

Tehnici Php: Manipularea proceselor

Tudor Popa

Web2.0 Jedi

Server + Client Side Efficiency Yoda Wannabe

Semantic Web Jedi

Wixi.com Jedi

Lucrul cu procese in Php(contra):



- “Cam greu de implementat pe WAMP!”
- “De ce sa ma complic cu procese cand pot face totul din acelasi script?!”
- “Exista riscul folosirii excesive a memoriei din cauza aparitiei mai multor instante!!!”
- “Cum pot sti daca ruleaza sau nu?!”
- “Mai bine ma folosesc de CRON!”
- “Daca ruleaza la nesfarsit si nu il pot stapani??”
- “Credeam ca am scapat de procese cand am terminat-o cu C-ul!”
- “Procese?! Ba, credeam ca suntem la GeekMeet, nu la tribunal...”

Modalitati de abordare

PCNTL:

Respecta stilul Unix in:

- Crearea proceselor
- Executia programului
- Manipularea semnalelor(cu ajutorul “ticks”-urilor)
- Terminarea proceselor
- Semantica

Mic exemplu cu fork

```
<?php
    // Facem fork
    $pid = pcntl_fork();
    if ($pid == -1) {
        die("Eroare la fork()");
    } else if ($pid) {
        // Suntem in Parinte
        print "Parintele aici, PID-ul fiului este: $pid\n";
    } else {
        // Suntem in Fiu
        print "Fiul aici!\n";
    }
}
```

Mic exemplu cu tratarea semnalelor

```
<?php
declare(ticks=1);
// Atribuim handlerul de semnale
pcntl_signal(SIGTERM, "sig_handler");
pcntl_signal(SIGHUP, "sig_handler");
// Procesul propriu-zis
while (1) {
    // Executa ceva
}
// Handlerul de semnale
function sig_handler($signo) {
    switch ($signo) {
        case SIGTERM:
            // Semnal de shutdown
            exit;
            break;
        case SIGHUP:
            // Semnal de restart
            break;
        default:
            // Alte semnale
    }
}
```

Rulare “manuala” prin Shell sau CRON:

- Administrare la nivel de consola
- Posibilitate de “supraveghere” (top, htop, supervise, etc.)
- Avantajul executarii unui singur task
- Avantajul rularii continue(daca se foloseste rulara prin CRON)
- Easy to strace

Mic exemplu de proces pornit din Shell

processTest.php

```
<?php
$evt = new UserEventsHandler();
while(1) {
    if ($evt->checkDb()) {
        $evt->handleEvents();
        sleep(2);
    }
    else {
        exit();
    }
}
```

Consola:

```
Cd <path_to_file>; php5 processTest.php
```

Probleme legate de exemplul anterior

- Posibilitatea aparitiei mai multor instante(in cazul rularii prin CRON)

Rezolvare:

```
// Verificarea pid-ului
```

```
function checkPid($pid) {  
    $cmd = "ps -o stat --no-headers $pid";  
    exec($cmd, $output, $result);  
    if(count($output) >= 1){  
        return true;  
    }  
    return false;  
}
```

Output consola

```
$ ps -o stat --no-headers 6730
```

```
S+
```

Folosind functia anterioara, verificam pid-ul salvat intr-un fisier

```
function checkPidFromFile() {
    if (file_exists(PATH_PID)) {
        $fh = fopen(PATH_PID, 'r');
        $data = fgets($fh);

        if ($data != null) {
            $pid = intval($data);
            if (checkPid($pid)) {
                fclose($fh);
                return true;
            }
        }
        else {
            return false;
        }
    }
    else {
        return true;
    }
}
```

Funcția care scrie pid-ul curent într-un fișier

```
function writePid() {  
    $pid = getmypid();  
    $fh = fopen(PATH_ PID, 'w+');  
    fwrite($fh, strval($pid));  
    fclose($fh);  
}
```

- Lipsa comunicarii(no output, “Silence is golden”, dar nu tot timpul)

Rezolvare:

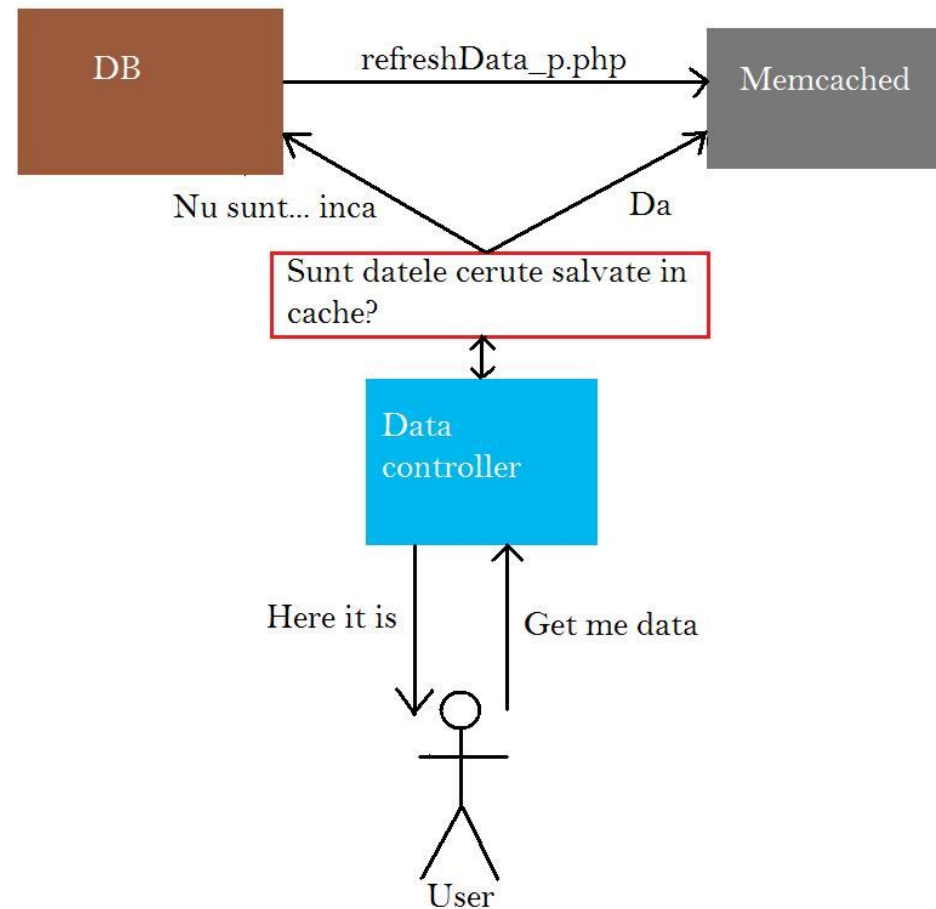
```
function talk($pid) {  
    if (isset($_GET['status'])) {  
        // Afisam date despre proces  
        ProcessHandler::tellProcessStatus($pid);  
        die();  
    }  
}
```

Varianta imbunatatita

```
// Verificam pid-ul din fisier
talk($pid);
if (!checkPidFromFile()) {
    // In cazul in care nu exista/nu ruleaza, il scriem
    writePid();
    $evt = new UserEventsHandler();
    while(1) {
        if ($evt->checkDb()) {
            $evt->handleEvents();
            sleep(2);
        }
        else {
            exit();
        }
    }
}
```

Cazuri favorabile utilizarii proceselor

- Data caching(exemplu Memcached)



Scenariu:

1. Userul cere date de la data controller
2. Data controller-ul cauta in cache
 - a. Daca rezultatele exista, le returneaza
 - b. Daca nu, le ia din baza de date (~~iar apoi le introduce in cache~~) si le returneaza
3. Scriptul refreshData_p.php, rulat ca proces prin CRON, introduce continuu in cache date din baza de date

Exemplu pentru modelul anterior:

refreshData_p.php

```
<?php
```

```
// Folosim functiile pt verificarea pid-ului si
```

```
talk
```

```
require_once("processFunctions.php");
```

```
talk($pid);
```

```
// Verificam pid-ul din fisier
```

```
if (!checkPidFromFile()) {
```

```
    // In cazul in care nu exista/nu ruleaza, il scriem
```

```
    writePid();
```

```
    while(1) {
```

```
        if (QueryChecker::isNewDataInQueries()) {
```

```
CacheHandler::addToCache(QueryChecker::returnNewQueryData);
```

```
    sleep(2);
```

```
    }
```

```
    else {
```

```
        exit();
```

```
    }
```

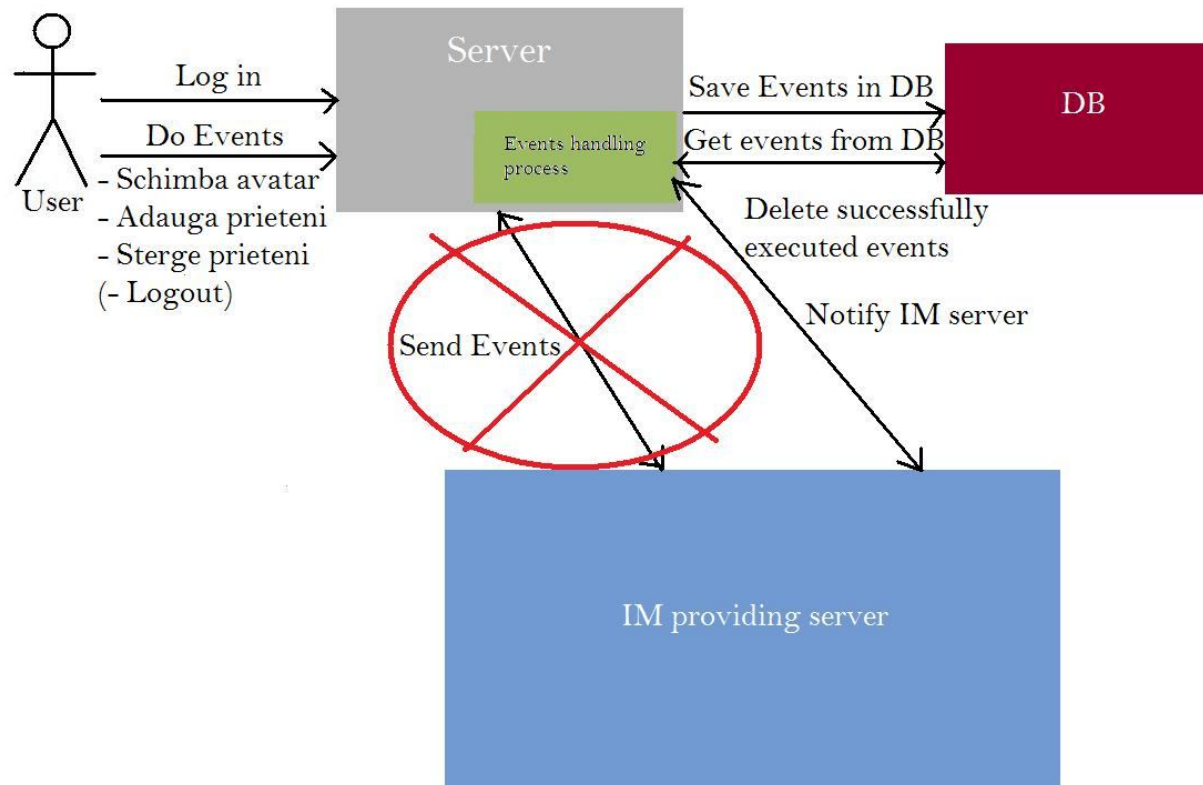
```
    }
```

```
    }
```

Beneficii:

- Mentinerea continua a datelor noi in cache
- Izolare in cazul erorilor de server/umane
- Update la interval de timp scazut datorita executarii unui singur task

- External server calls(exemplu community im)



Scenariu:

1. Userul se logheaza in site
2. Executa evenimente:
 - a. Schimba avatar-ul
 - b. Schimba nickname-ul
 - c. Schimba informatii personale
 - d. Adauga prieteni
 - e. Sterge prieteni
 - f. Log-out
3. ~~Serverul trimite evenimentele user-ului la serverul de IM si is continua taskurile~~
4. Serverul salveaza evenimentele in baza de date
5. Scriptul checkIMEvents_p.php, rulat ca proces prin CRON, selecteaza ultimele evenimente din baza de date
6. Le trimite la serverul de IM si le sterge pe cele efectuate cu succes din baza de date

Exemplu pentru modelul anterior:

checkIMEvents_p.php

```
<?php
```

```
// Folosim functiile pt verificarea pid-ului si
```

```
talk
```

```
require_once("processFunctions.php");
```

```
talk($pid);
```

```
// Verificam pid-ul din fisier
```

```
if (!checkPidFromFile()) {
```

```
    // In cazul in care nu exista/nu ruleaza, il scriem
```

```
    writePid();
```

```
    $evt = new IMEvents();
```

```
    while(1) {
```

```
        // Verificam conexiunea la db
```

```
        if ($evt->checkDb()) {
```

```
            // Trimitem evenimentele apoi le stergem
```

```
            $evt->handleEvents();
```

```
            sleep(2);
```

```
        }
```

```
    } else {
```

```
        exit();
```

```
    }
```

```
}
```

```
}
```

Beneficii:

- Notificarea rapida a serverului IM
- Efectuare aproape in timp real a evenimentelor datorita rularii in paralel
- Izolare in cazul erorilor de server
- Contorizare a evenimentelor

Concluzii:



- O alternativa eficienta a metodelor de programare secventiala
- Posibilitatea izolarii unui task care necesita mai mult timp de executie
- Implementare usoara, 90% in mediul Php
- Oferă modalitati de supraveghere constanta si facila
- Posibilitatea terminarii unui task esuat
- Suport maxim pe sistemele de operare Unix/Linux
- Avantajul rularii continue(actualizare continua a datelor)
- Respecta ideea de “unu’ si bun”