그러나 메모리는 CPU보다 느리다. 저장 장치의 계층 구조에서는 CPU와 가까운 쪽에 레지스터나 캐시를 배치하여 CPU가 작업을 빨리 진행할 수 있도록 해 준다. 또 메모리에서 작업한 내용을 하드디스크처럼 저렴하고 용량이 큰 저장 장치에 영구적으로 저장할 수 있게 한다. 저장 장치의 계층 구조는 사용자가 저렴한 가격으로 용량은 하드디스크처럼 사용하고, 작업 속도는 레지스터처럼 빠르게 느끼도록 해 준다.

5. 저장 장치 관련 이슈

파티션은 디스크를 논리적으로 분할하는 작업으로, 보통 파티션 하나에 파일 시스템 하나만 탑재한다. 윈도우에서는 각 파티션마다 알파벳을 부여하는데, 처음에 분할된 파티션에는 C 드라이브, 그다음에 분할된 파티션에는 D 드라이브처럼 C부터 알파벳 순서대로 드라이브 이름을 부여한다.

하드디스크가 2개라면, 첫 번째 하드디스크와 두 번째 하드디스크는 별도의 파티션으로 보인다. [그림 20]의 왼쪽은 하드디스크를 2개 사용한 것이다. 각 디스크에는 별도의 파일 시스템이 탑재되고, 하드디스크도 2개로 보인다.

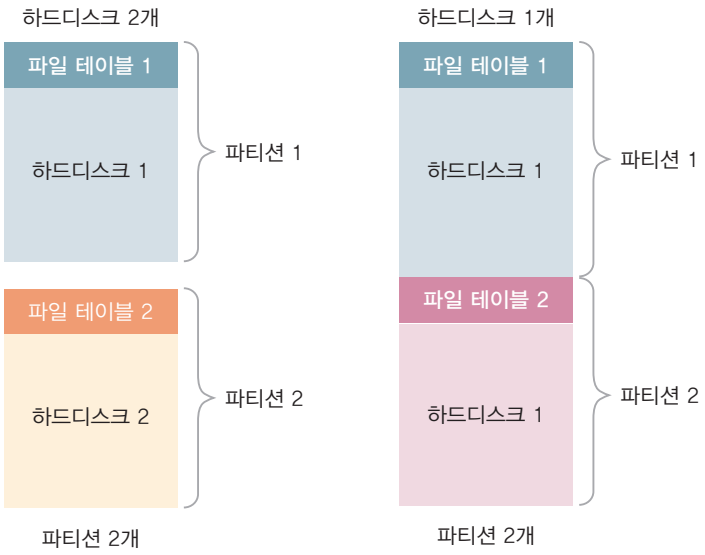

그림 20 디스크와 파티션 관계

대용량 하드디스크는 하나로 사용하기보다 여러 개로 나누어 사용하는 것이 관리하기 편하다. 예를 들어 1TB 하드디스크라면 500MB씩 2개로 나누어 사용하는 것이다. [그림 20]의 오른쪽은 하드디스크 하나를 파티션 2개로 나누어 각 파티션에 파일 시스템을 탑재한 모습이다. 이때도 하드디스크는 하나이지만, 마치 하드디스크 2개가 있는 것처럼 보인다.

윈도우에서 드라이브를 사용할 때는 다음 두 가지 사항에 유의해서 사용하길 바란다.

■ **C, D, E 드라이브 3개만 사용하자**

C 드라이브에는 운영체제와 각종 응용 프로그램을 설치하고, D 드라이브에는 문서와 사진, 영화 등 일반적인 데이터를 저장하며, E 드라이브에는 D 드라이브에 있는 데이터 중 중요한 데이터를 백업하는 용도로 사용하면 좋다. 이렇게 디스크를 나누어 사용하면 시스템 오류나 바이러스로 운영체제를 새로 설치해야 할 때 C 드라이브만 포맷하면 되므로 매우 편리하다. 또 D 드라이브의 데이터 중 중요한 데이터를 E 드라이브에 백업해 놓았기 때문에 중요한 데이터를 잃어버릴 가능성도 줄어든다.

■ **C 드라이브에는 개인인 데이터는 되도록 넣지 말고, 넉넉하게 비워 두자**

개인인 데이터를 C 드라이브에 넣으면 운영체제를 새로 설치할 때 잃어버릴 가능성이 있다. 또 가상 메모리에서 설명한 스왑 영역이 C 드라이브에 있기 때문에 C 드라이브에 여유 공간이 없으면 스왑 영역으로 사용할 공간이 그만큼 줄어들어 메모리 관리에 문제가 생긴다.

SECTION 04

삽입정렬과 버킷정렬

본문 연계: 8장

1. 삽입정렬

삽입정렬은 선택정렬이나 버블정렬과 성능이 비슷한 정렬 방식이다. 실험에 따르면, 선택정렬이나 버블정렬보다 좀 더 나은 성능을 보이는 것으로 나타났다. 그러나 코드를 이해하기에 조금 복잡하다.

삽입정렬의 개념

카드가 한 장만 있을 때는 정렬할 필요가 없다. 따라서 삽입정렬에서 첫 번째 카드 한 장은 정렬이 완료되었다고 가정한다.

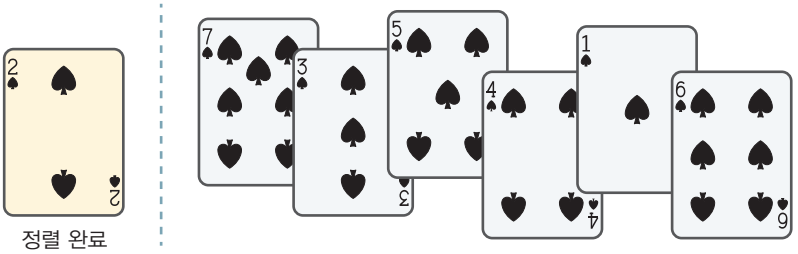

그림 21 삽입정렬의 초기 상태

다음 번 반복에서 그다음 카드인 7을 가져와 정렬될 때까지 카드를 앞으로 보낸다. 7은 2보다 크기 때문에 2–7 순서대로 놓이고 정렬이 완료된다.

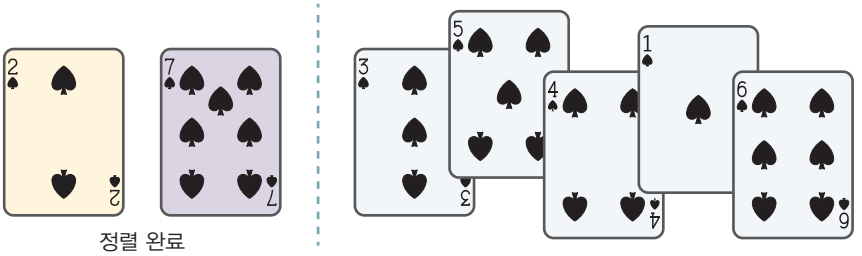

그림 22 삽입정렬의 두 번째 상태

그다음 카드인 3을 보자. 3이 자신의 자리를 찾을 때까지 이미 정렬된 2–7 카드 쪽으로 옮긴다. 카드 7의 앞쪽, 카드 2의 뒤쪽에 삽입시킨다. 그러면 카드 3의 정렬이 완료된다. 이렇게 이미 정렬된 카드에 새로운 카드를 삽입(insert)하면서 정렬하기 때문에 삽입정렬이라고 한다.

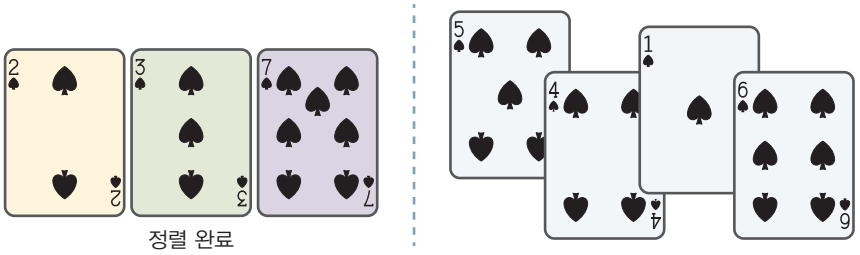

그림 23 삽입정렬의 세 번째 상태

삽입정렬 알고리즘

어떻게 삽입정렬을 하는지 그림으로 살펴보자. 삽입정렬에서 처음 카드 한 장은 이미 정렬했다고 가정하기 때문에 두 번째 값부터 정렬 알고리즘이 작동한다.

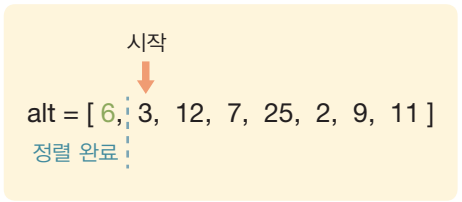

그림 24 첫 번째 값 정렬 완료(가정)

이미 정렬된 값 6에 대해 두 번째 값인 3을 삽입하려면 3이 들어갈 자리를 만들어야 한다. 그래서 var라는 변수에 3을 임시로 저장한다.

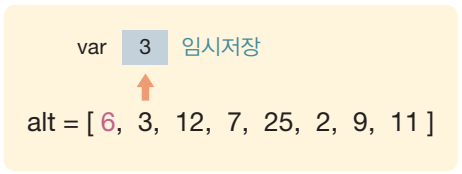

그림 25 두 번째 값을 var 변수에 임시 저장

카드 3이 들어갈 자리를 만들고자 6을 뒤로 옮긴다.

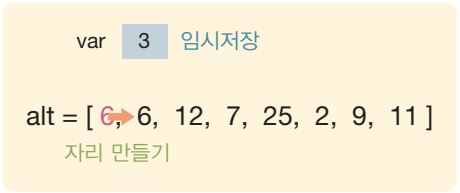

그림 26 첫 번째 값 자리 이동

3이 들어갈 자리를 만들었으면, var 변수에 임시로 저장된 값 3을 옮겨 정렬을 완료한다. 이러한 과정을 모든 수를 정렬할 때까지 반복한다.

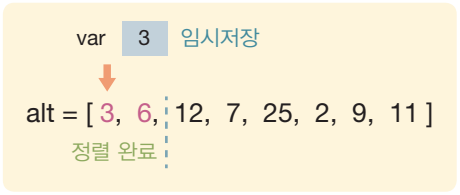

그림 27 첫 번째와 두 번째 값 정렬 완료

이렇게 완료된 삽입정렬을 [코드 1]에 나타냈다. 삽입정렬은 앞서 살펴본 선택정렬이나 버블정렬보다 코드 길이는 짧다. 그러나 코드를 이해하기는 좀 더 어려울 것이다.

코드 1 삽입정렬하기

```
01  alt = [6, 3, 12, 7, 25, 2, 9, 11]
02
03  for j in range(1, 8):
04      var = alt[j]
05      while j > 0 and alt[j-1] > var:
06          alt[j] = alt[j-1]
07          j = j - 1
08      alt[j] = var
09
10      print(alt)
```

실행 결과

```
[3, 6, 12, 7, 25, 2, 9, 11]
[3, 6, 12, 7, 25, 2, 9, 11]
[3, 6, 7, 12, 25, 2, 9, 11]
[3, 6, 7, 12, 25, 2, 9, 11]
[2, 3, 6, 7, 12, 25, 9, 11]
[2, 3, 6, 7, 9, 12, 25, 11]
[2, 3, 6, 7, 9, 11, 12, 25]
```

삽입정렬에서 가장 중요한 코드는 05행의 `while j > 0 and alt[j-1] > var:`이다. 이 코드는 새로운 숫자가 들어갈 자리를 만들려고 기존 숫자를 하나씩 뒤로 밀어 낸다. 여기에서 어디까지 밀어야 할지가 가장 중요하다. 두 가지 조건이 들어가는데, `j > 0`은 리스트의 맨 앞을 벗어나는 것을 방지한다는 의미다. 그리고 `alt[j-1] > var`는 밀어야 하는 숫자가 지금 삽입해야 하는 숫자보다 클 때까지만 뒤로 밀라는 의미다.

2. 알고리즘 성능 평가: 빅오 표기

누군가 새로운 알고리즘을 개발했다고 가정하자. 기존 알고리즘과 비교하여 이 알고리즘이 더 성능이 우수하다는 것을 어떻게 증명할 수 있을까? 가장 간단한 방법은 기존 알고리즘과 새로 만든 알고리즘에 같은 데이터를 사용하여 끝날 때까지 시간을 측정해 보는 것이다. 그러나 기존 알고리즘을 어떻게 프로그래밍했느냐에 따라 시간은 유동적으로 변할 것이다. 새로 만든 알고리즘은 빠르게 작동하도록 프로그래밍하고, 기존 알고리즘은 겨우 작동만 하도록 프로그래밍한다면 객관적으로 비교할 수 없다. 따라서 좀 더 객관적인 비교 방식이 필요하다.

앞서 7장의 알고리즘 분석에서 시간의 복잡도를 이야기했다. 알고리즘 분야에서 성능을 나타내는 지표로 빅오(Big O)를 사용한다. 빅오는 중심이 되는 계산에 대해 최악의 값을 성능으로 나타내는 것이다. 지금까지 배운 정렬 알고리즘으로 빅오를 만드는 방법을 알아보자.

빅오를 만들기 위해서는 중심이 되는 계산을 찾아야 한다. 가장 빈번하게 일어나는 계산은 중심이 되는 계산이다. 정렬 알고리즘에서 중심이 되는 계산은 비교 연산이다. 다시 말해 두 수에서 어떤 수가 큰지 작은지를 비교하는 연산이다. 앞서 본 선택정렬, 버블정렬, 삽입정렬 모두 두 수를 비교하여 자리를 바꾸는 연산을 한다. 그래서 비교 연산이 중심 연산이 된다.

중심 연산을 찾았다면, 데이터 N개에서 중심 연산을 몇 번 계산했는지 찾는다. 선택정렬에서 비교 연산을 하는 위치를 보면 반복문 2개 안에 있다. 또 반복문은 데이터 N개에 대하여 거의 N번씩 실행한다. 따라서 입력 값 N에 대하여 반복문 j에서 N번을 반복하고, 반복문 k에서 N번을 반복하므로 비교 연산은 $N \times N$, 즉 N^2번 실행한다.

```
for j in range(8):  N번 반복
    for k in range(j+1, 8):  N번 반복
        if ( alt[min] > alt[k] ):  비교 연산
```

그림 28 선택정렬의 중심 연산 – 비교 연산

버블정렬과 삽입정렬도 반복문 구조가 거의 비슷하기 때문에 N^2번 비교 연산을 한다. 정확하게 말하면 N^2보다는 적은 연산을 한다. 그러나 빅오는 이를 단순화시킨다. 예를 들어 A라는 알고리즘은 N^2-2N, B라는 알고리즘은 $2N^2+3N$이라고 가정해 보자. 이를 빅오로 나타내면 [그림 29]와 같다.

$N^2 - 2N \rightarrow O(N^2)$
$2N^2 + 3N \rightarrow O(N^2)$

그림 29 빅오 표기법

빅오는 연산식 중 가장 큰 차수의 연산을 제외하고, 상수와 낮은 차수의 연산을 없앤다. 사실 N^2-2N 연산을 하는 A 알고리즘은 조금 억울할 것이다. 그러나 N이 커진다면, 다시 말해 데이터 개수가 늘어나면 두 알고리즘 차이는 매우 작아진다.

정렬 알고리즘은 $O(N^2)$ 계열과 $O(N \log N)$ 계열로 나눌 수 있다. 이때 N이 1,000이라면 $O(N^2)$ 계열 알고리즘은 100만 번 계산하지만, $O(N \log N)$ 계열은 3,000번 계산한다. log1000=3이기 때문이다. N이 더 커진다면 그 차이는 점점 더 벌어진다. 따라서 $O(N \log N)$ 계열과 비교해 보면 N^2-2N이나 $2N^2+3N$ 차이는 매우 작기 때문에 무시한다.

지금까지 배운 선택정렬, 버블정렬, 삽입정렬은 $O(N^2)$ 계열의 정렬 알고리즘이다. 정렬 알고리즘 중 구현은 쉽지만 성능은 좋지 않은 알고리즘이다. $O(N \log N)$ 계열의 정렬 알고리즘으로 힙정렬, 병합 정렬, 퀵 정렬 등이 있다.

3. 버킷정렬

정렬 알고리즘은 알고리즘 분야에서 오래된 주제로 많은 수의 알고리즘을 개발했다. 초기 $O(N^2)$ 계열의 정렬 알고리즘을 지나 $O(N \log N)$ 계열의 정렬 알고리즘을 많이 개발했다. 그러던 중 기존 상식에서 벗어난 정렬 알고리즘을 개발했는데, 바로 버킷정렬이 그것이다. 버킷정렬은 기수정렬이라고도 하는데, 지금까지 정렬 알고리즘에서 사용하던 비교 연산이 없다. 지금까지 밝힌 바에 따르면 다른 어떤 정렬 알고리즘보다 빠른 알고리즘이다. 또 정렬하는 방식도 간단하고 독창적이다.

어떻게 버킷정렬을 하는지 살펴보자. 우선 정렬할 숫자 6, 53, 12, 7, 75, 2, 89, 11이 있고, 0번부터 9번까지 버킷이 마련되어 있다.

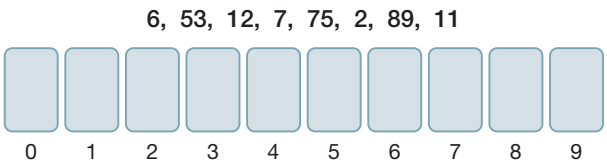

그림 30 버킷정렬의 초기 상태

처음 할 일은 각 숫자 1의 자리와 같은 버킷에 정렬할 데이터를 집어넣는 것이다. 이 작업을 마치면 [그림 31]과 같다.

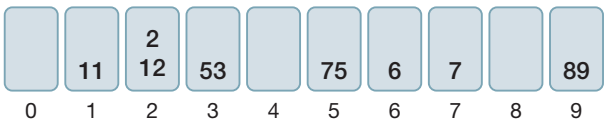

그림 31 버킷정렬의 첫 번째 상태

다음으로 버킷에 있는 숫자를 다시 뺀다. 주의할 점은 숫자를 뺄 때는 버킷의 앞에서 뒤로 가면서 숫자를 빼야 한다는 것이다. 같은 버킷에 숫자가 여러 개 있을 때는 아래 숫자부터 뺀다.

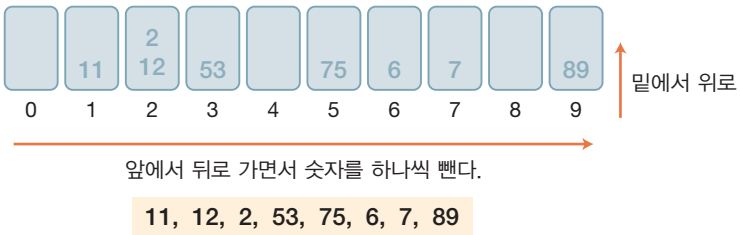

그림 32 버킷정렬의 두 번째 상태

뺀 숫자에서 이제 10의 자리를 가지고 앞의 두 방법을 반복한다. 즉, 10의 자리에 해당하는 버킷에 숫자를 집어넣고 다시 버킷에서 숫자를 뺀다. 앞서 강조했듯이 앞에서 뒤로, 아래에서 위로 숫자를 뺀다. 그러면 2, 6, 7, 11, 12, 53, 75, 89로 정렬이 완료된다.

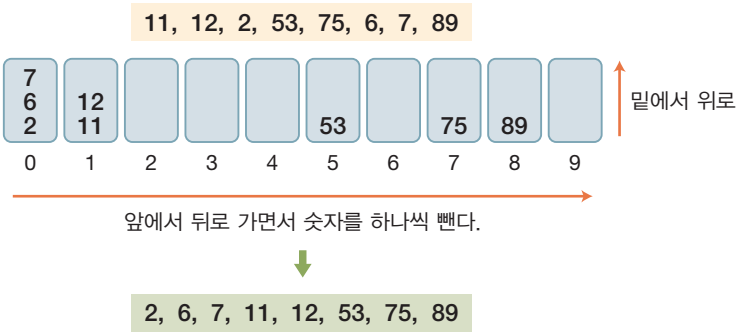

그림 33 버킷정렬의 세 번째 상태

버킷정렬에는 선택정렬, 버블정렬, 삽입정렬에서 하는 두 숫자를 비교하는 연산이 없다. 버킷에 숫자를 넣고 빼는 것만으로 정렬이 완료된다.

버킷정렬은 정렬하려는 숫자의 자릿수만큼 버킷에 넣었다 빼면 되기 때문에 매우 단순하고 쉽게 정렬할 수 있다는 장점이 있다. 버킷정렬의 단점이라면 공간의 복잡도가 올라간다는 것이다. 앞에서 배운 정렬들은 추가적인 공간이 필요 없지만, 버킷정렬은 버킷이라는 추가적인 공간이 필요하다. 따라서 공간의 복잡도는 나쁘다.

버킷정렬을 프로그래밍할 때는 2차원 리스트처럼 복잡한 자료 구조를 사용해야 하는데, 이는 책에서 다루는 프로그래밍 수준을 훨씬 뛰어넘기 때문에 생략한다. 사고의 전환이 좋은 결과를 가져올 수 있다는 것을 보여 주고자 버킷정렬 알고리즘을 소개했다. 많은 사람이 두 수를 비교하는 정렬 알고리즘을 고민하고 있을 때, 누군가는 전혀 새로운 방식으로 이 문제를 해결했다.

SECTION 05

LAN 토폴로지와 오류 코드

본문 연계: 10장

1. 통신의 분류

통신의 방향으로 통신 방식을 분류하면, 단방향 통신과 양방향 통신으로 나눌 수 있다. 단반향 통신은 심플렉스(simplex)라고 하는데, 한쪽 방향으로만 통신을 하는 방식을 가리킨다. 모스 부호 시스템, 라디오, TV가 단방향 통신에 해당한다.

양방향 통신은 듀플렉스(duplex) 혹은 풀듀플렉스(full-duplex)라고 하는데, 양쪽 방향에서 동시에 통신을 하는 방식을 가리킨다. 전화를 비롯한 대부분의 통신 방식이 양방향 통신에 해당한다. 또 단방향 통신과 양방향 통신의 중간적인 형태로 반양반향 통신이 있는데, 하프듀플렉스(half-duplex)라고 한다. 양방향 통신이기는 하지만, 한순간에는 한쪽 방향으로만 통신이 가능한 시스템이다. 무전기가 반양방향 통신의 대표적인 예다.

표 3 통신 방식에 따른 분류

분류	설명	예
단방향	한쪽 방향으로만 통신이 가능하다.	모스 부호, 방송
양방향	양쪽 방향으로 통신이 가능하다.	대부분 통신 시스템
반양반향	양방향 통신이지만, 한순간은 단방향 통신만 가능하다.	무전기

통신 대상에 따라 일대일 통신과 일대다 통신으로 구분할 수 있다. 일대일 통신의 예로는 전화나 무전기가 있다. 일대다(多) 통신은 불특정 다수와 통신하는 것으로, 브로드캐스팅이라고 한다. 라디오나 TV가 대표적인 예다. 일대다 통신 중 특정한 다수와 하는 통신은 멀티캐스팅(multi casting)이라고 한다. 예를 들어 회사 내 직원 전체에게 사내방송을 하면 브로드캐스팅이지만, 영업부에만 들리도록 방송을 한다면 멀티캐스팅이 된다.

2. LAN 토폴로지

앞서 많이 사용하는 LAN 구조로 스타형, 링형, 버스형이 있다고 설명했다. 이 중 가장 많이 사용하는 구조는 링형과 버스형이다. 강조했다시피 하드웨어적 연결만으로 네트워크는 구성되지 않으며, 프로토콜이 필요하다. LAN에서도 누가 먼저 데이터를 보낼지, 서로 간 충돌이 발생했을 때 어떻게 해결할지를 정하는 프로토콜이 필요하다.

LAN의 링형은 IBM이 개발했다. 링 구조에서 사용하는 프로토콜로 토큰 링(token ring)을 사용한다. IBM은 링 구조에 토큰 링 프로토콜을 사용하는 LAN을 FDDI(Fiber Distributed Data Interface)라고 명명했다. 버스형은 제록스(Xerox)에서 개발했고, 데이터 전송 프로토콜로 CSMA/CD(Carrier Sense Multiple Access with Collision Detect)를 사용한다. 제록스는 버스 구조에 CSMA/CD 프로토콜을 사용하는 LAN을 이더넷(Ethernet)이라고 명명했다.

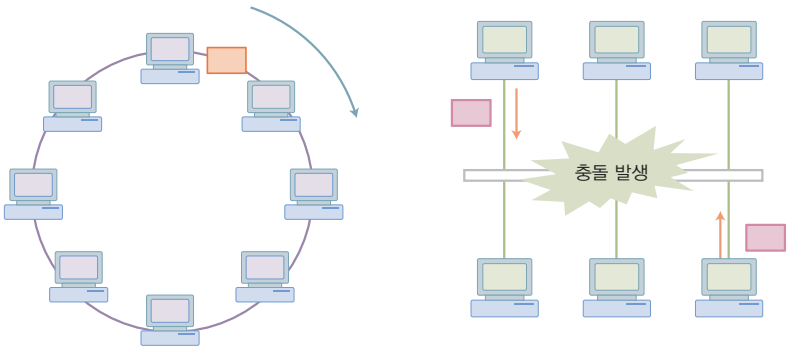

그림 34 FDDI(왼쪽)와 이더넷(오른쪽)

FDDI가 사용하는 토큰 링 프로토콜은 모든 컴퓨터가 공평하게 데이터를 보낼 수 있는 방식이다. 토큰 링 방식에서는 한쪽 방향으로 빈 패킷이 돌고 있다. 데이터를 보내려는 컴퓨터는 빈 패킷을 가져간 후 데이터를 싣고, 다시 한쪽 방향으로 돌린다. 데이터를 받으려는 컴퓨터가 패킷 내용물을 복사함으로써 데이터가 전송된다. FDDI는 빠르고 공평한 LAN 구조였지만, 원으로 선을 구성해야 하는 불편함이 있었다. 또 빈 패킷이 사라지거나 LAN에 참가하는 기기 중 하나라도 이상이 생기면 전체 네트워크가 먹통이 되었다.

이더넷에서 데이터를 누가 먼저 보낼지 결정하는 프로토콜인 CSMA/CD는 이름은 길지만 작동 방식은 매우 단순하다. 참여하는 모든 컴퓨터는 LAN선이 비었을 때 알아서 데이터를 보내면 된다. 그러다 보면 [그림 34]의 오른쪽과 같이 둘이 거의 동시에 보내어 충돌이 발생한다. 이때는 조금 쉬었다가 다시 보낸다. 한마디로 알아서 보내면 된다.

이더넷에서 여러 컴퓨터가 데이터를 보내고자 할 때, 공평성은 없다. 운이 좋은 컴퓨터가 더 많이 데이터를 보낼 수 있다. 이렇게 무식한 방식을 사용하는 이더넷이 현재 전 세계 LAN 시장의 90% 이상을 차지한다. 집이든 회사든 간에 네트워크 대부분은 이더넷을 사용한다. 이더넷이 LAN 시장을 장악할 수 있었던 것은 단순함 때문이다. FDDI는 공평성은 있지만, 빈 패킷이 사라지거나 중간에 컴퓨터가 고장 나면 네트워크가 작동하지 않는다. 따라서 지속적으로 관리를 해야 한다. 반면에 이더넷은 관리라고 할 것이 없다. 알아서 데이터를 보내면 된다. 구조가 단순하다 보니 LAN을 만들기도 쉽고 가격도 싸다. 또 LAN 공유기에 꽂기만 하면 특별한 설정 없이도 사용이 가능하다. 이러한 이유로 시장을 장악했다.

우리가 사용하는 통신기기는 대부분 표준을 따른다. 통신 표준 중 가장 유명한 것이 IEEE ('아이트리플E'로 읽는다) 규격이다. IEEE 802에 랜 표준이 상세히 기재되어 있는데, 그중 IEEE 802.11이 WiFi다. WiFi는 2.4GHz 대역을 사용하며, 요즘 나오는 제품은 5GHz 대역을 지원한다. 또 5Gbps 전송 속도를 지원하는 제품도 선보이고 있다.

3. 서브넷 마스크와 게이트웨이

앞서 인터넷 프로토콜인 IP를 설명하면서 IP 주소와 DNS 서버를 설명했다. IP 설정 대화상자에 보면 IP 주소와 DNS 서버 이외에 서브넷 마스크와 게이트웨이 설정이 있다. 이 설정값들이 의미하는 바를 살펴보자.

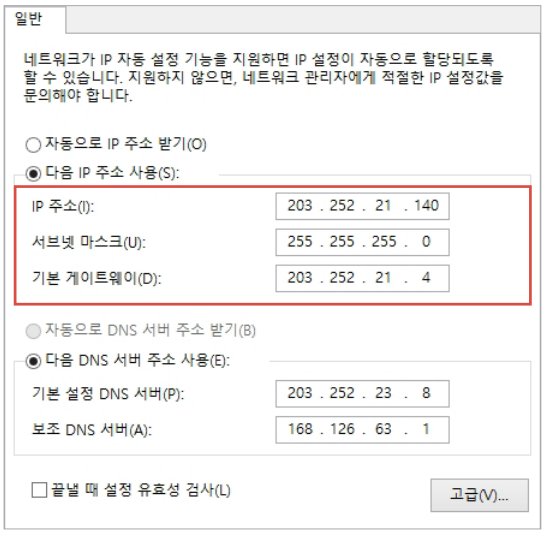

그림 35 IP 관련 설정 값

서브넷 마스크

앞서 IP 주소를 설명하면서 IP 주소에는 지역코드가 숨어 있다고 했다. 우편번호나 전화번호처럼 코드가 지역을 나타낸다는 의미는 아니다. 다만 사무실이나 실습실에 있는 컴퓨터의 IP 주소는 앞에서 세 자리까지는 같다는 의미다. 예를 들어 어느 사무실의 IP 주소가 228.255.75.46이라고 가정해 보자. 그 회사 내에 있는 컴퓨터의 앞쪽 세 자리는 모두 228.255.75로 같다. 228.255.75로 시작하는 주소를 가진 모든 패킷은 그 회사에 도착하게 된다. 회사 내 각 컴퓨터는 끝자리 1에서부터 255까지 개별번호를 가진다. 따라서 228.255.75.46은 228.255.75 네트워크에 있는 46번 컴퓨터라는 의미다. 이렇게 앞쪽 세 자리는 네트워크 주소를 가리키고, 나머지 한 자리는 개별 호스트 주소를 가리키는 IP 주소를 C 클래스 주소라고 한다.

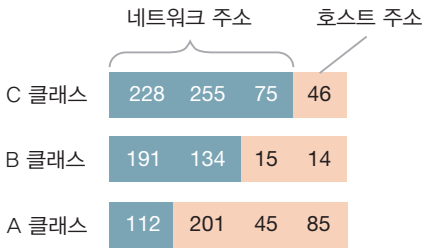

그림 36 IP 주소 클래스

주소 하나가 1바이트로 구성되기 때문에 C 클래스 주소는 이론적으로 컴퓨터 256(2^8)대를 가질 수 있다. 회사나 학교 내에 256대보다 많은 컴퓨터가 있다면 C 클래스 1개만으로는 부족하다. 이때는 C 클래스 주소를 여러 개 붙여서 사용한다. C 클래스를 5개 가진 회사의 네트워크 관리자가 있다고 하자. 관리자 입장에서 보면, 앞의 두 자리는 네트워크 주소고, 뒤의 두 자리는 호스트 주소다. 이렇게 앞의 두 자리는 네트워크 주소고 뒤에 두 자리는 호스트 주소인 클래스를 B 클래스 주소라고 한다.

한국 통신처럼 우리나라 전체의 네트워크를 관리하는 곳의 네트워크 관리자라고 생각해 보자. B 클래스는 2바이트가 호스트 주소이기 때문에 컴퓨터를 최대 6만 5,535(2^{16})대 가질 수 있다. 한국 통신이 관리하는 네트워크 숫자는 6만 대를 훨씬 뛰어넘는다. 이때는 앞의 한 자리는 네트워크 주소고 뒤의 세 자리는 호스트 주소인 A 클래스 주소를 사용한다.

A 클래스와 B 클래스는 네트워크를 관리하는 관리자 입장에서 보이는 주소다. 같은 사무실에 있는 컴퓨터는 C 클래스 주소를 사용하므로 사용자 입장에서는 언제나 C 클래스 주소만 보인다. C 클래스를 사용하는지 B 클래스를 사용하는지 사용자는 알 필요가 없다. 그러나 컴퓨터는 알아야 한다. 228.255.75로 시작하는 C 클래스 네트워크가 있다고 가정해 보자.

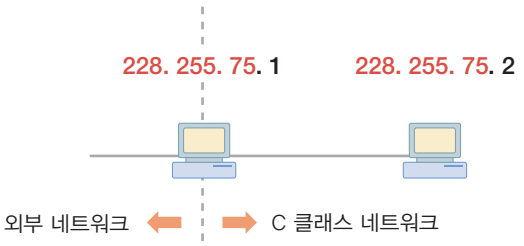

그림 37 내부 네트워크와 외부 네트워크

누군가 228.255.75.8로 패킷을 보냈다. C 클래스 주소와 앞 세 자리가 같기 때문에 이 패킷은 내부에서 처리해야 한다. 패킷 주소가 228.255.1.8이라고 하면 이 패킷은 외부 네트워크로 나가야 한다. 이렇게 어디까지가 네트워크 주소고 어디까지가 호스트 주소인지 알아야만 패킷을 외부로 내보낼지 말지를 결정할 수 있다. 이렇게 컴퓨터에 네트워크 주소와 호스트 주소가 어디까지인지 알려 주는 것이 서브넷 마스크(subnet mask)다.

IP 주소	228. 255. 75. 1
서브넷 마스크	255. 255. 255. 0

그림 38 C 클래스 서브넷 마스크

서브넷 마스크를 사용하는 방법은 의외로 간단하다. 네트워크 주소 자리에는 255, 호스트 주소 자리에는 0을 쓰면 된다. C 클래스 주소의 서브넷 마스크는 255.255.255.0, B 클래스 주소의 서브넷 마스크는 255.255.0.0, A 클래스 주소의 서브넷 마스크는 255.0.0.0이다. 반대로 서브넷 마스크를 보면 자신이 속한 주소 클래스도 알 수 있다.

게이트웨이

게이트웨이는 문이라는 의미로, 데이터가 지나다니는 통로라고 할 수 있다. [그림 37]에서 내부 네트워크와 외부 네트워크 중간에 있는 228.255.75.1이 게이트웨이 컴퓨터다. 게이트웨이는 외부 네트워크와 내부 네트워크의 경계선에 있는 컴퓨터 혹은 장치를 가리킨다. 게이트웨이 역할은 외부에서 들어온 패킷을 내부 컴퓨터에 나누어 주거나 내부 패킷을 외부로 내보낸다. 따라서 자신이 속한 네트워크 게이트웨이를 지정해야만 데이터를 외부로 내보내거나 외부에서 온 데이터를 받을 수 있다.

4. 포트와 소켓

앞서 전송 제어 프로토콜(TCP)에서 사용하는 주소를 포트라고 설명했다. TCP를 이용한 네트워크 프로그램을 소켓 프로그램이라고 한다. 포트와 소켓 사이의 연관 관계를 알려면 클라이언트/서버 구조를 확실하게 이해할 필요가 있다. 혹시 누군가 "서버 한 대 사와!"라고 말했다면, 이것은 서버를 잘못 이해하고 있는 것이다. 서버는 특정 하드웨어를 가리키는 단어가 아니다. 클라이언트/서버 구조를 이해하기 위해 먼저 웹이 어떻게 작동하는지 살펴보자.

웹 시스템에서는 클라이언트 컴퓨터에서 웹 브라우저를 사용하여 인터넷에 접속한다. 우리가 사용하는 인터넷 익스플로러, 크롬, 파이어폭스는 웹 브라우저를 구현한 소프트웨어 이름이다. 이러한 소프트웨어 전체를 표현하는 단어가 웹 브라우저다. 웹 브라우저 역할은 서버에 웹 페이지를 달라고 요청하여 이를 사용자에게 보여 주는 것이다. 이때 웹 브라우저가 사용하는 프로토콜은 하이퍼텍스트(웹 페이지)를 보내 달라는 프로토콜이다. 영어로 'Hyper Text Transfer Protocol'이며, 약어로는 HTTP다. 서버가 클라이언트에서 요청을 받으면, 웹 페이지(HTML)를 클라이언트에 전송시켜 준다. 이렇게 받은 웹 페이지를 웹 브라우저가 보여 준다.

Tip 구경을 하다는 뜻인 영어 단어는 브라우징(browsing)이다. 그래서 웹 브라우저라고 한다.

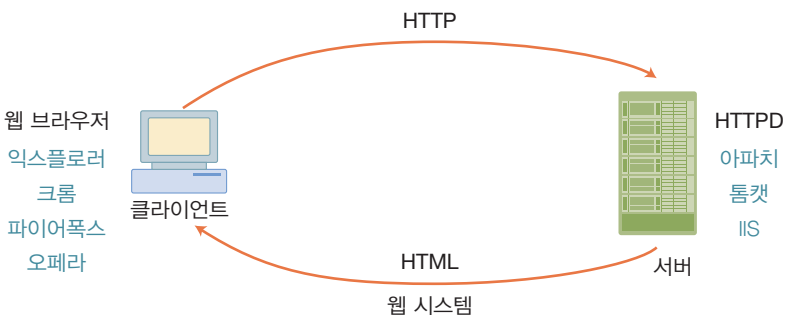

그림 39 HTTP

웹 브라우저처럼 클라이언트 소프트웨어는 사용자가 필요할 때 사용했다가 필요 없어지면 종료시킨다. 그러나 서버는 입장이 다르다. 언제 서비스를 요청할지 모르기 때문에 항상 클라이언트를 기다려야 한다. 이렇게 죽지 않고 계속 살아 있는 프로세스를 데몬(daemon)이라고 한다. 그래서 웹 시스템의 서버 쪽에는 웹 서비스를 받아 줄 데몬이 설치되어 있어야 한다. 서버에서 24시간 죽지 않고 살아서 웹 페이지를 전송해 주는 프로세스를 HTTPD(Hyper Text Transfer Protocol Daemon: 웹 데몬)라고 한다. 데몬 이름은 프로토콜 이름 뒤에 D만 붙이면 된다.

서버는 특정 하드웨어를 지칭하는 단어가 아니라 특정 소프트웨어가 깔린 컴퓨터를 가리킨다. 서버는 '데몬이 깔려 있는 컴퓨터'라고 정의할 수 있다. 우리는 HTTPD가 깔린 컴퓨터를 웹 서버라고 한다. 웹 브라우저와 마찬가지로 HTTPD는 웹 데몬 전체를 지칭하는 단어며, HTTPD를 구현한 제품으로 아파치, 톰캣, IIS 등이 있다. 따라서 아파치, 톰캣, IIS 등의 소프트웨어를 설치하는 것은 HTTPD로 웹 서버를 구축하는 것이다.

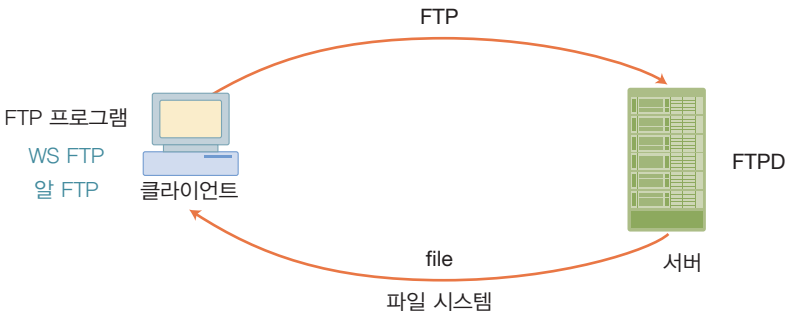

그림 40 FTP

웹 파일 시스템도 마찬가지다. 클라이언트 컴퓨터는 FTP를 사용하여 파일 서버에 파일을 요청한다. 클라이언트 요청을 처리하고 싶을 때는 컴퓨터에 FTPD를 설치하면 파일 서버가 된다. 참고로 이메일 시스템도 유사한데, 이메일을 주고받는 데 사용하는 프로토콜 이름은 SMTP(Simple Mail Transfer Protocol)다. 따라서 이메일 데몬은 SMTPD이며, SMTPD를 설치한 컴퓨터는 이메일 서버가 된다.

이렇듯 서버는 특정 컴퓨터를 가리키는 단어가 아니라 데몬이 설치된 컴퓨터를 지칭하는 단어다. 물론 서버 역할을 하려면 좋은 CPU와 24시간 안정적으로 작동하는 파워가 필요하다. 그래서 고사양 컴퓨터를 '서버급 컴퓨터'라고 한다.

클라이언트 컴퓨터로 돌아가 보자. 크롬 웹 브라우저를 여러 개 열어 네이버, 구글, 다음을 방문한다고 하자. 클라이언트에 네이버, 구글, 다음에서 보내온 패킷이 도착했다. 모든 패킷에는 클라이언트의 IP 주소인 10.10.10.10이 적혀 있다. 이 경우 패킷을 웹 브라우저에 전달하는 데 사용하는 TCP 주소가 포트번호다.

앞서 포트번호를 설명하면서 IP 주소는 아파트 동번호고, 같은 동 내의 포트번호는 호수번호라고 설명했다. 이를 위해 운영체제는 네트워크를 사용하는 모든 소프트웨어에 임시로 포트번호를 나누어 준다.

서버 쪽에 있는 데몬도 네트워크 프로그램이기 때문에 포트번호를 사용하지만, 클라이언트의 포트번호와는 조금 다르다. 클라이언트 포트번호는 운영체제에서 임시로 받는 번호이기 때문에 실행할 때마다 번호가 바뀐다. 데몬도 실행할 때마다 번호가 바뀐다면 클라이언트가 데몬 위치를 찾을 방법이 없다. 오늘은 103호에 살다가 내일은 307호에 살고 있다면 어떻게 찾아가겠는가? 그래서 데몬에는 미리 정해진 포트번호를 부여한다. 웹 데몬의 포트번호는 80번, FTP 데몬의 포트번호는 21번으로 하기로 정했다. 이렇게 미리 정한 포트번호를 잘 알려진 포트번호라고 한다.

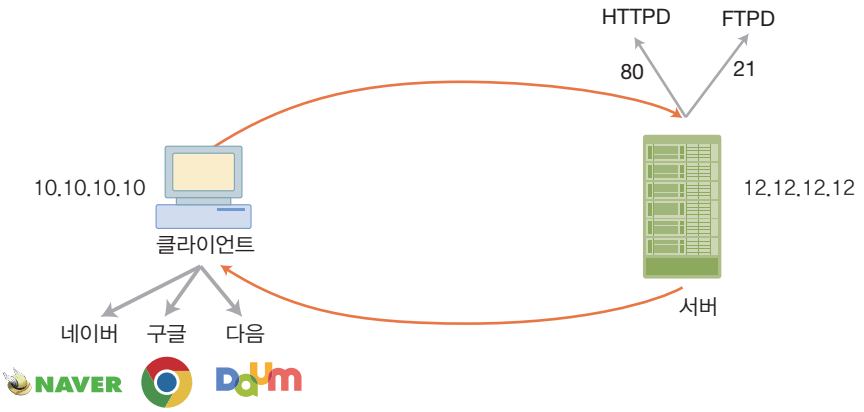

그림 41 포트번호와 잘 알려진 포트번호

네이버에는 하루 수만 명이 몰려드는데, 모두 http://naver.com:80에 방문한다. 네이버 서버 입장에서 보자. 포트번호 80에 수만 명이 몰려왔다. 이들에게 어떻게 서비스를 해 줄 것인가? 그 해답은 전기에 있다. 벽에는 220V 콘센트가 하나뿐이고 노트북, 스마트폰, 블루투스 이어폰을 동시에 충전해야 할 때, 멀티탭을 사용한다. 마찬가지다. 80번 포트에 멀티탭을 꽂은 후 몰려드는 사용자에게 구멍을 하나씩 나누어 주면 된다. 이때 사용자가 접속하는 구멍 하나를 소켓이라고 한다. 소켓은 같은 포트에 연결되어 여러 명을 동시에 처리할 수 있는 소프트웨어적 접속 장치다.

그림 42 멀티탭과 소켓

5. 오류 코드

앞서 CRC라는 오류 탐색 코드를 설명했다. 오류 탐색 코드는 내용물 중에 원래 값이 아닌 것을 찾아낼(탐색할) 수 있도록 만든 특수한 코드다. 가장 간단한 오류 탐색 코드로 패리티 비트(parity bit)가 있다. 패리티 비트 방식은 보내려는 데이터의 추가 비트를 두어 1의 개수가 짝수 혹은 홀수가 되도록 만드는 것이다. 1의 개수가 짝수 개인 것을 짝수 패리티 비트(even parity bit), 홀수 개인 것을 홀수 패리티 비트(odd parity bit)라고 한다. [그림 43]은 짝수 패리트 비트의 예다.

그림 43 짝수 패리티 비트

보내려는 데이터에는 1의 개수가 4개다. 그러므로 맨 마지막 패리티 비트에 0을 넣어 보낸다. 받은 사람은 패리티 비트까지 포함하여 1의 개수가 짝수면 이상이 없다고 생각한다. 패리티 비트 방식은 간단하지만, 변형이 짝수 개 발생하면 오류를 찾지 못한다. 예를 들어 [그림 43]과 같이 보내려는 데이터 중 2개가 동시에 바뀌면 1의 개수가 여전히 짝수이기 때문에 오류를 발견하지 못한다.

그래서 패리티 비트보다 강력하면서도 오버헤드가 적은 오류 검색 코드를 개발했는데, 그것이 앞서 설명한 CRC(순환 중복 검사) 코드다. 패리티 비트와 비교해 보면, 패리티 비트는 오류를 1개 찾기 위해 추가 비트가 1개 필요하다. 따라서 오류를 6만 5,535개 잡기 위해서는 추가 비트가 6만 5,535개 필요하다. 앞서 설명했듯이, CRC-16은 0을 제외한 모든 나머지 값이 오류다. 따라서 찾을 수 있는 오류 개수는 $2^{16}-1$개다. 2^{16}은 6만 5,536이므로 4바이트로 찾을 수 있는 오류 개수는 6만 5,535($2^{16}-1$)개나 된다. 따라서 CRC-16은 추가되는 16비트만으로도 오류를 6만 5,535개 찾을 수 있어 효율성이 높다.

오류코드에는 오류 탐색 코드 외에 오류 정정 코드(Error Correction Code, ECC)도 있다. 오류 탐색 코드는 오류를 찾아내지만, 오류 정정 코드는 오류를 찾을 뿐 아니라 원래 값으로 정정도 할 수 있다. 대표적인 오류 정정 코드로 허밍코드(humming code)가 있다. 허밍코드는 보내려는 데이터를 홀수 배로 늘려서 보내는 방식이다.

그림 44 허밍코드

허밍코드에서 0을 보낼 때는 000을 보내고, 1을 보낼 때는 111을 보낸다. 오류가 1개 발생할 때는 오류 정정이 가능하다. 001은 0으로 바꾸고, 011은 1로 바꾸면 된다. [그림 44]와 같이 오류가 2개 발생할 때는 원래 값을 찾을 수 없다. 오류를 2개 보정하려면 원래 값의 5배를 만들어 보내야 한다. 0을 보낼 때 00000을 보내면 오류를 2개까지 보정할 수 있다.

허밍코드는 오류를 보정할 수 있어 좋지만, 추가되는 비트가 너무 많다는 단점이 있다. 오류를 1개 보정하려면 원래 값을 3배로 크게 만들어 보내야만 한다. 이렇게 오버헤드가 크기 때문에 잘 사용하지 않는다.

6. 웹 관련 프로그래밍 언어

웹 데몬(HTTPD)의 역할은 HTML 형태로 미리 만든 웹 페이지를 클라이언트에 전달하는 것이다. 웹 페이지는 책처럼 미리 만들어진 정적 데이터다. 최초의 웹 시스템은 자신이 연구하는 내용을 다른 사람에게 알려 주려고 만들었기 때문에 시간처럼 동적으로 변하는 데이터가 필요 없었다. 현재 시간처럼 계속 변화하는 데이터를 웹 페이지에 표시하려면 절차가 조금 복잡하다. 웹 페이지에 현재 시간을 나타내려면 웹 데몬이 운영체제에 시간을 물어본다. 운영체제가 여기에 응답하여 현재 시간을 알려 주면 이를 문자로 만들어 웹 페이지에 끼워 넣은 후 전송한다. 이때 웹 데몬이 운영체제나 다른 프로세스에 데이터를 요청할 때 사용하던 프로그램을 CGI(Common Gateway Interface)라고 불렀다. 복잡한 프로그램이 아니기 때문에 보통 C 언어로 작성했다.

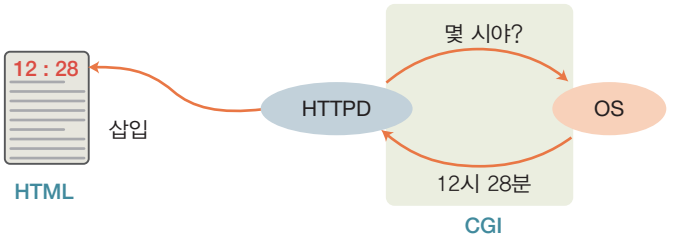

그림 45 웹 데몬과 CGI

현대에 와서는 동적인 데이터가 넘쳐 나기 시작했다. 웹 페이지에서 로그인과 패스워드는 데이터베이스가 가지고 있다. 따라서 데이터베이스에 질문을 해야 로그인을 할 수 있다. 더 나아가 쇼핑 사이트에서 물품을 보여 주고, 장바구니에 담고, 결제를 하는 일 모두 CGI가 처리해야 한다. 이렇듯 동적으로 처리해야 하는 작업이 늘어남에 따라 웹 데몬이 CGI를 쉽게 처리할 수 있는 프로그래밍 언어를 제공하기 시작했다.

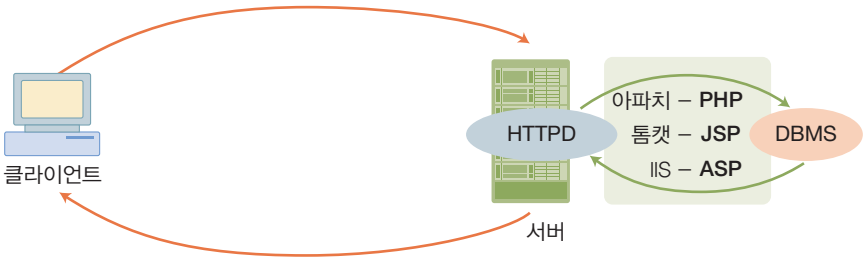

그림 46 웹 데몬과 프로그래밍 언어

가장 많이 사용하는 웹 데몬은 아파치다. 아파치가 제공하는 웹 페이지용 프로그래밍 언어를 PHP(Hypertext Preprocessor PHP)라고 한다. PHP로 작성된 코드를 HTML 소스 문서 안에 넣으면, 데몬이 이를 실행하여 결과를 웹 페이지에 자동으로 삽입한다. 자바를 만든 썬(SUN)에서 제공하는 웹 데몬이 톰캣(Tomcat)이고, 톰캣에서 제공하는 CGI가 JSP(Java Server Pages)다. JSP는 자바를 기초로 만든 스크립트 언어다. 마이크로소프트에서 만든 웹 데몬은 IIS(Internet Information Services)인데, 여기서 사용하는 언어가 ASP(Active Server Pages)다.

여러분이 웹 프로그래머가 되고 싶다면 아파치, 톰캣, IIS 중 하나를 설치하는 방법을 배워야 한다. 이후에 자신이 설치해 보고, 잘 알고 있는 웹 데몬의 프로그래밍 언어를 배우면 된다.