PHP

이번호에는 PHP에서 시스템 자원을 이용하는 경우를 설명한다. PHP의 수많은 함수는 시스템 자원을 효과적으로 이용하기 위한 것이지만 이번에는 시스템 프로그래밍에 가까운 내용을 다뤄 보기로 한다. 하지만 별다른 내용이 아닐(?) 수도 있다.

안윤호 하이큐텍 mindengine@intizen.com

영원한 아마추어 프로그래머를 자칭하는 안윤호 이사는 '스몰컴퍼니' 라는 PHP 기반 오픈소스 그룹웨어를 발표해 개발자와 업계로부터 반향을 일으킨 주인공. 컴퓨터 하드웨어에서 네트워크, 애플리케이션 프로그래밍까지, 컴퓨터에서 시작해 언어학과 종교, 철학까지 그의 관심분야는 끝이 없다. PHP, 펄 등 스크립트 언어 예찬론자이면서도 운영체제와 C 프로그래밍의 중요성을 누구보다 강조한다. 티벳으로 도 닦으러 가는 게 꿈이라고 한다.

PHP 프로그래밍 | PHP의 시스템 자원 이용

공유 메모리와 세마포어 활용

지금까지 주로 PHP의 중요한 라이브러리 가운데 하나인 PHPLIB을 중심으로 세션과 인증, 템플릿을 다루었다. 그리고 한정된 범위 내에서 PHP의 OOP (Object Oriented Programming)를 다루었다.

간단한 PHP 코딩에는 이러한 내용이 필요하지 않다고 생각하지만 복잡한 시스템을 구현하려면 고려해야 하는 내용이었다고 생각한다. PHPLIB은 이러한 내용을 모두 포함하는 프레임워크를 제공한다(필자는 코딩 그 자체보다는 체계적 사고방식과 접근방식을 같이 설명하려고 노력했지만 충분히 전달되지 않았다면 필자의 역량이 부족하기 때문일 것이다). 지금까지 소개한 코드와 코드를 구성한 체계는 독자들이 시간을 들여 하나하나 분석하면 충분한 보상이 있을 것이다. 이들을 검토하고 자신의 것으로 만드는 것은 PHP 코딩실력을 증진하는 데 도움을 줄 것이다. 분명히 엄밀성은 증가한다.

이번호에는 시스템 자원(resource) 이용에 대해 설명하려 한다. 목표는 역시 '내공' 향상이다. 지난호와 달리 어려운 코드는 나오지 않는다. 이 주제를 다룬 이유는 웹에서 개인 또는 단체가 중요한 작업을 진행하는 경우가 점점 많아지고 있기 때문이다.

공유 메모리와 임계 영역

먼저 공유 메모리(shared memory)와 임계 영역(critical section)에 관련된 문제를 다루려고 한다. 이 부분은 일반적으로 PHP에서는 잘 다뤄지지 않는 내용이다. 임계 영역(또는 동기화 문제)과 공유 메

모리를 사용해 PHP의 변수값을 여러 사용자 간에 공유하고 특정한 작업에 대한 통제를 실시하는 과제가 골자다.

공유 메모리는 우리가 흔히 사용하는 쿠키나 세션과는 다르다. 세마포어는 특정한 프로그램 영역에 대해 다른 프로세스가 접근하는 것을 통제하는 방법이다. 이 방법을 동원하면 네트워크에서 여러 명이 함께 공동으로 작업할 때나 상호 통신과 제어기능이 있는 웹 서버를 구현하는 경우 등에 매우 편리하게 이용할 수 있다.

어떤 작업에 참여하는 사용자 수를 제한하거나 특정 작업 중 다른 사용자의 접근을 차단하거나 접속중인 다른 사용자의 웹 서버 프로세스와 통신하기 위해서는 이러한 방법이 가장 나을 것 같다. 원래 운영체제의 자원이용과 프로세서간의 작업을 통제하기 위해 설계한 개념이므로 뿌리 깊은 역사가 있고 제대로 이용하는 경우 시스템의 신뢰성을 대폭 향상시킬 수 있다. 시스템 프로그래밍을 배운 프로그래머라면 세마포어나 공유 메모리 정도는 다 알고 있겠지만 웹 코딩만 해온 코더라면 친숙하지 않을 수도 있기 때문에 이번 주제로 시스템 자원 이용이라는 관점으로 접근하게 됐다.

전반적으로 논의할 내용은 운영체제(예를 들면, 유닉스)에서 다루는 내용과 비슷하다. PHP의 API는 운영체제와 밀접하게 관련된 기능을 제공하고 있기 때문에 이러한 코딩이 가능하다. 만약 조금만 엽기적으로(?) 코딩한다면 소켓과 리다이렉션 등을 사용해 거의 모든 인터넷과 시스템 자원을 사용할 수 있다. 특히 유닉스에서는 거의 완벽할 만큼 운영체제의 자원을 이용할 수 있다. 이러한 점이 PHP 사용의 가장 큰 매력 중 하나일 것이다.

아무튼 이번호에도 필자의 '아마추어적인' 접근으로 몇 가지 유용한 방법을 소개하려 한다. 이번호의 큰 주제는 결국 공동작업 또는 동시작업으로 압축할 수 있다. 좀더 거창하게 제목을 붙이면 '웹에서 하는 시스템 프로그래밍'이 되겠다. 즉, 웹을 통해 여러 명이 동시에 작업을 진행하는 경우에 도움이 될 수 있는 내용이다.

세마포어 또는 동기화 그리고 공유 메모리

먼저 공유 메모리에 관한 내용부터 살펴보자.

공유 메모리

일반적으로 PHP로 만든 페이지를 호출하면 다른 사용자들이 무슨 일을 하고 있는지 알 수 없다. 이를테면 사용자 A가 어떤 작업을 하고 있고 사용자 B가 다른 작업을 하고 있다고 생각할 때 A와 B는 서로 어떠한 관련성도 없어 보인다. 사용자 A와 B 사이의 데이터를 교환하는 방법은 보통 데이터베이스를 사용하거나 보안이 떨어지는 쿠키를 부분적으로 사용하는 것이 있다. 사용자 A가 작업한 데이터를 임시로 보존하는 방법 역시 비슷한 방법으로 구현하고 있을 것이다. 만약 여러 명이 작업한다면 사용자 간 데이터 교환 구현은 매우 복잡하게 될 것이다.

이 때 유닉스 시스템을 사용하고 있다면 이미 시스템에 구현되어 있는 공유 메모리를 사용하는 좋은 방법이 있다. 공유 메모리는 서버 메모리의 일부를 얻어내(할당받아) 사용자 사이에서 공유하는

것이다. 어찌 보면 당연한 개념이지만 별로 사용되고 있지 않고 NT에서는 지원하지 않는다.

실제 유닉스 머신에서는 이러한 일을 IPC(Inter Process Call)라는 방법으로 해결한다. 서로 독립적인 프로세스 사이에서 통신하는 방법은 1970년대부터 개발되어 왔고, 우리가 익히 알고 있는 시그널이나 여타의 갖가지 방법을 동원해 해결하고 있다. 이들 문제는 당당히 운영체제 교과서에서 수십 페이지를 차지한다. 역사적으로 버클리의 BSD와 AT&T의 시스템 V(System V)는 이러한 문제에 대해 서로 조금씩 다른 방안을 제시해 해결했고, 이번에 다루려는 내용은 시스템 V에서 유래한 방법이다. 시스템 V의 IPC는 세마포어, 메시지 큐, 공유 메모리다. 하나의 프로세스는 이들을 이용해 다른 프로세스와 통신할 수 있고, PHP에서도 이러한 기능을 사용할 수 있다(이들에 대한 자세한 예제 중 고전적 예는 W.R. Stevens의 'Advanced Programming in the Unix Environment'에서 찾아볼 수 있으며 최신 버전 POSIX 예제는 Wrox 출판사의 'Beginning Linux Programming'에서 찾아볼 수 있다).

PHP는 시스템 V 공유 메모리를 사용하는 공유 메모리 함수를 제공한다. 공유 메모리는 말 그대로 여러 개의 프로세스에서 데이터를 공유할 수 있도록 메모리의 특정 영역을 할당한 것이다. 공유 메모리는 우선적으로 전역 변수에 접근하기 위해 사용한다. 일반적인 전역변수와 차이점이라면 httpd-daemon 또는 심지어 펄(Perl)이나 C로 작성된 다른 프로그램에서도 이 공유 메모리 변수에 접근할 수 있고 광범위한 데이터 교환이 가능하다. 즉, C로 만든 프로그램과 PHP로 만든 프로그램이 서로 통신할 수 있다.

공유 메모리는 매우 편리하지만 동시에 여러 곳에서 접근하는 것에 대해서는 안전장치가 없다. 이를 방지하기 위해 세마포어를 사용해 동기화(synchronize)해야 한다. 이러한 일에 대비해 PHP는 시스템 V 세마포어를 사용하는 세마포어 함수를 제공한다. 세마포어는 보통 현재 컴퓨터의 자원에 대해 배타적으로 접근하거나, 한 자원을 동시에 사용할 수 있는 프로세스 수를 제한할 때 사용한다.

공유 메모리와 세마포어는 매우 어려워 보이지만 실제로는 쉽다. 그래서 간단한 예제와 함께 설명하겠다. 공유 메모리를 먼저 설명하고 세마포어를 나중에 설명한다.

공유 메모리 사용법

현재 PHP를 사용하고 있다면 공유 메모리와 세마포어를 사용하기 위해 다시 컴파일해야 한다. PHP 컴파일 옵션에서 —enable-sysvsem과 —enable-sysvshm를 각각 추가해야 한다. 이것으로 사용 준비는 끝난다. 만약 PHP 3를 사용하고 있다면 버전이 3.0.6 이상이어야 하고 PHP 4는 버전에 관계없이 컴파일할 수 있다. 필자는 FreeBSD와 리눅스에서 별 문제없이 사용할 수 있었다. 일반적으로 POSIX 호환인 다른 유닉스에서도 사용할 수 있을 것이다. NT에서는 시스템 V 자원을 이용할 수 없다.

우선 예제를 만들기 전에 이들의 특성을 조사해보자. 공유 메모리와 관련해 PHP 매뉴얼에 나오는 일반적인 환경변수는 다음과 같다.

- ◆ **SHMMAX** : 공유 메모리의 최대 크기로 보통 13만 1072바이트
- ◆ **SHMMIN** : 공유 메모리의 최소 크기로 보통 1바이트
- ◆ **SHMMNI** : 공유 메모리 세그멘트(segment)의 최대 개수로 보통 100개
- ◆ **SHMSEG** : 프로세스당 공유 메모리의 최대 개수로 보통 여섯 개

공유 메모리와 관련해 PHP 매뉴얼에서 나오는 일반적인 함수는 다음과 같다.

- ◆ **shm_attach** : 공유 메모리의 세그먼트를 만들거나 연다.
- ◆ **shm_detach** : 공유 메모리 세그먼트와 연결을 끊는다.
- ◆ **shm_remove** : 유닉스 시스템에서 공유 메모리를 제거한다.
- ◆ **shm_put_var** : 공유 메모리 안에 변수를 삽입하거나 변수 값을 변경한다.
- ◆ **shm_get_var** : 공유 메모리 변수를 반환한다.
- ◆ **shm_remove_var** : 공유 메모리에서 변수를 제거한다.

전반적인 사용 시나리오는 다음과 같다. 먼저 test.php3라는 파일이 공유 메모리를 시스템으로부터 할당받는다고 하자. 우선 shm_attach를 사용해 공유 메모리 세그먼트를 만들고 shm_put_var를 이용해 변수를 공유 메모리에 저장한다. 일반적으로 다음과 같은 코드가 될 것이다(아마 가장 간단한 코드가 될 것이다).

```
<?php
// test.php3
$shm_id= shm_attach(1010); // 1010은 임의의 키 값이다.
echo "id=$shm_id ";
shm_put_var($shm_id,u,10);
?>
```

shm_attach 함수의 형식은 다음과 같다. 앞의 예에서 메모리 크기와 퍼미션은 생략했다는 것을 알 수 있다.

```
int shm_attach (int key [, int memsize [, int perm]])
```

이 함수는 key값에 해당하는 id값을 리턴한다. 경우에 따라 메모리 값을 지정할 수 있는데 이 값은 PHP의 config 파일에 잇는 sysvshm.init_mem의 값이며 지정되지 않으면 1만 바이트를 할당한다. 일반적으로 10만 바이트면 충분한 값이다. 그리고 퍼미션을 지정할 수 있다(기본값은 0666으로 모두가 읽고 쓰고 할 수 있는 값이다. 0644로 지정하면 다른 사용자는 값을 읽을 수만 있다). 다른 *.php 파일에서 shm_attach()를 호출하면 리턴되는 id값은 다르지만 같은 영역을 지정하게 된다.

그 다음에 호출된 shm_put_var는 공유 메모리 안에 변수를 삽입하거나 변수 값을 변경한다. 이 함수의 형식은 다음과 같다.

```
int shm_put_var (int shm_identifier, int variable_key,
mixed variable)
```

공유 메모리의 id인 shm_identifier는 앞의 예에서 $shm_id다. variable key는 u라는 변수 값으로 할당하고 값을 10으로 했다. 이 변수의 형태는 무엇이든지 상관이 없다. 변수형식도 mixed variable로 되어 있지 않은가.

다음은 test1.php3 코드의 일례다. 이 코드는 다음과 같이 작성할 수 있다.

```
<?php
// test1.php3
$shm_id= shm_attach(1010); // 1010은 임의의 키 값이다.
$i=shm_get_var($shm_id,u);
echo "$i ";
?>
```

앞의 예에서 shm_attach의 키 값은 1010으로 test.php3와 똑같다. 반환되는 $shm_id의 값은 같은 공유 메모리를 지정하고 있다. 따라서 shm_get_var를 이용해 변수 u의 값을 추출할 수 있다.

함수 shm_get_var()는 공유 메모리 변수를 반환한다. 이 함수의 형태는 다음과 같다.

```
mixed shm_get_var (int id, int variable_key)
```

이 함수는 특정 id에서 원하는 키 값의 결과를 리턴한다. 값을 리턴하더라도 변경되거나 제거하지 않는 한 원래의 값은 유지된다. 리턴된 값은 앞서 말했듯이 원래의 변수 형태를 유지한다.

그 다음은 test2.php3를 만들어 보자. 이 함수는 test.php3의

변형이다. test2.php3를 실행하고 나면 결과 값이 바뀌어 있을 것이다.

```
<?php
// test.php3
$shm_id= shm_attach(1010); // 1010은 임의의 키 값이다.
echo "id=$shm_id ";
shm_put_var($shm_id,u,"I have changed");
?>
```

이제 앞에서 사용하지 않은 나머지 함수를 정리 해보자 .

- **int shm_detach (int shm_identifier)** : 공유 메모리 세그먼트와 연결을 끊는다. shm_detach()는 shm_attach()에서 지정한 특정한 id(앞에서 예시한 1010 같은)에 의해 만들어진 메모리 세그먼트와 연결을 끊는다. 그러나 유닉스 공유 메모리에 있는 데이터들이 지워지거나 없어지지는 않는다.
- **int shm_remove (int shm_identifier)** : 유닉스 시스템에서 공유 메모리를 제거한다. 유닉스 공유 메모리에 있는 데이터들이 실제로 해제된다.
- **int shm_remove_var (int id, int variable_key)** : 공유 메모리에서 변수를 제거한다. shm_get_var와 같은 형태이지만 공유 메모리에서 해당 변수를 불러오는 것이 아니라 해제하며 할당되지 않은 메모리의 양이 증가한다.

공유 메모리, 어디에 사용하면 좋은가

공유 메모리는 특정 페이지에 접근한 사용자들이 모두 공유하는 데이터를 부여함으로써 PHP에서 데이터 공유를 위해 SQL을 사용하지 않아도 되며 페이지마다 다른 세그멘트 키를 사용함으로써 페이지별로 다른 데이터를 공유하거나 별개의 데이터를 공유할 수도 있다.

가장 간단한 예제로 인트라넷에서 접속 카운터를 만드는 코드를 생성해 보자. 사용자는 로그인했고(PHP_AUTH를 사용하거나 쿠키인증을 하거나 아니면 PHPLIB처럼 세션을 이용한 AUTH를 시행하거나) 누가 접속해 무슨 일을 하고 있는지 알고 싶은 경우가 있다. 데이터베이스를 이용하면 매우 복잡하겠지만 이 작업을 공유 메모리를 사용해 구현한다면 다음과 같은 유사코드(pseudo code)로 만들어낼 수 있다.

```
<?
// 이 부분에서 인증문제를 해결한다.

$shm_id= shm_attach(1010); // 1010은 임의의 키 값이다.

$user_list_temp =shm_get_var($shm_id,user_list);
$value=  time() . ":::" . "$작업내용" ; // 현재 로그인한 id가 my_id다.
$user_list_temp[$my_id ] =$value;
shm_put_var($shm_id,user_list,$user_list_temp);
......
?>
```

다른 *.php3 파일에서 $user_list_temp = shm_get_var($shm_id, user_list)를 처리하면 접속자의 통계를 유지하면 된다. 이들을 리스트하거나 접속 통계를 낼 수 있다. 이를테면 사용자 A는 무엇을 하고 있고 B는 어떤 작업을 하는가 등 나중에 일정 시간이 지난 사용자는 통계에서 제외하거나 변수를 지울 수 있다. 결과적으로 앞의 코드는 데이터베이스보다 매우 간단하고 편리해질 수 있는 가능성을 제시한 것이다. 아니면 좀더 간단한 예제를 만들 수 있다.

```
<?
$shm_id= shm_attach(1010); // 1010은 임의의 키 값이다.
shm_put_var($shm_id,message,"안녕하세요. 저는 처음 접속했어요.");
......
?>
```

다른 화면에서 shm_get_var($shm_id, message)를 사용해 필요한 변수를 리턴 받으면 채팅 사이트 같은 데서는 데이터베이스 없이 메신저 같이 이용할 수 있고 작은 위키위키나 클립보드 같은 형태로 자유롭게 사용할 수도 있다. 변수 저장 형태에 제한이 없기 때문에 상당히 자유롭다. 그 외에도 공유 메모리를 이용하는 다른 애플리케이션들, 이를테면 C 루틴이나 펄로부터 공유 메모리에 데이터를 실시간으로 넘겨받을 수 있다는 장점이 있다.

단, 공유 메모리는 서버가 재부팅되지 않는 한 서버에 남아 있기 때문에 공유 메모리 세그먼트를 직접 제거해 주거나 관리자 루틴에서 제거해야 한다(이번호에서 다루지 않는 또 다른 공유 메모리의 형태는 역시 유닉스 공유 메모리 자원을 이용하는 shmop 함수가 있다. 이 함수는 PHP 4에서만 이용할 수 있고 특별히 사용이 더 편한 것도 아니므로 자세히 설명하지 않았다. 관심이 있는 독자는 이 부분에 대한 PHP 매뉴얼을 읽어 보기 바란다).

좀더 복잡한 실제 예제

이번에는 조금 더 복잡한 예를 들어보자. phplib에 있는 세션 컨테이너인 'ct_모듈'의 하나로 세션 컨테이너를 공유 메모리로 만든 경우다(ct_shm.inc). 이 코드는 지난번에 설명한 세션 구현을 읽지 않은 독자라면 분석하지 않아도 된다. 데이터베이스를 사용한 지난호의 세션 컨테이너 코드보다 훨씬 단순하게 세션을 구현할 수 있다는 것을 알 수 있다. 분석을 단순하게 하기 위해 의도적으로 초기버전의 코드를 예시했다.

〈리스트 1〉은 세션 id 변수를 만들어 키 값으로 이용하고 이 키 값을 근거로 자신의 세션 값들을 공유 메모리에 저장하고 불러오는 것이다. 공유 메모리의 세그먼트 키 값은 $shm_key=900000으로 고정되어 있다. 또한 퍼미션이 0600으로 고정되어 다른 사람은 공유 메모리 데이터에 접근할 수 없고 공유 메모리를 만든 사용자만 이용할 수 있다(읽고, 쓰고). 이 코드는 나중에 인증이나 사용자 변수를 처리할 때 상당히 많은 변형의 자유를 부여한다. 단순한 md5로도 상당한 수준의 보안이나 사용자를 관리할 수 있다.

ac_get_lock()이나 ac_release_lock() 같은 함수는 코드 접근 통제를 위해 세마포어를 사용했다. 컬럼을 죽 읽어 온 독자라면 연습삼아 다음 코드를 사용해 세션을 구현해 보기 바란다. 사용자 수가 많지 않고 세션에 저장할 데이터가 크지 않은 경우라면 공유 메모리를 사용해 세션을 매우 빠르고 안전하게 구현할 수 있다. 어떠한 데이터도(SQL 데이터나 플레인 파일 또는 LDAP) 밖에서는 관측할 수 없으므로 매우 안전하다.

코드를 훑어봤으면 다음에 설명할 세마포어로 넘어가자.

세마포어

〈리스트 1〉의 ct_shm.inc 코드는 특정 영역을 보호하기 위해 세마포어를 사용했다. 세마포어는 일반적으로 운영체제나 멀티쓰레드 프로그램에서 흔히 볼 수 있다. 세마포어 구현코드는 운영체제를 설명한 책에서 찾아보기 바라며 여기서는 그 특성과 사용법에 대해 설명할 것이다.

다음은 본지의 디지털 메신저 컬럼을 쓰고 있는 나성언 씨가 운영체제를 강의하면서 세마포어를 설명한 내용이다(www.chollian.net/~lase). 이미 세마포어의 개념에 대해 알고 있는 독자는 건너뛰어도 무방하지만 매우 재미있게 설명한 글이라서 부분적으로 인용했다.

"술집에 다니다 보면 화장실을 잠가두고 손님이 주인에게 요청을 해야 열쇠를 내어주는 그런 집이 있다. 여기서 화장실 키의 기능은 세마포어의 기능과 거의 같다고 볼 수 있다. 하나 밖에 없는 화장실은 술집 시스템의 공유 자원이며 손님은 그

〈리스트 1〉 ct_shm.inc

```
<?php

##
## Copyright (c) 1998,1999 Sascha Schumann <sascha@schumann.cx>
##
## $Id: ct_shm.inc,v 1.4 1999/07/07 20:43:44 sas Exp $
##
## PHPLIB Data Storage Container using Shared Memory
##

class CT_Shm {
   ##
   ## Define these parameters by overwriting or by
   ## deriving your own class from it (recommened)
   ##

   var $max_sessions = 500; ## maximum supported sessions
   var $shm_key = 900000;   ## key of shared memory segment (unique)
   var $shm_size = 64000;   ## size in bytes

   ## end of configuration

   var $shmid; ## our shared memory handle
   var $semid; ## our semaphore handle

   function extract($id) {
      return substr($id, 0, strpos($id, "_"));
   }

   function ac_start() {
      $this->shmid = shm_attach($this->shm_key, $this->shm_size, 0600);
   }

   function ac_get_lock() {
      $this->semid = sem_get($this->shm_key + 1);
      sem_acquire($this->semid);
   }

   function ac_release_lock() {
      shm_detach($this->shmid);
      sem_release($this->semid);
   }

   function ac_newid($str, $name) {
      for($i = 1; $i <= $this->max_sessions &&
            (@shm_get_var($this->shmid, $i) != false); $i++);
      $id = $i."_".$str;
      $this->ac_store($id, $name, "");
      return $id;
   }

   function ac_store($id, $name, $str) {
      $val = "$id;".urlencode($name).";".urlencode($str).";".time();
      shm_put_var($this->shmid, $this->extract($id), $val);
      return true;
   }

   function ac_delete($id, $name) {
      shm_remove_var($this->shmid, $this->extract($id));
   }

   function ac_gc($gc_time, $name) {
      $cmp = time() - $gc_time * 60;
      for($i = 1; $i <= $this->max_sessions; $i++)
         if(($val = @shm_get_var($this->shmid, $i)) != false) {
            $dat = explode(";", $val);
            if($name == $dat[1] && strcmp($dat[3], $cmp) < 0)
               shm_remove_var($this->shmid, $i);
         }
   }

   function ac_halt($s) {
      echo "<b>$s</b>";
      exit;
   }

   function ac_get_value($id, $name) {
      $i = $this->extract($id);
      $var = shm_get_var($this->shmid, $i);
      if($var == "") return("");
      $dat = explode(";", $var);
      ## if classname or md5 id does not match...
      if($name != urldecode($dat[1]) || $dat[0] != $id)
         $this->ac_halt("security stop");
      return urldecode($dat[2]);
   }
}
?>
```

자원을 요구하는 태스크와 같다. 주인(커널)은 요청한 순서대로 손님에게 키를 주고, 키를 받은 손님은 볼일을 보고 난 후 주인에게 키를 반납한다. 만일 한 손님이 화장실을 사용하는 중에 다른 손님이 키를 요구한다면 주인은 화장실이 사용중이니 기다리라고 할 것이다. 실제 운영체제에서도 이미 할당된 자원을 요청한 태스크가 할 수 있는 일은 기다리는 것뿐이다. 따라서 한 시점에서 자원을 사용하려고 하는 두 태스크들은 상호 배제(Mutual Exclusion)된다. 여기에 든 예에서 발생할 수 있는 다른 사항, 즉 다른 곳에다 실례를 한다든지 하는 경우는 물론 논외로 할 필요가 있다. 좀더 개념을 확장해서 만약 화장실이 두 곳이라면 키도 두 개를 둘 수 있다. 즉, 자원이 동시에 사용할 수 있는 여유가 있다면 그만큼 키의 개수도 늘어날 수 있다. 이와 같이 세마포어는 1과 0의 두 가지 상태만을 갖는 경우와 카운터를 통해 여러 개의 값을 갖게 될 수 있는 경우로 구분할 수 있다. 여기서 한 가지 주의할 점은 화장실 자체는 공유 자원이지만 키는 자원의 사용권을 나타내는 상징적인 도구가 된다는 점이다. 화장실 자체와 키는 대응되는 관계에 불과하며 세마포어를 사용할 때 운영체제는 화장실 자체는 관리하지 않고 키만을 관리한다."

결국 세마포어는 보호장치의 일종인 것이다. 어떤 프로세스가 바이너리 세마포어를 설정해 어떤 영역에 대한 보호를 설정하면 다른 프로세스는 이 세마포어가 해제될 때까지 이 영역에 대해 접근할 수 없다. 더 PHP적으로 말하면 세마포어가 걸려있는 SQL 업데이트 영역이나 공유 메모리 접근 영역은 사용할 수 없게 하는 것이다.

어찌 보면 별 내용이 아닌 것 같지만 동기화나 크리티컬 섹션의 문제는 오랫동안(수십 년 동안) 수학자와 전산학자를 괴롭힌 문제이며 가장 어려운 부분이기도 하다. 이 문제에 대한 자세한 설명은 운영체제론을 공부하는 수밖에 없다. 대부분의 간단한 문제는 간단한 세마포어 적용만으로 해결할 수 있고, 이는 PHP도 마찬가지다.

PHP의 세마포어 함수

PHP에서 제공하는 세마포어 관련 함수는 모두 세 가지다. sem_get은 세마포어 id를 얻어낸다. 이 함수의 형식은 다음과 같다.

```
int sem_get (int key [, int max_acquire [, int perm]])
```

리턴 값이 양수이면 성공이고 아니면 세마포어 획득에 실패한 것이다. 키 값은 앞서 설명한 공유 메모리 키의 개념과 같다. 첫 번째 호출에서 세마포어를 만들어 내고 세마포어 id를 반환한다. 퍼미션의 기본값은 0666이지만 따로 지정되는 경우 세마포어 사용 퍼미션을 부여하고 max_acquire 인수는 따로 지정되지 않는 한 1의 값을 가지며 어떤 세마포어에 동시에 접근할 수 있는 프로세스의 개수를 뜻한다. 같은 프로그램이나 다른 프로그램에서 sem_get을 호출하면 다른 id가 반환되지만 이 id는 결국 같은 세마포어를 지정하고 있다.

sem_acquire는 세마포어를 획득하기 위해 필요하다. 이 함수의 형태는 다음과 같다.

```
int sem_acquire (int sem_identifier)
```

sem_identifier는 앞의 sem_get을 특정한 키로 호출하고 세마포어가 만들어지면서 반환한 세마포어의 id다. 이 id를 사용해 sem_acquire를 호출했을 때 세마포어를 사용할 수 있으면 true를, 아닌 경우 false를 반환한다. sem_acqire()는 세마포어가 사용가능할 때까지 블록된다. 이미 다른 프로세스에 의해 점유된(획득된) 세마포어를 점유하려는 시도는 세마포어를 점유한 프로세스의 개수가 max_acquire 이하가 될 때까지 계속 차단된다.

sem_release는 세마포어를 풀어준다. 이 함수의 형태는 다음과 같다.

```
int sem_release (int sem_identifier)
```

결과는 세마포어 해제에 성공하면 true를, 실패하면 false를 반환한다. 일반적으로 세마포어 해제는 세마포어를 만든 프로세스가 행하는 것이 보통이다. 다른 프로세스가 해제하는 경우는 주의 메시지가 출력될 것이다(불가능한 것은 아니다). 세마포어가 해제되고 나면 다시 임의의 프로세스가 이 세마포어를 획득할 수 있다.

세마포어 사용 예제

가장 간단한 예제를 통해 세마포어의 동작을 확인해 보자. 먼저 다음 파일을 작성한다. 이 스크립트의 용도는 세마포어를 획득하고 시간이 걸리는 작업을 실행하는 것이다.

```
<?
   // sema.php3
   $sem_id = sem_get(1001);            // 1001은 고유한 값이다.
   $sem_result=sem_acquire($sem_id);   // 세마포어를 획득한다.
   echo $sem_result;

   for ($i=1;$i<100000;$i++)
   {
      echo " - $i <br> ";
   }

   sem_release($sem_id);               // 세마포어를 해제한다.

?>
```

이번에는 다음 파일을 입력한다. 이 스크립트의 역할은 앞의 코드가 세마포어를 점유하는 동안 앞의 프로그램이 이미 획득한 세마포어를 가져오려고 대기하는 것이다. 이러한 세마포어를 블러킹 세마포어라고 부르며 일반적으로 많이 사용하는 형태다.

```
<?
   // sema_observe.php3

   echo "* begin *";
   $sem_id = sem_get(1001);            // 1001은 고유한 값이다.
   .
   $sem_result=sem_acquire($sem_id);  // 세마포어 획득을 시도하며 대기한다.
   echo $sem_result;                  // 결국 성공하면 1을 출력한다.
   echo "* end *";
   sem_release($sem_id);              // 세마포어를 해제한다.
?>
```

웹 브라우저에서 sema.php3를 실행하고 숫자가 증가하는 것을 지켜보면서 sema_oberve.php3를 실행한다. 두 번째 프로그램인 sema_observe.php3가 한참 대기후에 실행되는 것을 볼 수 있다. 이것이 바로 의도하던 상호접근 차단 메커니즘이다. 결과적으로 이 코드는 해당 영역에 대한 동시 접근을 막는 것이다. 이제 다음 예제를 보면서 조금 더 실제적인 문제를 해결하기로 하자.

웹 사이트의 어떤 웹 페이지를 수천 명이 동시에 사용하는 경우를 가정해보자. 이 웹 페이지에 사용자가 한 번씩 접근할 때마다 공유 메모리 카운터가 증가하는 경우를 생각할 수 있다. 아니면 SQL의 조회수가 증가한다고 생각할 수도 있다. 만약 공유 메모리를 사용하는 경우라면 다음과 같은 코드를 만들어 보자.

```
<?
   // sema_and_shm.php3
   $shm_id= shm_attach(1010);          // 1010은 임의의 키 값이다
   $sem_id = sem_get(1001);            // 1001은 고유한 값이다.

   $sem_result=sem_acquire($sem_id); // 세마포어를 획득한다.
   // 이 곳부터 크리티컬 섹션이다.
   $i=shm_get_var($shm_id,u);          // 카운트 변수를 불러온다.
   echo "Count= $i ";
   shm_put_var($shm_id,u,$i+1);
   sem_release($sem_id);               // 세마포어를 해제한다.
   // 크리티컬 섹션은 여기에서 끝난다.
?>
```

세마포어가 없는 경우와 비교하는 것이 이 코드의 장점을 가장 잘 설명하는 예가 될 것이다. 만약에 사용자가 Count=9일 때 동시에 두 명이 접속한 경우를 고려해 보자. 사용자 A는 현재 카운트가 9이므로 카운트를 10으로 올릴 것이다. 이 값이 업데이트되더라도 사용자 B 역시 카운트가 9라고 인식하고 카운트를 10으로 올리는 경우가 발생할 수 있다. 원래는 11이 돼야 하지만 10으로 인식되는 경우가 발생한다. 이는 분명히 오류다. 앞의 코드에서는 임계영역이라고 부르기도 하는 크리티컬 섹션으로 업데이트하는 사용자의 수를 한 명으로 제한해 이러한 문제를 근본적으로 해결하고 있다.

데이터베이스를 사용하는 경우 lock()을 걸어 이러한 일을 어느 정도 막을 수 있지만 사용에 제한이 있고 역시 특정 코드 영역에 대해 사용자를 제한하는 경우가 더 안전할 것은 틀림이 없다. 많은 웹 사이트에서 잘못된 링크나 카운트는 이러한 고려가 없는 상황에서 발생하는 경우가 많다. 전자 상거래 같은 경우 임계영역이 구현되지 않으면 심각한 문제가 발생할 수 있다.

너무나 싱겁게 설명이 끝나지만 이것이 지금까지 장황하게 설명한 공유 메모리와 세마포어의 중요한 포인트이며 코드차원의 보호가 아닌 시스템 자원을 동원하는 근원적 해결점이다.

그 외에도 두 개 이상 사용하고 몇 개의 공유 메모리를 동원하면 여러 개의 PHP 프로세스가 상호교신하면서 순차적 작업을 진행할 수도 있는데 동기화라는 관점에서는 매우 재미있는 예제이지만 독자들이 PHP에서 이러한 경우를 사용할 가능성이 거의 없기 때문에 생략한다. 하지만 조금 생각해보면 공유 메모리와 세마포어를 사용해 페이지 이동과 열려있는 윈도우 숫자를 거의 완벽하게 통제할 수 있다.

다음에 설명할 예제는 앞의 예보다는 조금 단순한 시스템 자원 이용문제에 관한 것이다.

다른 방법으로 시스템 자원을 이용하는 경우

우리가 앞에서 본 세마포어와 공유 메모리는 시스템 자원을 이용하는 일례에 지나지 않는다. PHP 함수는 매우 광범위하고 다양하다. 함수 외에도 시스템 이용에 관한 많은 변수가 있다. 이들을 모두 고려한다는 것은 불가능하지만 중요한 부분을 재조명하고 PHP의 확장에 관해 이야기하려고 한다(자원을 이용한다는 표현이 어떤 경우에는 어울리지 않을 수 있고 단순히 시스템과 통신하는 것에 지나지 않더라도 필자 나름대로는 고심한 내용이다).

다른 유닉스 프로그램의 기능 이용

만약 다른 프로그램의 기능을 이용해야 하는 경우가 있다면 어쩔 수 없이 이용해야 할 것이다. 이미 기존 PHP 함수가 수없이 준비되어 있지만 유닉스에서 구현된 수많은 다른 자원을 사용해야만 할 때가 반드시 있을 것이다. 유닉스의 유틸리티나 도구세트들은 30년 이상 개발돼 온 것이며 이들을 제대로 이용하는 것은 빠른 개발이나 프로토타입 구성에 결정적이다. 어떤 프로그램들은 PHP로는 개발할 수 없는 복잡한 기능이 있는 것도 있다.

첫 번째로 생각할 수 있는 응용은 exec()이나 system()으로 프로그램을 직접 구동하는 경우다. 이러한 응용은 펄로 코딩할 때와 유사하다. PHP는 그 자체가 아파치 서버의 모듈로 동작시키기 위해 개발된 것이어서 실행 프로세스의 수를 증가시키는 exec()이나 system() 함수로 다른 프로그램을 구동하는 것을 권장하지는 않는다. 그러나 경우에 따라 어쩔 수 없이 다른 프로그램을 이용하는 경우라 할 수 있다.

프로그램을 실행하기 위해 생각할 수 있는 가장 간단한 코드는 (backtick)을 이용하는 경우다. 시스템에 접속한 사람을 웹으로 감시하기 위한 간단한 예제를 만들 수 있다.

```
<?php
$who=`who`;
echo $who;
?>
```

모양이 좋지는 않겠지만 시스템에 접속된 사람들을 보여 줄 것이

다. 모양을 다듬기 위해 grep이나 awk를 사용할 수도 있다. 이를테면 $who= who | grep ...처럼 사용하는 다음과 같은 용법도 가능하다.

```
$x="mind";
$who=`who | grep $x`;
```

Sed나 awk 또는 펄을 사용한 복잡하고 본격적인 필터를 사용할 수도 있다.

주기적인 시스템 감시를 위한 cron이나 at 데몬의 출력 파일을 이용할 수 있다. 원하는 시간마다(5분, 10분 아니면 특정 시간) 특별한 조작이나 감시를 할 수도 있다. 비슷한 기능을 하는 함수로 exec()이나 passthru() 같은 것들이 있다. PHP 매뉴얼에서 이들의 기능을 따져보고 적당한 함수를 골라 써야 할 것이다.

백틱()을 이용하는 경우 메모리 낭비를 막기 위해 파이프(pipe)를 사용할 수 있다. 백틱은 간편하지만 모든 출력을 하나의 변수에 저장하므로 메모리 사용이 많아진다. 이를 회피하기 위해 파이프로 실행 프로그램의 출력 스트링을 처리한다.

```
$fp = popen ("프로그램", "r");

while ($line=fgets($var,1024))
{
   $output.=$line ; // 또는 라인별로 처리할 수 있다.
}

pclose($fp);
```

파이프는 센드메일(sendmail) 같은 프로그램에 직접 데이터를 기록해야 하는 경우에도 사용할 수 있다. 몇 개의 웹 메일 프로그램은 mail() 함수를 이용하지 않고 바로 프로그램에 파이프를 이용한 입력을 시도한다.

더 복잡한 프로그램은 출력의 리다이렉션을 포함한 파이프의 양방향 통신을 한다. 파이프 역시 유닉스의 강력한 IPC 중 하나이기 때문에 다른 프로세스와 통신과 통제가 가능한 메커니즘을 쉽게 구현할 수 있다.

파이프 다음으로 자원에 대한 직접적인 통제는 네트워크 환경에서 소켓(af_inet이나 af_unix)을 이용한 다른 네트워크 데몬과 직접적인 통신을 통해 일어난다. 가장 간단한 예는 다음과 같이 웹 서버에 직접 소켓으로 접속해 데이터를 얻어오는 경우다.

```
<?
$fp = fsockopen ("www.php.net", 80, $errno, $errstr, 30);
if (!$fp) {
   echo "$errstr ($errno)<br>\n";
} else {
   fputs ($fp, "GET / HTTP/1.0\r\n\r\n");
   while (!feof($fp)) {
      echo fgets ($fp,128);
   }
   fclose ($fp);
}

?>
```

이 코드가 무척 단순해 보이지만 Snoopy 라이브러리는 소켓으로 데이터를 얻은 후, 텍스트를 처리해 웹 페칭 라이브러리를 만들었다. 강력한 PHP 링크 생성기나 웹 컨텐츠 분석을 쉽게 도와주는 도구로도 사용할 수 있다. 이 프로그램도 처음에는 간단한 소켓 처리기로 구현된 것이다. 하지만 인터넷 접속 프로토콜에 익숙해지면 더 많은 일을 할 수 있다. 약간의 코드 수정으로 원시적인 텔넷이나 FTP 같은 것을 쉽게 구현할 수 있다. 실제로 네트워크 관련 PHP 함수들은 네트워크 클라이언트의 인터페이스를 PHP로 구현한 것이다. 따라서 개발자의 마음에 들지 않으면 PHP 함수마저 무시하고 개발자가 직접 시스템과 통신하는 애플리케이션을 만들기도 한다.

대표적인 예가 squirrel mail이다. squirrel mail은 PHP에 이미 구현된 IMAP 클라이언트를 사용하지 않고 PHP로 IMAP 서버와 직접 소켓으로 통신하면서 데이터를 처리하는 대표적인 예다. IMAP 클라이언트가 구현하기 어렵다는 것을 감안하면 이 프로그램의 개발자는 여러 가지 기법을 동원해 IMAP 클라이언트를 구현했다. 아마도 PHP에 내장된 IMAP 함수 중 무엇인가 마음에 들지 않았기 때문에 직접 소켓만으로 구현해 만들어냈을 것이다. 다음은 대표적인 코드의 일부를 예시한 것으로 소스의 스타일을 보이기 위해 발췌했다.

```
$imap_stream = fsockopen ($imap_server_address, $imap_port,
$error_number, $error_string, 15);
// 직접 소켓으로 IMAP에 접속한다.
$server_info = fgets ($imap_stream, 1024);

// Decrypt the password
$password = OneTimePadDecrypt($password, $onetimepad);

/** Do some error correction **/
if (!$imap_stream) {
   if (!$hide) {
      set_up_language($squirrelmail_language, true);
      printf (_("Error connecting to IMAP server: %s.")."
        <br>\r\n", $imap_server_address);
      echo "$error_number : $error_string<br>\r\n";
   }
   exit;
}

fputs ($imap_stream, "a001 LOGIN \"" . quoteIMAP($username) .
        '" "' . quoteIMAP($password) . "\"\r\n");
// 서버에 직접 소켓으로 통신을 시도
$read = sqimap_read_data ($imap_stream, 'a001', false, $response,
$message);
// IMAP 데이터를 직접 읽음
/** If the connection was not successful, lets see why **/
   if ($response != "OK") {
      if (!$hide) {
         if ($response != 'NO') {
            // "BAD" and anything else gets reported here.
            set_up_language($squirrelmail_language, true);
            ..
```

IMAP 같이 복잡한 프로토콜을 소켓과 정규표현식만으로 다룰 수 있다는 것을 고려한다면 PHP에서 다른 네트워크 자원을 이용하는 것은 상당한 수준까지 가능하다고 봐야 할 것이다. 프로토콜만 파악할 수 있으면 메신저나 다른 네트워크 서비스들도 PHP에서 직접 처리할 수 있다. 그 외에도 약간의 보안문제를 고려한다면 suid 같은 방법을 동원해 PHP의 일반적인 사용자인 nobody에게 루트 수준의 권한을 부여한 C 루틴을 만들어 시스템을 직접 통제할 수도 있다. 이런 것은 예전의 CGI들에서 흔히 볼 수 있는 테크닉들이다. 시스템 자원의 이용을 위해서는 과거의 CGI 문서들을 찾아보는 것도 많은 도움이 될 것이다.

그 외의 자원 이용에 대해

'PHP Developer's Coolbook'의 17장, 20, 23장에는 자바, COM, CURL 같은 인터페이스를 다루는 방법을 설명하고 있다. 이들은 시스템 또는 PHP의 API에 걸친 내용이다. 이 책에는 이번호에서 다루지 못한 내용을 포함해 새로운 내용이 많다. 코딩수준을 높이고 싶은 독자는 반드시 읽어보기를 권한다. 다양한 예제와 새로운 내용이 많이 포함되어 있는 책이다.

끝으로

연재 시작부터 몇 달의 시간이 흐르는 동안 PHP의 중요한 프로젝트들은 개발이 부진하기도 하고 완전히 새로운 양상의 진화과정을 밟기 시작한 것들도 있다. PHP로 만든 소프트웨어들은 양적으로 거의 두 배 가까운 규모로 늘었다. 이들의 전반적인 특징이라면 사용되는 함수의 이용 수준이 크게 높아졌으며 사용자 인터페이스가 강화됐다는 점이다. 이제는 간단한 애플리케이션과 복잡한 애플리케이션이 확연하게 차이나며 복잡한 PHP 코드의 수준은 이제 펄 수준에 근접하는 정도이거나 그 이상인 경우가 많아진 것으로 보인다. 또 다른 특징은 코드의 재사용이 늘어난 것이다. 즉, 어느 정도 안정적이 된 모듈을 지속적으로 재사용하는 경향이 늘어나고 있다. 이것은 오픈소스 개발에서는 당연한 접근방법일 것이다.

최근에는 PHP의 유용성에 관한 논의가 많이 일어나고 있다. 실제로 PHP만으로 사이트를 만드는 곳은 별로 없으며 액티브X나 자바 애플릿을 통합하는 사례가 늘어나고 있다. 참고로 말하면 필자 역시 PHP나 젠드(Zend)만이 가장 훌륭한 웹 애플리케이션 언어라고 보지는 않는다.

필자도 상용 서비스를 위한 웹 사이트를 만들 때는 자바 애플릿을 포함해 구성하고 있다. 그래도 웹 사이트의 프레임을 만드는 것은 역시 PHP나 ASP 같은 언어다. 구현에 필요하기 때문에 ASP나 PHP를 사용하는 것이다. 언어 자체에 정당성을 부여하는 것은 결국 사용자층이다(파이썬은 매력적이지만 아직도 많은 API를 스스로 만들어야 한다. 자바 역시 훌륭하지만 필요한 클래스나 라이브러리들을 일일이 재구성해야 한다. 이러한 작업에는 시간과 노력, 제작비가 많이 소요된다. PHP는 이러한 점에서 큰 강점을 지니고 있다. 하지만 PHP 역시 너무나 많은 함수 모듈이 있어서 오히려 이러한 점이 성장의 장애가 될지도 모르겠다).

넷크래프트 조사에 의하면 이제는 600만여 개(IP로는 100만개가 넘는다)의 웹 사이트가 PHP를 사용한다는 통계가 보고됐다. PHP는 웹 애플리케이션 언어로는 매우 성공적이었기 때문에 유용성에 관한 논의는 필요없을 것 같다. 실용적인 관점에서 볼 때는 어떻게 하면 더 쉽게 좋은 애플리케이션을 만들 수 있는지에 대한 논의가 가장 중요하다. 유지비용도 저렴해야 할 것이다. 결국 코드 생산성 문제가 대두되며 여기에는 튼튼한 기초를 가진 코드들이 뒷받침돼야 한다. 기초라는 것은 개발 환경과 개발자의 실력을 모두 포함한다. 이번에 소개한 내용이 독자들이 문제를 해결하는 데 도움이 되기를 바라면서 글을 마친다.

정리 : 송우일 wooil@sbmedia.co.kr

참 고 자 료

❶ PHP 매뉴얼(www.php.net)
❷ Beginning Linux Programming 2/e, R.Stones & N.Matthew, Wrox Press(정보문화사의 번역본도 나와 있다)
❸ Advanced Programming in the Unix Environment, W.R. Stevens, Addison Wesley
❹ Operating System Concepts, A.Silberschatz, John Wiley & Sons
❺ PHP Developer's Cookbook, S.Hughes, A. Zmievski, SAMS
❻ Managing IMAP, D.Mullet K.Mullet, O' Reilly