IPC with PHP : API, Approaches and Applications

Andrey Hristov
Dorten GmbH

International PHP Conference 2004
Spring Edition
May 5th, 2004

Who am I?

- BSc in Computer Engineering
- Student at University of Applied Sciences, Stuttgart, Germany
- Programming for the Web since year 2000
- Working on the PHP core since year 2002
- Author of pecl/stats
- Working for Dorten GmbH, Germany

Survey

How many of you know about IPC in means of shared memory, semaphores, message queues?

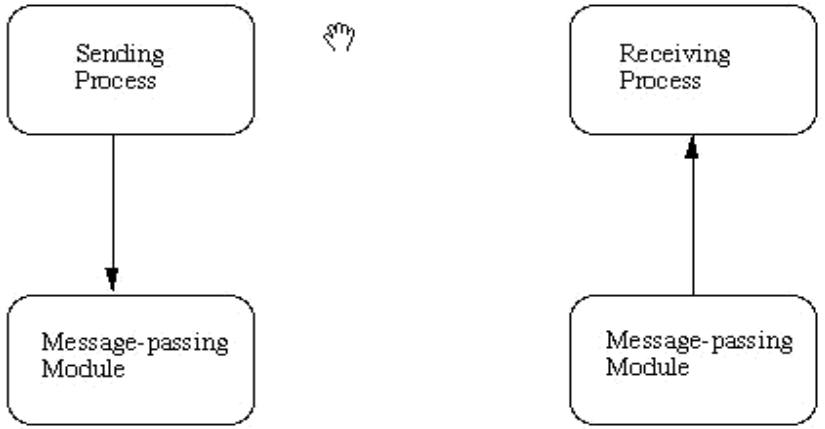
How many of you have worked with IPC primitives with PHP?

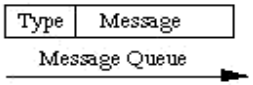
Semaphores

- Invented by Dijkstra . P(s) and V(s) operations
- Used for synchronisation
- Binary semaphores
- Non-binary semaphores

Message queues

- Simple idea : two or more processes exchange information via messages.





- IPC messaging allows sending and receiving in arbitrary order – on the contrary of reading from a stream.
- Blocking and non-blocking send/receive.

Shared memory

- Memory segments available for more than one process
- The processes map their virtual space to the shared memory segment
- The access is permission based (just like every file in a *nix OS)
- Synchronization needed in case of multi-writing – a binary semaphore is usually used.

PHP API for System V IPC (0)

- ext/sysvshm --enable-sysvshm (>= 3.0.6)
- ext/sysvsem --enable-sysvsem (>= 3.0.6)
- ext/sysvmsg --enable-sysvmsg (>= 4.3.0)

PHP API for System V IPC (1)

- shm_attach()
- shm_detach()
- shm_get_var()
- shm_put_var()
- shm_remove_var()
- shm_remove()

- msg_get_queue()
- msg_receive()
- msg_remove_queue()
- msg_send()
- msg_set_queue()
- msg_stat_queue()

- sem_acquire()
- sem_get()
- sem_release()

ext/shmop

- Written by Ilia Alshanetsky & Slava Poliakov
- Works on Win32 and *nix
- Better interoperability with the non-PHP world.
- Since PHP 4.0.3 (4.0.4 – current naming)

API:

- shmop_open()
- shmop_close()
- shmop_delete()
- shmop_read()
- shmop_write()

pecl/memcache

- “memcached is a high-performance, distributed memory object caching system, generic in nature, but intended for use in speeding up dynamic web applications by alleviating database load”.
- memcached runs on every server in the web server farm. It is CPU lightweight and memory “hungry”. Good symbiosis with httpd.
- Developed by Antony Dovgal.
- Works on *nix and Win32 (PHP4 and PHP5).
- Both procedural and OO API.

Working with semaphores (procedural way) (0)

```
<?php
$key = ftok('/home/andrey/test/phpconf/sem_create.php', 'R');
var_dump($key);
$sem_id = sem_get($key, 1, 0666);
if ($argc == 2 && $argv[1] == "clear") {
        echo "Clearing semaphore : $key\n";
        sem_remove($sem_id);
        exit;
}
$res = sem_acquire($sem_id);
var_dump(strftime('TIME :  %H:%M:%S', (int)microtime(1)));
sleep(5);
$res = sem_release($sem_id);
var_dump(strftime('TIME :  %H:%M:%S', (int)microtime(1)));
?>
```

Output :

```
Console 1 :
andrey@poohie:~/test/phpconf> php sem_create.php
int(1376196715)
string(16) "TIME :  16:12:05"
string(16) "TIME :  16:12:10"
Console 2 :
andrey@poohie:~/test/phpconf> php sem_create.php
int(1376196715)
string(16) "TIME :  16:12:10"
string(16) "TIME :  16:12:15"
```

Working with shared memory (procedural way) (0)

```
<?php
$key = ftok('/home/andrey/test/phpconf/sem_create.php', 'R');
var_dump($key);
$shm_id = shm_attach($key, 1024, 0666);
if ($argc == 2 && $argv[1] == "clear") {
        echo "Clearing shared memory : $key\n";
        shm_detach($shm_id);
        exit;
}
$var = strftime('TIME :  %H:%M:%S', (int)microtime(1));
echo "Putting into SHM : ";
var_dump($var);
$res = shm_put_var($shm_id, 1/*var_key*/, $var);
var_dump($res);
$var2 = shm_get_var($shm_id, 1/*var_key*/);
echo "Read from SHM : ";
var_dump($var2);
shm_remove_var($shm_id, 1);
?>
```

Output :

```
andrey@poohie:~/test/phpconf> php shm_create.php
int(1376196715)
Putting into SHM : string(16) "TIME :  17:17:54"
bool(true)
Read from SHM : string(16) "TIME :  17:17:54"
```

Working with shared memory (procedural way) (1)

```
<?php
$key = ftok('/home/andrey/test/phpconf/shm_create2.php', 'R');
$shm_id = shm_attach($key, 1024, 0666);
$var = strftime('TIME :  %H:%M:%S', (int)microtime(1));
echo "Putting into SHM : ";
var_dump($var);
$res = shm_put_var($shm_id, 1/*var_key*/, $var);
var_dump($res);
sleep(10);
$var2 = shm_get_var($shm_id, 1/*var_key*/);
echo "Read from SHM : ";
var_dump($var2);
?>
```

Output :

Console 1 :

```
andrey@poohie:~/test/phpconf> php shm_create2.php
Putting into SHM : string(16) "TIME :  17:54:47"
bool(true)
Read from SHM : string(16) "TIME :  17:54:51"
```

Console 2 :

```
andrey@poohie:~/test/phpconf> php shm_create2.php
Putting into SHM : string(16) "TIME :  17:54:51"
bool(true)
Read from SHM : string(16) "TIME :  17:54:51"
```

Working with shared memory (procedural way) (2)

```
<?php
$key = ftok('/home/andrey/test/phpconf/shm_create2.php', 'R');
$shm_id = shm_attach($key, 1024, 0666);
$sem_id = sem_get($key, 1, 0666);
if ($argc == 2 && $argv[1] == "clear") {
  echo "Clearing shared memory / semaphore : $key\n";
  shm_detach($shm_id);sem_remove($sem_id);exit;
}
$var = strftime('TIME :  %H:%M:%S', (int)microtime(1));
echo "Putting into SHM : ";var_dump($var);
sem_acquire($sem_id);
printf("Acquired access @%s:\n", strftime('TIME :  %H:%M:%S', (int)microtime(1)););
$res = shm_put_var($shm_id, 1/*var_key*/, $var);
sleep(10);
$var2 = shm_get_var($shm_id, 1/*var_key*/);
sem_release($sem_id);
echo "Read from SHM : ";var_dump($var2);
?>
```

Output :

Console 1 :

```
Putting into SHM : string(16) "TIME :  18:03:23"
Acquired access @TIME :  18:03:23:
Read from SHM : string(16) "TIME :  18:03:23"
```

Console 2 :

```
Putting into SHM : string(16) "TIME :  18:03:25"
Acquired access @TIME :  18:03:33:
Read from SHM : string(16) "TIME :  18:03:25"
```

Working with a message queue

Server :

```
<?php
$key = ftok("/home/andrey/test/phpconf/mqueue.php", 'R');
$queue = msg_get_queue($key, 0666);
$message = NULL;
$err = NULL;
msg_receive($queue, 1/*desired*/, $real_type, 16384, $message, 1/*ser*/,0,$err)
echo "Received : ";var_dump($message);
echo "Error code : ";var_dump($err);
?>
```

Client :

```
<?php
$key = ftok("/home/andrey/test/phpconf/mqueue.php", 'R');
$queue = msg_get_queue($key, 0666);
$message = "Hello World!";
$err = 0;
echo "Sending :";var_dump($message);
msg_send($queue, 1/*msg_type*/, $message, 1/*ser*/, TRUE,$err);
var_dump($err);
?>
```

Output :

```
Console 1 : andrey@poohie:~/test/phpconf> php mqueue.php
Received : string(12) "Hello World!"
Error code : int(0)

Console 2 : andrey@poohie:~/test/phpconf> php mqueue_client.php
Sending :string(12) "Hello World!"
int(0)
```

ext/shmop example (0)

```
<?php
// Create 100 byte shared memory block
$shm_id = shmop_open(ftok(__FILE__,'R'), "c"/*mode*/, 0644/*rights*/, 100/*size*/);
if (!$shm_id) {
    echo "Couldn't create shared memory segment\n";exit;
}
// Get shared memory block's size
$shm_size = shmop_size($shm_id);
echo "SHM Block Size: " . $shm_size . " has been created.\n";
// Lets write a test string into shared memory
$shm_bytes_written = shmop_write($shm_id, "my shared memory block", 0);
if ($shm_bytes_written != strlen("my shared memory block")) {
    echo "Couldn't write the entire length of data\n";exit;
}
// Now lets read the string back
$my_string = shmop_read($shm_id, 0/*start*/, $shm_size/*count*/);
if (!$my_string) { echo "Couldn't read from shared memory block\n"; exit; }
echo "The data inside shared memory was: " . $my_string . "\n";
//Now lets delete the block and close the shared memory segment
if (!shmop_delete($shm_id)) {
    echo "Couldn't mark shared memory block for deletion.";exit;
}
shmop_close($shm_id);
?>
```

Output :

```
andrey@poohie:~/test/phpconf> php shmop.php
SHM Block Size: 100 has been created.
The data inside shared memory was: my shared memory block
```

ext/shmop example (1) : semaphore emulation

```
<?php
$key = ftok(__FILE__, "K");
error_reporting(0);
do {
     $sem_id = shmop_open($key, "n", 0644, 10);
     usleep(100);
} while ($sem_id === false);
error_reporting(E_ALL);
echo "Entered critical section at :".strftime("%H:%M:%S\n", time());
sleep(5);
shmop_delete($sem_id);
echo "Exited critical section at  :".strftime("%H:%M:%S\n", time());
?>
```

Output :

Console 1:

```
andrey@poohie:~/test/phpconf> php shmop2.php
Entered critical section at :16:50:41
Exited critical section at  :16:50:46
```

Console 2:

```
andrey@poohie:~/test/phpconf> php shmop2.php
Entered critical section at :16:50:46
Exited critical section at  :16:50:51
```

pecl/memcache example

```
<?php
$memcache =
     memcache_connect('localhost', 11211);
if ($memcache) {
     $memcache->set("str_key",
          "String to store in memcached");
     $memcache->set("num_key", 123);

     $object = new StdClass;
     $object->attribute = 'test';
     $memcache->set("obj_key", $object);

     $array = Array('assoc'=>123, 345, 567);
     $memcache->set("arr_key", $array);

     var_dump($memcache->get('str_key'));
     var_dump($memcache->get('num_key'));
     var_dump($memcache->get('obj_key'));
     var_dump($memcache->get('arr_key'));
} else {
     echo "Connection to memcached failed";
}
?>
```

Output :

```
andrey@poohie:~> php example.php
string(28) "String to store in memcached"
string(3) "123"
object(stdClass)#3 (1) {
  ["attribute"]=>
  string(4) "test"
}
array(3) {
  ["assoc"]=>
  int(123)
  [0]=>
  int(345)
  [1]=>
  int(567)
}
```

Using semaphores/shared memory in a OO way

- Procedural way is makes the code larger and more error prone.
- With OO semaphore, one has only the P and V operations visible to the programmer :
  - *acquire()*
  - *release()*.
- With OO shared memory, one uses only
  - *getVar()*
  - *putVar()*
- The parameters are passed during instantiation.

OO semaphores : the class

```
class Shm_Semaphore extends Shm_SharedObject {
    var $_sem_name      = NULL;
    var $_sem_id        = NULL;
    var $_max_acquire   = NULL;
    var $_perm          = NULL;

    function Shm_Semaphore($sem_name, $max_acquire = 1, $perm='666') {
        parent::Shm_SharedObject();
        $this->_sem_name = $this->_memSegSpot(substr($sem_name,0,4));
        $this->_max_acquire = $max_acquire;
        $this->_perm = $perm;
    }// Shm_Semaphore

    function acquire() {
        $this->_sem_id = sem_get($this->_sem_name, $this->_max_acquire,
                                 OctDec($this->_perm));//only matters the first  time
        if (!sem_acquire($this->_sem_id)) {
            printf("cannot acquire semaphore<br>\n");
            return false;
        }// if
        return true;
    }// acquire

    function release() {
        return sem_release($this->_sem_id);
    }// release
}// class Shm_Semaphore
```

OO semaphores : example

```
<?php
 require 'shm.php';
 $sem = new Shm_Semaphore("test", 2); // semaphore's name is "test"
                  // max_acquire is 2. All processes can use
                  // the semaphore
 $sem->acquire();
 printf("Entered section at : %s\n", strftime("%D %T", time()));
 sleep(5);
 printf("Exited  section at : %s\n", strftime("%D %T", time()));
 $sem->release();
?>
```

Output :

Console 1:

```
andrey@poohie:~/test/phpconf> php sem1.php
Entered section at : 04/25/04 17:00:32
Exited  section at : 04/25/04 17:00:37
```

Console 2:

```
andrey@poohie:~/test/phpconf> php sem1.php
Entered section at : 04/25/04 17:00:33
Exited  section at : 04/25/04 17:00:38
```

OO shared memory : the class

```
class Shm_Var extends Shm_SharedObject {
    var $_debug         = false;
    var $_key           = 1;
    var $_shm_name      = NULL;
    var $_shm_id        = NULL;
    var $_memory_size   = NULL;
    var $_perm          = NULL;

    function Shm_Var($shm_name, $memory_size, $perm) {
        $this->_shm_name = (int)$this->_memSegSpot(substr($shm_name,0,4));
        $this->_debug && var_dump($this->shm_name);
        $this->_memory_size = $memory_size;
        $this->_perm = $perm;
        $this->_shm_id =shm_attach($this->_shm_name,$this->_memory_size,OctDec($this->_perm));
    }// Shm_Var

    function getVar() {
        return @shm_get_var($this->_shm_id, $this->_key);
    }// getVar

    function putVar($val) {
        return shm_put_var($this->_shm_id, $this->_key, $val);
    }// putVar
}// class Shm_Var
```

OO shared memory : example

```
<?php
 require 'shm.php';
 //allocating 1024 bytes for the variable
 $shm_var = new Shm_Var("acm3", 1024, "666");
 if (!($str = $shm_var->getVar())) {
   $str = strftime("First script started at : %H:%M:%S\n", time());
   $shm_var->putVar($str);
 }
 echo "From memory : ".$str;
 echo "Current time: ".strftime("%H:%M:%S\n", time());
?>
```

Output :

```
andrey@poohie:~/test/phpconf> php shm1.php
From memory : First script started at : 17:23:58
Current time: 17:23:58
andrey@poohie:~/test/phpconf> php shm1.php
From memory : First script started at : 17:23:58
Current time: 17:24:01
```

OO protected shared memory : the class

```
class Shm_Protected_Var {
  var $_debug       = false;
  var $_sem         = NULL;
  var $_shm_var     = NULL;
  var $_cached_val  = NULL;
  var $_in_section  = false;

  function Shm_Protected_Var($name, $size) {
   $sname = substr($name,0,4);
   $this->_sem=&new Shm_Semaphore($sname,1);
   $this->_shm_var=&new Shm_Var($sname,$size);
  }// Shm_Protected_Var

  function startSection() {
    if ($this->_in_section === true) {
      printf("Already in critical section\n");
      return false;
    }// if
    $this->_sem->acquire();
    $this->_in_section = true;

    return true;
  }// startSection

  function endSection() {
    if ($this->_in_section === false) {
      printf("Not in critical section\n");
      return false;
    }// if
    $this->_in_section = FALSE;
    $this->_sem->release();

    return true;
  }// endSection

  function setVal($val) {
    if ($this->_in_section === false) {
      printf("Not in critical section\n");
      return false;
    }// if
    $this->_cached_val = $val;
    $this->_shm_var->putVar($this->_cached_val);
    return true;
  }// setVal

  function getVal() {
    if ($this->_in_section === false) {
      printf("Not in critical section\n");
      return false;
    }// if
    return $this->_shm_var->getVar();
  }// getVal
}// class Shm_Protected_Var
```

OO protected shared memory: example

```
<?php
  require 'shm.php';

  $guarded_var = new Shm_Protected_Var("acm4", 1024);//allocating 1024 bytes
  $guarded_var->startSection();
  $s = sprintf("putVal() at : %s\n",microtime());
  echo $s;
  sleep(5);
  $guarded_var->setVal($s);
  sleep(5);
  echo $guarded_var->getVal();
  $guarded_var->endSection();
?>
```

Output :

Console 1:

```
andrey@poohie:~/test/phpconf> php shm2.php
putVal() at : 0.99202900 1082906968
putVal() at : 0.99202900 1082906968
```

Console 2:

```
andrey@poohie:~/test/phpconf> php shm2.php
putVal() at : 0.99968900 1082906978
putVal() at : 0.99968900 1082906978
```

OO memory queue (0)

- Less details exposed to the programmer
- Still quite powerful since all things that can be done in a procedural way are possible by using the OO memory queue
- Less and more structured code which is easier to read and less error-prone.

OO memory queue (1) : API

- ::__construct($shm_name, $size = 16384, $perm = '666')
- ::setReceiveOptions($options)
- ::setSendOptions($options)
- ::send($message, $options = array())
- ::receive($options = array())
- ::getRecvMsg()
- ::getRecvMsgType()
- ::getErrorCode()

OO memory queue : example

Server.php :

```
<?php
require_once 'shm.php';

$i = -1;
$jobs = $results = array();
while (++$i<10) {
  $jobs[$i] = array($i, $i);
}
$i = 0;

$q = new Shm_Message_Queue('ahp1');
$jobs_q = new Shm_Message_Queue('jobs');
do {
  $res = $q->receive();
  if ($res) {
    dump("Received:", $q->getRecvMsg());
    $worker_pid = $q->getRecvMsg();
    if (is_int($worker_pid) && $i < 10) {
      $jobs_q->send($jobs[$i],
         array('message_type'=>$worker_pid));
      dump("Sending:", $jobs[$i]);
      $i++;// next chunk
    } else if (is_array($worker_pid)) {
      $results[$worker_pid[0]] = $worker_pid[1];
    }
  } else {
    echo "error:\n";
    var_dump($q->getErrorCode());
  }
  if (count($results) == 10) {
    echo "Ending\n"; break;
  }
} while (1);
var_dump($jobs);
var_dump($results);

function dump($msg, $v) {
  echo $msg."\n";
  var_dump($v);
  echo str_repeat("-", 20)."\n";
}?>
```

---

client.php :

```
<?php
require_once 'shm.php';

$q = new Shm_Message_Queue('ahp1');
$jobs_q = new Shm_Message_Queue('jobs');

$q->send((int) getmypid());
$jobs_q->receive(array(
         'desired_message_type' =>getmypid(),));
$job = $jobs_q->getRecvMsg();
$sum = $job[0] + $job[1];
$q->send(array(getmypid(), $sum));
var_dump($job, $sum);
echo "End.\n";
?>
```

Use case 1 : overload protection

- Why?
  - sites with high number of hits
  - cases with scripts that uses quite a lot of resources or are long running
- How to do it ?
  - using database
  - disk file (with locking)
  - using shared memory

Use case 1 : code

Overall overload protection :

```
<?php
  require 'shm.php';

  $protector =
      new Shm_Load_Protector("bgdt",200);
  //max 200 scripts using bgdt
  if (!$protector->increaseLoad()) {
    /* There are 200 scripts running */
    include 'high_load.html';
    exit;
  }
               register_shutdown_function
("shtd_function");
  ....
  ....
  ....
  function shtd_function() {
    global $protector;
    $protector->decreaseLoad();
  }
?>
```

Duscrimination between registered and non-registered users :

```
<?php
require 'consts.php';
require 'shm.php';

if (!empty($_SESSION['user_id'])) {
  $code_name   = "1log";
  $max_scripts = MAX_PROCESSES_1LOG;
} else {
  $code_name   = "0log";
  $max_scripts = MAX_PROCESSES_0LOG;
}
$protector = new Shm_Load_Protector($code_name,
                     $max_scripts);
if (!$protector->increaseLoad()) {
  include 'high_load.html';
  exit;
}
register_shutdown_functio("shtd_function");
....
....
....
function shutdown_function() {
  global $protector;
  $protector->decreaseLoad();
}
?>
```

Use case 2 : persistent data

- Good for cases when extracting data from different resources takes time or cache access time have to be low (no disk caching).
- Example : pre-parsed XML configuration file. The config file rarely changes -> put into shared memory on first script load.
- Get the configuration from the shared memory on every next request.
  Note : deserialization needs some time.
- If you cache strings then better use ext/shmop since it does not serialize the content.

Use case 2 : XML example

```
<?php
require_once 'shm.php';
function rec($obj) {
  $ret_val = array();
  foreach ($obj as $k => $v) {
    if (($tmp = rec($v)) === NULL) {
      list(,$vv) = each($v);
      if (array_key_exists($k, $ret_val)) {
        $ret_val[$k][0] = $ret_val[$k];
      }
      if (is_array($ret_val[$k]))
        $ret_val[$k][] = $vv;
      else
        $ret_val[$k] = $vv;
    } else {
      if (array_key_exists($k, $ret_val)
        && !(is_array($ret_val[$k])
        && array_key_exists(0, $ret_val[$k]))){
          $ret_val[$k] = array($ret_val[$k]);
      }// if
      if (is_array($ret_val[$k]))
        $ret_val[$k][] = $tmp;
      else
        $ret_val[$k] = $tmp;
    }// if
  }// foreach
  if (!count($ret_val)) return NULL;
  return $ret_val;
}// rec
```

```
$shm = new Shm_Protected_Var("cff2", 16384,
           "666");
$shm->startSection();
if (!($configuration = $shm->getVal())) {
  echo "Not cached\n";
  $xml = simplexml_load_file('server.config');

  $configuration = rec($xml);
  $shm->setVal($configuration);
} else {
  echo "Cached\n";
}
$shm->endSection();

var_dump($configuration);

?>
```

Use case 2 : XML example : outcome

- Caching in shared memory with semaphore locking was about ~4x faster
- Pre-caching and no locking at later stage gives more ~2x performance boost.
- Total speed-up can be up to 10x – this saves minutes/hours of CPU time on busy machines.

Use case 3: logging

- Logging in applications help during debugging when a debugger cannot be used or when an event has ocurred. Big amounts of log data can save you.
- Logging to disk is more reliable but is slower.
- Log the data you need to shared memory or distribute it in the cluster with pecl/memcache

Questions?

I am reachable at :

andrey_php_net

This presentation is available at :

http://andrey.hristov.com/projects/php_stuff/pres/

memcached :

http://www.danga.com/memcached/

pecl/memcache :

http://pecl.php.net/package/memcache