파이썬 GIL 깊숙히! (上)

[November 12, 2006](http://highthroughput.org/wp/cb-1136/) by [hyeshik](http://highthroughput.org/wp/author/hyeshik/" \o "View all posts by hyeshik)

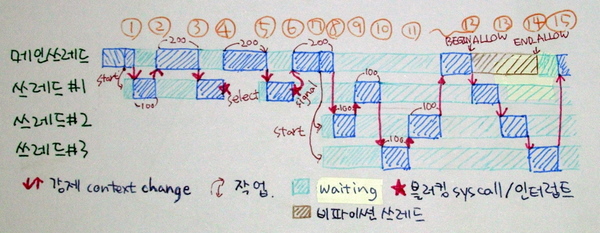
대표적인 파이썬 구현인 [CPython](http://www.python.org/download/)에서 거의 십년여 지속적으로 문제로 두루 지적 받아 왔지만 여전히 그대로 있는 것으로 바로 GIL (Global Interpreter Lock)이 있습니다. 점점 CPU 여러 개를 동시에 쓰는 시스템들이 많아지면서 특히 요즘 자주 언급되고 있는데, GIL은 짧게 말해 동시에 CPU를 1개 밖에 못 쓴다는 그 문제의 근원입니다.

**역사적 배경과 존재의 이유**

GIL은 처음 파이썬에 쓰레드 구현이 들어갈 때 같이 생긴 이후로, 지금까지 계속 내려오고 있습니다. 파이썬 구현은 상당 부분이 라이브러리 전역 변수에 의존하고 있고, 여기 저기 객체 구조에 따라서 참조를 따라서 마구 접근을 하기 때문에, 동시에 쓰레드가 여러 개 돌아갈 때 심각한 문제가 발생할 수 있습니다. 따라서, 파이썬은 초창기부터 파이썬 라이브러리 코드가 돌아가는 중에는 전역적인 락(GIL)을 걸어서 동시에는 절대로 같이 못 돌아가도록 해서 해결하였습니다.

물론 이 문제를 해결하기 위해  [1996년과 2000년에 Greg Stein이 GIL을 완전히 없애고 섬세한 락(fine-grained lock)을 구현](http://mail.python.org/pipermail/python-dev/2000-April/003605.html)하였지만, 그 결과 SMP에서는 1.6배 속도가 빨라졌지만, 대부분의 사용자가 쓰는 UP에서는 엄청나게 느려져서 실제적으로 대부분 사용자들은 불평을 할 수 밖에 없었고, 그로 인해서 모듈을 만들기가 쉽지 않은 문제들과 복잡성이 증가해서, 다른 개발자들이 선뜻 유지보수 가능한 코드 개발에 나서지 않아서 그냥 GIL이 지금까지 남아있게 되었습니다.

**GIL은 대략 어떤가?**



위 그림에서는 메인 쓰레드와 거기서 띄운 쓰레드 3개가 같이 돌아가는 모습을 그려 놓았습니다. (못 생겨도 이해 바랍니다. ^^;) 파란색 상자가 돌아가는 상태에 있는 것을 표시한 것인데, 대충 눈에 보이듯이 동시에는 파이썬 쓰레드는 1개만 돌 수 있습니다. 여기서 파이썬 쓰레드라 함은, 파이썬 인터프리터가 호출하는 모든 코드를 뜻합니다. 즉, C코드와 파이썬 코드의 구분 없이 일단은 모두 파이썬 쓰레드처럼 동시에 1개만 돌아가게 GIL의 영향을 받습니다. 여기에 예외도 있는데 이것은 후에 설명하겠습니다.

**쓰레드 실행이 바뀌는 시점은 언제일까?**

파이썬은 아시다시피 내부적으로 바이트코드로 번역된 다음에 실행됩니다. 예를 들어 보면 다음과 같이 바이트 코드를 볼 수 있습니다.

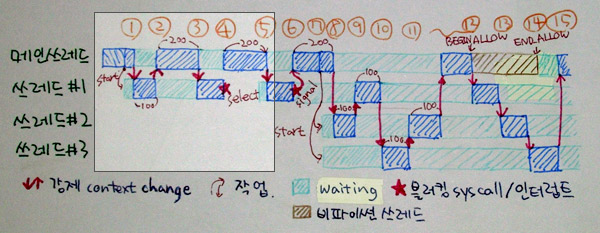
Python



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | &gt;&gt;&gt; import dis  &gt;&gt;&gt; dis.dis(lambda x: x\*x[3]\*y\*3)    1           0 LOAD\_FAST                0 (x)                3 LOAD\_FAST                0 (x)                6 LOAD\_CONST               1 (3)                9 BINARY\_SUBSCR               10 BINARY\_MULTIPLY               11 LOAD\_GLOBAL              1 (y)               14 BINARY\_MULTIPLY               15 LOAD\_CONST               1 (3)               18 BINARY\_MULTIPLY               19 RETURN\_VALUE |

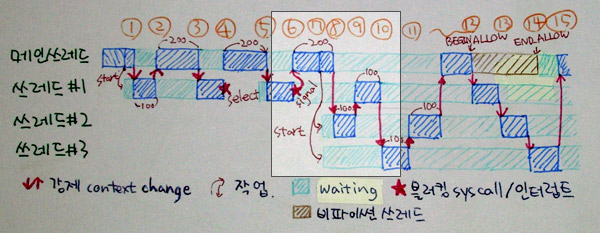
평소에는 GIL을 꽉 쥐고 있다가, 각 줄을 파이썬 인터프리터에서 처리하면서 내부적으로 카운터를 1씩 증가시킵니다. 그러다가 100이 되면 GIL을 풀었다가 다시 쥐게 되는데, 풀었다 쥐는 사이 공간에서 문맥 전환(context change)가 일어납니다. 즉, 파이썬 인터프리터가 쓰레드가 1개만 뜨는 것이 아니라, 인터프리터도 쓰레드마다 도는데, 서로 GIL 때문에 동작은 1개씩만 하는 상황이죠. 그래서 그림을 보시면 자발적이 아니라 틱이 100이 돼서 전환되는 쓰레드들은 너비가 100의 배수로 되어 있습니다. 100이 어느 정도 범위인가 하면, 보통 쓰는 정도의 파이썬 코드 20~30줄 내외가 바이트코드 100줄 정도 되기 때문에, 그렇게 크지도 않고 작지도 않은 단위라고 볼 수 있겠습니다.

**특별한 사건이란?**



위의 그림 ①번에서 보듯이, 심지어 새로운 쓰레드를 시작할 때에도 즉각적으로 새 쓰레드가 실행되지는 않습니다. 기다리고 있다가, 원래 돌던 쓰레드의 카운터가 100이 돼야 돌아가게 됩니다. 그런데, 100개 단위로 일어나는 것이 아니라 “특별한 사건”이 있을 때는, 갑자기 100으로 퍽~ 뛰어서 문맥 전환이 일어납니다. 그림의 ②이나 ③은 틱에 의해서 전환된 것이지만, ④번 같이 블러킹 시스템콜이 일어나게 되면 [select(2)](http://www.freebsd.org/cgi/man.cgi?query=select&apropos=0&sektion=0&format=html)하는 동안에 락을 걸고 있으면 다 멈추니까, 다른 쓰레드로 넘겨줍니다.

**쓰레드가 어떤 순서로 실행되나?**



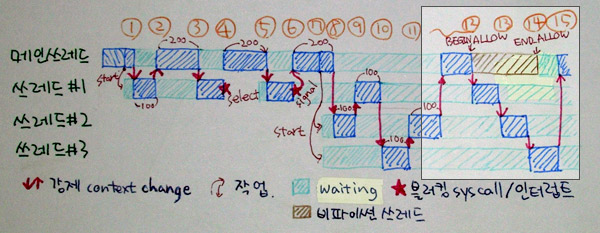
위 그림의 ⑥번도 마찬가지로 시그널이 발생하면 일단 시그널을 큐에 쌓아둔 다음에 틱을 100으로 증가시켜서 즉각적으로 다른 곳으로 넘겨버립니다. (그래서 이렇게 시그널 핸들러가 직접 호출되지 않는 경우가 있기 때문에,  
시그널 핸들러가 희한하게 꼬인 상황에서 엉뚱한 예외가 발생한다거나, 시그널 때문에 예외가 씹힌다거나, 순서가 뒤죽박죽이 되는 일이 일어납니다.)

그런데, 쓰레드 수가 많을 때 실행되는 순서는 어떻게 될까요?  
뭔가 내부적으로 정해진 것이 있으면 좋지만, 사실 파이썬에서는 시스템의  
쓰레딩 라이브러리에 전적으로 맡깁니다. 즉, POSIX 시스템에서는  
[sem\_post(3)](http://www.freebsd.org/cgi/man.cgi?query=sem_post&apropos=0&sektion=0&format=html)/[sem\_wait(3)](http://www.freebsd.org/cgi/man.cgi?query=sem_wait&sektion=3&apropos=0) 또는  
[pthread\_mutex\_unlock(3)](http://www.freebsd.org/cgi/man.cgi?query=pthread_mutex_unlock&sektion=3&apropos=0)/[pthread\_mutex\_lock(3)](http://www.freebsd.org/cgi/man.cgi?query=pthread_mutex_lock&apropos=0&sektion=3&format=html) 쌍을 순서대로 호출해서, 그냥 아무데나 가도록 합니다.

그런데, 이게 쓰레드 구현마다 다르고 쓰레드의 종류마다 달라서, 일정한 동작이 있지 않습니다. 일반적으로 시스템 스코프 쓰레드들은 lock/unlock을 확인하는데 커널을 거쳐야하기 때문에, 굉장히 자주 바뀌고 순서가 아주 제멋대로인 반면에,  
프로세스 스코프 쓰레드들은 lock/unlock을 프로세스 내부에서 처리해버리기 때문에, 좀처럼 잘 바뀌지 않고, 순서도 거의 일정합니다. 심지어 쓰레드 50개를 만들 동안에 50개 모두 전혀 실행 안 되고 버티는 경우까지도 종종 있을 정도입니다.

따라서, 쓰레드의 실행 순서나 실행의 정도에 의존하는 프로그래밍은 쓰레드  
라이브러리가 바뀌면 안 돌아갈 수도 있기 때문에, 좀 생각해 볼 필요가 있습니다.

**쓰레드가 1개만 돌면 난감하지 않은가?**



CPU가 여러 개 있을 경우나 블러킹 시스템콜이 중간에 끼이는 경우 같이, 여러 개의 쓰레드가 꼭 동시에 돌아야할 때가 있습니다. 특히 GUI 네트워크 프로그램들 같은 경우에 반응성과 직결되는 문제인데요, 이 경우를 위해 명시적으로 일부분에서 락을 풀어주는 방법을 제공합니다.

위 그림의 ⑫와 ⑭사이가 그런 경우입니다. ⑫에서 Py\_BEGIN\_ALLOW\_THREADS를 해 주면, 그때부터 그 쓰레드는 GIL과 상관 없이 다른 파이썬 쓰레드와 동시에 돌 수 있게 됩니다. 물론, 이 때에는 파이썬 함수를 호출하거나 파이썬 객체를 건드리는 등 파이썬과 관련된 것은 어떤 것도 해서는 안 됩니다. 이런 종류의 작업이 끝나면 ⑭번에서 Py\_END\_ALLOW\_THREADS를 호출해 줘서 이제 그 쓰레드도 GIL의 영향 하에 들어가서, 문맥 전환을 기다리게 됩니다. 물론 즉시 바로 받는 것은 아니고 돌아가던게 100이 되길 기다리는 거죠뭐~ -O-

**GIL을 왜 쓸까?**

위에서 GIL을 대충 살펴봤는데요, 보시다시피 쿼드코어 CPU가 난무하는 이 시대에 좀 껄쩍지근~하게 들리는 것이 사실입니다. 그럼에도 불구하고 파이썬에서 GIL을 계속 쓰고 있는 것은, GIL이 UP에서 상당히 효율적이기 때문입니다. 파이썬 같이 객체가 많고 객체간 접근이 두리뭉실해서 효율적인 락을 구현하기 힘든 경우에는 필요 없는 락에  
의해서 낭비되는 자원이 상당합니다. FreeBSD 4.x까지도 그랬듯이, 전체에 락을 거는 것은 상당히 코드의 복잡성을 줄여주고 UP에서 락에 자원을 낭비하지 않고 높은 효율성을 유지할 수 있게 됩니다.

그렇지만, 역시나 게임기에도 쿼드코어가 들어가는 세상이 되면서, MP지원의 중요성 은 점점 커져가고 있고, 파이썬도 나름대로 그에 대한 대처를 하려고 여러가지 이야기가 나오고 있습니다. 그에 대한 이야기는 다음 시간에~~

*◈ 좀 더 자세한 것을 알고 싶으신 분들은 파이썬 소스 중*[*Python/ceval.c*](http://svn.python.org/view/python/trunk/Python/ceval.c)*에서부터 보기 시작하시면 됩니다.  
◈ 그림에 오류가 하나 있는데, 쓰레드 #1에서 ④~⑤사이에 비어있는 부분은 갈색 블럭이 되어야 합니다.*

Categories[python](http://highthroughput.org/wp/category/python/)Post navigation

[Django 책](http://highthroughput.org/wp/cb-1135/)

[(가짜) 한글 Lorem ipsum](http://highthroughput.org/wp/cb-1137/)

11 thoughts on “파이썬 GIL 깊숙히! (上)”

1. http://0.gravatar.com/avatar/?s=50&d=mm&r=g

[**까나리**](http://kkanari.egloos.com/)

[November 12, 2006 at 1:22 am](http://highthroughput.org/wp/cb-1136/#comment-13798) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13798#respond)

와~~ 이해는 못하지만 그림은 충분히 귀여워요 ㅎㅎㅎ

1. http://0.gravatar.com/avatar/?s=50&d=mm&r=g

[**상우**](http://swoo.org/)

[November 12, 2006 at 2:46 am](http://highthroughput.org/wp/cb-1136/#comment-13799) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13799#respond)

잘 읽었습니다~ FreeBSD SMPng 생각도 나고.. 흥미진진하네요 \_

1. http://0.gravatar.com/avatar/?s=50&d=mm&r=g

**서지원**

[November 12, 2006 at 5:35 am](http://highthroughput.org/wp/cb-1136/#comment-13800) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13800#respond)

파이썬에서 GIL을 없앨 수 없는 가장 큰 이유는 reference counting때문입니다. 다른 부분은 fine-grained locking으로 어느정도 cover가 되지만 – single threaded application에 아주 큰 overhead를 주지는 않지만 – reference counting은 객체에 접근할 때마다 lock을 걸고 풀어야 해서 overhead가 엄청 큽니다. Boehm-gc를 쓰거나 하지 않는다면 GIL을 없애기는 현재로서는 쉽지않은 상황인듯 싶습니다. Boehm-gc를 쓴다면, 지금까지 Tim Peters랑 여러 사람이 엄청나게 optimize시켜놓은 memory allocator는 어떻게 될지 궁금하네요.

1. http://1.gravatar.com/avatar/?s=50&d=mm&r=g

[**daybreaker**](http://daybreaker.info/)

[November 13, 2006 at 4:04 am](http://highthroughput.org/wp/cb-1136/#comment-13801) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13801#respond)

혹시 이게 그 고질적인 Trac의 Subversion 바인딩 버그(apr\_pool\_t 변수 어쩌고 나오는…)와도 관련이 있을까요?;  
Trac 쓰면서 그거 때문에 골치아픈 적이 꽤 많았습니다. ㅠㅠ (뭐 임시해결책으로 mod\_python 인터프리터를 강제로 하나만 쓰게 하는 식으로 어찌 해결은 봤습니다만 완전하지는 않더군요.)

1. http://2.gravatar.com/avatar/?s=50&d=mm&r=g

**상민**

[November 13, 2006 at 5:49 pm](http://highthroughput.org/wp/cb-1136/#comment-13802) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13802#respond)

루비나 펄같은 다른 언어의 경우는 어떤가요?

1. http://2.gravatar.com/avatar/b22de0bd36d2ba3946e56efa88444474?s=50&d=mm&r=g

[**wafe**](http://rnd.xinics.com/~wafe/weblog/)

[November 14, 2006 at 4:44 pm](http://highthroughput.org/wp/cb-1136/#comment-13803) | [Reply](http://highthroughput.org/wp/cb-1136/?replytocom=13803#respond)

무슨 펜을 쓰시는지 궁금하네요 ^^