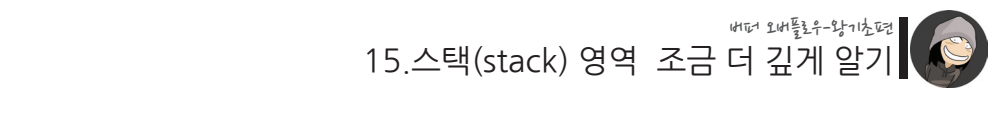


Section 15

스택(Stack) 영역 조금 더 깊게 알기

메모리 맵 중에서도 특히 스택 영역은 버퍼 오버플로우 공격에 있어서 매우 중요한 부분입니다. 앞서 말씀드린 바대로 스택 영역엔 함수 호출과 관련된 정보들, 그 중에서도 특히 리턴 어드레스가 저장되기 때문입니다. 이번 시간엔 스택의 개념과 특징, 그리고 용도에 대해 알아보겠습니다.

스택이란 "데이터 구조"라는 컴퓨터 분야에 나오는 개념으로서, 메모리의 데이터들을 효율적으로 다루기 위해 고안된 데이터 참조 방식 중 하나입니다.

스택(stack)이라는 단어는 차곡 차곡 쌓여진 더미를 의미하는데, 이는 가장 먼저 입력된 데이터가 가장 아래쪽에 쌓이고, 나중에 입력된 데이터는 그 위에 쌓이게 된다는 구조상의 특징이 마치 더미를 쌓아 올린 모습과 흡사하기 때문입니다.

15.스택(stack) 영역 조금 더 깊게 알기

스택의 기본 개념은 “가장 먼저 처리해야 할 것을 가장 가까운 곳에 둔다”인데요, 위 그림처럼 차곡 차곡 쌓아올려진 데이터가 실제 사용될 때에는 가장 위의 것부터 반대 순서로 사용되어 나갑니다.

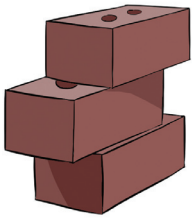
이와 같은 형태의 구조를 쉽게 이해하기 위해선 택시 기사분들께서 사용하시는 동전통을 연상하시면 됩니다.

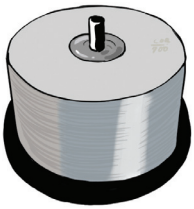
위 동전통이 그 구조상 가장 마지막에 올려진 동전이 가장 먼저 빼내어지게 되어 있는 것처럼, 스택 역시 가장 마지막에 추가된 데이터가 가장 먼저 빼내어져서 사용되고, 가장 최초로 넣었던 데이터는 가장 마지막에 빼내어지게 됩니다.

15.스택(stack) 영역 조금 더 깊게 알기

이와 비슷하게 층층이 쌓아 올려진 연탄이라던지,

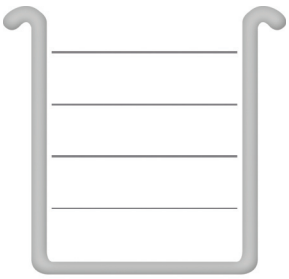
혹은 쌓인 벽돌, 공시디 통, 혹은 맛난 과자인 플링구스 등 역시 스택 구조를 연상시키는 좋은 예입니다.





이처럼 스택은 가장 나중에 들어온 자료가 가장 먼저 나가게 되기 때문에 LIFO(Last-In, First-Out), 우리말로는 "후입선출형 구조"라고 합니다. 간혹 이를 FILO(First-In, Last-Out)형 구조라고도 하는데, 먼저 들어온 것이 나중에 나간다는 결국 동일한 의미이기는 하지만 정식 용어로는 LIFO가 맞습니다.

그럼 이제부터 스택 영역에 새로운 데이터가 추가되고 사용되는 모습의 예를 살펴보겠습니다. 다음과 같은 모양의 빈 통이 스택 영역이라고 생각하면서 진행해 봅시다.

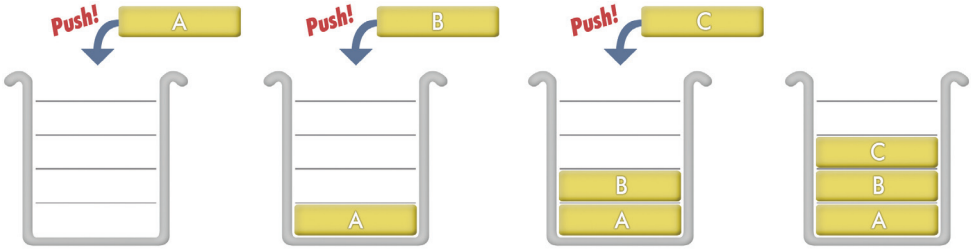


스택(STACK)

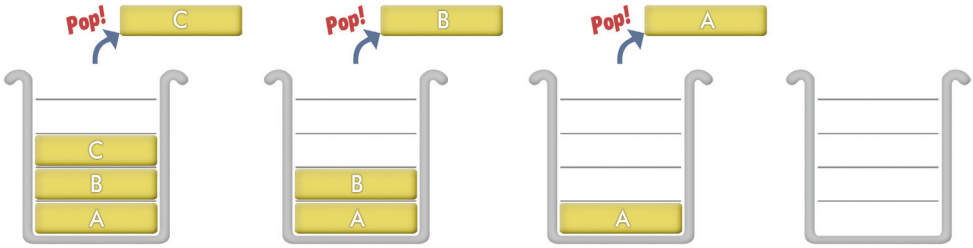
15.스택(stack) 영역 조금 더 깊게 알기

PUSH(푸쉬)와 POP(팝)

스택에 새로운 자료를 추가하는 것을 컴퓨터 용어로 PUSH라고 부릅니다.
다음과 같이 PUSH를 하면 할 수록 기존 데이터 위에 새로운 데이터가 순서대로 쌓아 올려지게 됩니다.



반면에 이렇게 PUSH에 의해 스택에 저장된 값을 다시 빼내어 내는 것은 POP이라고 부릅니다.

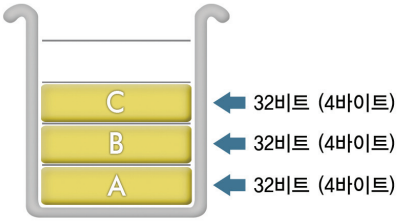


이처럼 스택은 기본적으로 PUSH와 POP이라는 두 개의 명령으로만 데이터를 추가하거나 제거할 수 있으며, 이 때 PUSH 혹은 POP되는 데이터의 크기는 스택을 구현하는 프로그래머가 마음대로 정할 수 있습니다.

그리고 OS에 기본으로 구현되어 있는 스택을 “시스템 스택”이라고 하는데, 우리가 메모리 맵에서 본 그 스택이 바로 시스템 스택입니다. 이 시스템 스택의 기본 데이터 크기는 프로그램의 레지스터 크기와 일치합니다.

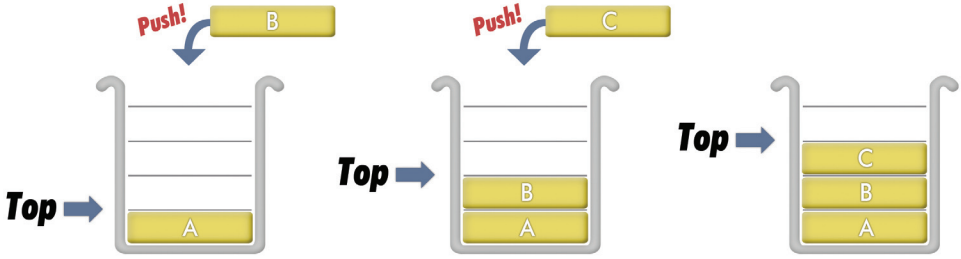
즉, 우리는 현재 32bit 프로그램을 기준으로 하고 있기 때문에 시스템 스택 데이터의 기본 크기는 32비트(4바이트)인 것입니다. 앞으로 이 시스템 스택 안에 함수와 관련된 각종 정보들이 PUSH되거나 POP될 것입니다.

시스템 스택데이터의 크기 == 프로그램 레지스터의 크기



TOP(탑)과 BOTTOM(바텀)

TOP과 BOTTOM은 스택의 특정 위치를 가리키는 용어들입니다.
먼저 TOP이란, 단어 자체에서 나타나는 바대로 현재 스택에 쌓인 데이터들의 위치 중 가장 높은 위치의 메모리 주소 값**(즉, 실제로는 가장 낮은 주소 값)**을 가리키는 용어입니다.



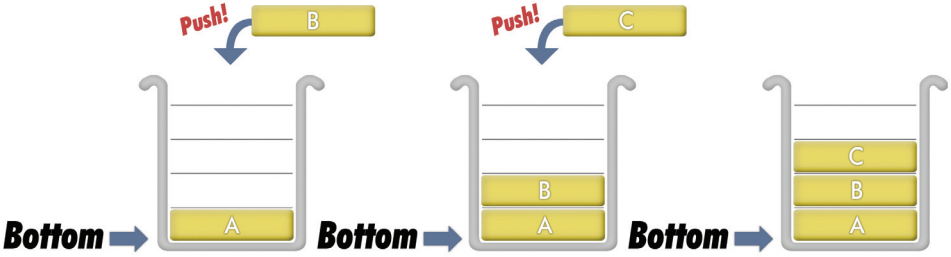
이처럼 스택에 쌓이는 데이터 양의 변화에 따라 TOP의 위치는 계속해서 변하게 됩니다. 새 데이터가 추가되면 스택 내의 TOP의 위치는 높아지고, 반대로 데이터가 제거되면 TOP의 위치는 다시 낮아집니다. TOP의 용도는 스택에 데이터가 추가되거나 제거될 때에 기준이 되는 위치를 정하는 것입니다.

15.스택(stack) 영역 조금 더 깊게 알기

즉, 스택에 PUSH가 이루어질 때엔 현재 TOP에 해당하는 위치를 기준으로 새로운 데이터가 추가되고, TOP의 위치는 4바이트만큼 높아집니다. 반대로 POP이 이루어질 때엔 현재 TOP에서 4바이트 크기의 데이터를 빼내어 오며, TOP의 위치는 다시 4바이트만큼 낮아집니다.

이처럼 TOP은 스택에서 데이터가 추가되거나 제거되는 위치를 "가리키고" 있기 때문에, Stack Pointer라고 불리기도 합니다.

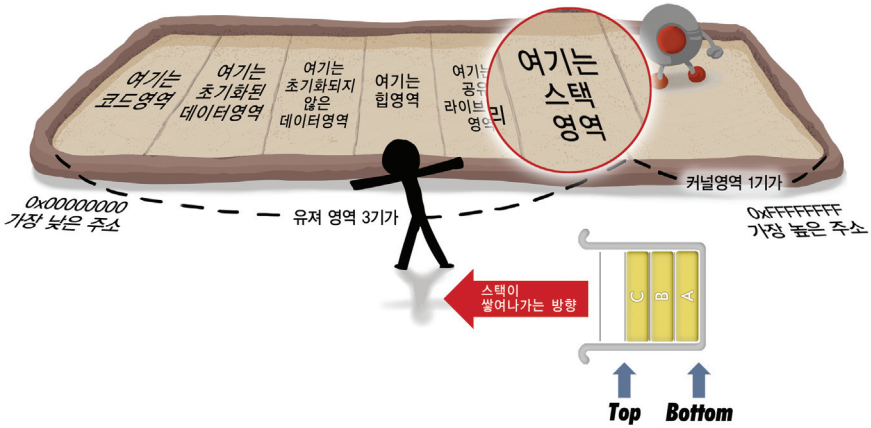
다음으로 BOTTOM은 스택에서 가장 아랫 부분에 해당하는 메모리 주소 값**(즉, 실제로는 가장 높은 주소 값)**을 가리킬 때 사용하는 용어입니다.



BOTTOM은 TOP과는 달리 스택의 가장 아랫 부분만을 가리키기 때문에 항상 동일한 값이 유지됩니다. 그리고 만약 TOP 값이 BOTTOM 값과 같아진다면, 스택의 가장 아랫 부분에 도달했기 때문에 더 이상의 저장된 자료가 없음을 의미합니다. 이 때엔 더 이상의 POP 명령을 수행 할 수 없습니다.

그렇다면 메모리 맵을 기준으로 볼 때 스택의 TOP과 BOTTOM은 각각 어느 방향이 될까요?

15.스택(stack) 영역 조금 더 깊게 알기



이처럼 TOP은 메모리 맵의 낮은 주소쪽인 왼쪽, BOTTOM은 높은 주소쪽인 오른쪽에 해당합니다.
언뜻 생각하기엔 스택이 점점 쌓여 나갈 수록 메모리 주소 값도 커질 것이며, 그렇기 때문에 TOP이 오른쪽, BOTTOM이 왼쪽에 해당할 것만 같지만 실제로는 그 반대인 것입니다.

그렇기 때문에 스택에 새로운 데이터가 추가될 수록 TOP(Stack Pointer)에 해당하는 메모리 주소 값은 반대로 점점 작아지게 되며, 스택의 BOTTOM은 항상 커널과 맞닿는 부분에 해당하는 0xc0000000 주소 값이 됩니다.

그런데 왜 스택이 자라날 수록 메모리 주소 값은 반대로 작아지도록 설계를 해놨을까요? 스택이 커질 수록 메모리 주소 값도 함께 커진다면 스택의 구조를 기억하기가 더 쉬웠을텐데 말입니다.

이는 두 가지 이유로 설명할 수 있는데, 첫째는 스택이 항상 커널의 반대 방향으로 자라기 때문에 영원히 커널을 만나지 않게 됩니다. 다시말해, 스택이 아무리 커져도 접근 불가 영역인 커널을 건드리지 않게 됩니다.

둘째 이유는 힙과 관련이 있습니다. 힙 영역은 스택과는 달리 새로운 데이터가 추가될 수록 더 큰 메모리 주소를 할당받게 됩니다. 이처럼 스택 영역과 힙 영역이 공유 라이

15.스택(stack) 영역 조금 더 깊게 알기

브러리 영역을 가운데에 두고 서로 마주보는 형태를 갖기 때문에 메모리 공간을 알뜰하게 활용 할 수 있게 됩니다.