DOI: 10.58168/CISMP2024_672-677
УДК 004.9

РАБОТА С СИГНАЛАМИ ЧЕРЕЗ МНОГОЗАДАЧНОСТЬ И ФОРКИ ПРОЦЕССОВ С ИСПОЛЬЗОВАНИЕМ PCNTL_FORK В PHP

А.А. Андрюшин, А.А. Псарев
artem.psarev4849@gmail.com

ФГБОУ ВО «Воронежский государственный лесотехнический университет имени Г.Ф. Морозова»

Аннотация. В статье рассматривается использование функции pcntl_fork в языке PHP для создания многозадачных приложений и работы с сигналами. Подробно описаны принципы работы функции pcntl_fork, возможности её применения для создания дочерних процессов и оптимизации работы серверных приложений. Статья направлена на повышение производительности PHP-приложений при обработке параллельных задач и асинхронных событий.

Ключевые слова: PHP, многозадачность, pcntl_fork, процессы, сигналы.

WORKING WITH SIGNALS THROUGH MULTITASKING AND PROCESS FORKING USING PCNTL_FORK IN PHP

A.A. Andryushin, A.A. Psarev
artem.psarev4849@gmail.com

Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. This article discusses the use of the pcntl_fork function in PHP for creating multitasking applications and handling signals. It provides a detailed explanation of how the pcntl_fork function works, its potential for creating child processes, and optimizing server application performance. The article aims to improve the efficiency of PHP applications in processing parallel tasks and asynchronous events.

Keywords: PHP, multitasking, pcntl_fork, processes, signals.

Введение

PHP, как мною было выяснено, будучи в первую очередь языком для веб-разработки, в последние годы значительно увеличил свои возможности из-за внедрения функций PCNTL (process control). Эти функции позволяют использовать многозадачность процессов, что открывает перед разработчиками новые возможности. Предположим, когда нужно выполнять несколько задач одновременно, особенно в серверных приложениях, где важно обрабатывать запросы без задержек. Одной из таких ключевых функций является pcntl_fork. она дает возможность создавать дочерние процессы, которые работают параллельно с основным. Этот подход я думаю особенно понадобиться и будет

© Андрюшин А. А., Псарев А. А., 2024

полезный для распределения вычислительных задач и асинхронного выполнения этих задач.

Когда мы работаем с веб-приложениями или сложными задачами на сервере, наверное, часто возникает необходимость обрабатывать сразу несколько запросов или операций. В этом случае простого выполнения одного процесса за другим недостаточно, и параллельное выполнение становится настоящим помощником для разработчиков. Именно здесь на сцену выходит pcntl_fork.

Основы pcntl_fork

pcntl_fork в PHP даёт нам возможность создать новый дочерний процесс, который является копией родительского процесса, что особенно важно. После вызова этой функции оба процесса продолжают работу независимо друг от друга. Как если бы вы сделали копию себя и поручили каждому из своих клонов выполнять разные задачи, удобно правда? Родительский процесс продолжает выполнение программы, а дочерний может, к примеру, взять на себя другую часть работы.

Когда вызывается pcntl_fork, происходит следующее:

- родительский процесс получает идентификатор дочернего процесса (PID), который больше нуля.
- Дочерний процесс получает значение 0, что позволяет ему понять, что он работает в режиме дочернего процесса.
- Если при создании дочернего процесса произошла ошибка, функция возвращает нас на шаг назад, что позволяет программе обработать этот сбой.

```php
$pid = pcntl_fork();

if ($pid == -1) {
    die('Не удалось создать дочерний процесс');
} elseif ($pid) {
    // Родительский процесс
    echo "Это родительский процесс, PID дочернего: $pid\n";
    pcntl_wait($status); // Ожидание завершения дочернего процесса
} else {
    // Дочерний процесс
    echo "Это дочерний процесс\n";
    sleep(5); // Эмулируем длительную задачу
    exit(0);  // Завершение дочернего процесса
}
```

Рисунок 1 – Пример работы с pcntl_fork

Этот кусочек кода показывает нам базовое создание процесса с использованием pcntl_fork. родительский процесс получает PID дочернего процесса, а дочерний выполняет свою задачу отдельно от него.

Почему это важно по моему мнению?

Использование pcntl_fork в php помогает разделять задачи и распределят нагрузку, что точно в наши дни актуально и важно для серверных приложений. Представим, что вы создаете веб приложение, которое обрабатывает очень-очень много запросов от пользователей. Если каждый вопрос будет выполняться один за одним, пользователи могут столкнуться с задержками и лагами. Но если разделить их между разными процессами, время отклика будет гораздо быстрее.

Многозадачность — это не просто удобство. Это необходимость в мире высоконагруженных серверных приложений. Как я выяснил и изучил и использовал здесь, pcntl_fork даёт возможность обрабатывать несколько запросов одновременно, что даёт возможность не ждать завершения прошлых задач. Это очень удобно, когда вам нужно наладить стабильную работу приложения под огромной нагрузкой.

Пример с несколькими процессами

Теперь я хочу рассмотреть и показать более сложный пример, где создается несколько дочерних процессов для выполнения различных задач. Это, я надеюсь, поможет лучше понять, как pcntl_fork может быть использован для создания многозадачного приложения.

```php
for ($i = 0; $i < 3; $i++) {
    $pid = pcntl_fork();

    if ($pid == -1) {
        die("Ошибка при создании дочернего процесса\n");
    } elseif ($pid) {
        // Родительский процесс
        echo "Родитель создал процесс с PID: $pid\n";
    } else {
        // Дочерний процесс
        echo "Дочерний процесс с PID: " . getmypid() . "\n";
        sleep(2); // Выполнение задачи
        exit(0);
    }
}
```

Рисунок 2 – Создание дочерних задач

В этом моём примере создаются три дочерних процесса. Каждый процесс выполняет свою задачу независимо от остальных. Родительский процесс делает выполнение, не для завершения дочерних. Этот некий подход удобно использовать в ситуациях, когда нужно обработать очень много запросов или задач одновременно.

Многозадачность и контроль процессов

Использование pcntl_fork эффективно для многозадачных приложений, где требуется выполнять несколько операций одновременно. Например, я думаю, что это можно использовать в серверных приложениях, это может быть обработка запросов от пользователей или выполнение параллельных вычислений.

Пример обработки нескольких задач одновременно

Предположим, что у нас есть задача обработки огромного объема данных, и мы хотим разделить эту работу между несколькими процессами.

```php
$tasks = ['task1', 'task2', 'task3'];

foreach ($tasks as $task) {
    $pid = pcntl_fork();

    if ($pid == -1) {
        die("Не удалось создать процесс для задачи $task\n");
    } elseif ($pid) {
        // Родительский процесс
        pcntl_wait($status); // Ожидание завершения дочернего процесса
    } else {
        // Дочерний процесс
        echo "Выполняем задачу: $task\n";
        sleep(2); // Эмуляция выполнения задачи
        exit(0);
    }
}
```

Рисунок 3 – Разделение работы

Этот пример показывает, как можно разделить обработку задач между несколькими дочерними процессами. Каждый процесс отвечает за выпполнение своей задачи, а родительский процесс ожидает завершения.

Когда стоит использовать pcntl_fork?

Хотя как я выяснил, php не является языком, изначально предназначенным для создания многозадачных приложений, использование pcntl_fork открывает перед разработчиками новые возможности. Вот несколько сценариев, когда применение этой функции, по моему мнению, более оправданно:

- обработка нескольких клиентских запросов на сервере. Например, каждый запрос может обрабатываться в отдельном процессе, что уменьшит задержки.

- Параллельная обработка данных. В случае, если у вас есть несколько незавяназзных между собой задач, их можно ловко разделить между дочерними процессами для ускорения выполнения программы.

Фоновые процессы. Если вам нужно, чтобы какая-то задача выполнялась в фоновом режиме, вы можете использовать pcntl_fork для ее выполнения параллельно с основной программой.

Заключение

pcntl_fork — это мощный инструмент для создания многозадачных приложений на pHp. Хотя сам язык чаще всего используют для веб разработки, но при этом с помощью этой функции можно воплотить в реальность параллельное выполнение задач, что я думаю очень важно для серверных приложений. Примеры, рассмотренные в моей этой статье, показывают нам, как легко можно использовать pcntl_fork для релиза дочерних процессов и практичного управления задачами.

Я пришёл к выводу, что, применяя многозадачность, вы можете улучшить производительность своих приложений. Функция pcntl_fork помогает не только справляться с мега-большими нагрузками, но и облегчить использование кода, делая приложения более быстрыми и гибкими. Поэтому я думаю, если перед вами стоит задача обработки нескольких параллельных операций, рассмотрите возможность использования pcntl_fork – она может стать вашим надежным помощником.

Список литературы

1. Боярский, А.Е. Программирование на PHP: руководство для профессионалов / А.Е. Боярский. – Москва: Вильямс, 2020. – 432 с.
2. Ларсен, Дж. Архитектура серверных приложений и многозадачность / Дж. Ларсен, П. Мэтьюс. – Санкт-Петербург: Питер, 2019. – 384 с.
3. Петроченков, А.И. Серверное программирование и многозадачность в PHP / А.И. Петроченков. // Программирование и компьютерные науки. – 2022. – №5. – С. 87–95.
4. Зольников К.В., Гамзатов Н.Г., Евдокимова С.А., Потапов А.В., Допира Р.В., Кучеров Ю.С., Яночкин И.Е., Стоянов С.В., Плотников А.М. Моделирование процессов в полупроводниковых структурах при радиационном воздействии // Моделирование систем и процессов. – 2022. – Т. 15, № 3. – С. 106-127.
5. Жаксыбаев Д.О., Бакиев М.Н. Алгоритмы классификации текстовых документов с учетом близости в признаковом пространстве // Моделирование систем и процессов. – 2022. – Т. 15, № 1. – С. 36-43.
6. Журавлева И.В., Попова Е.А. Полупроводниковые технологии для реализации радиационно-стойких СБИС // Моделирование систем и процессов. – 2022. – Т. 15, № 1. – С. 44-52.

References

1. Boyarsky, A.E. Programming in PHP: A Guide for Professionals / A.E. Boyarsky. – Moscow: Williams, 2020. – 432 p.

2. Larsen, J. Architecture of Server Applications and Multitasking / J. Larsen, P. Matthews. – Saint Petersburg: Piter, 2019. – 384 p.

3. Petrochenkov, A.I. Server Programming and Multitasking in PHP / A.I. Petrochenkov. // Programming and Computer Science. – 2022. – №5. – pp. 87–95.

4. Zolnikov, K.V., Gamzatov, N.G., Evdokimova, S.A., Potapov, A.V., Dopira, R.V., Kucheryov, Y.S., Yanochkin, I.E., Stoyanov, S.V., Plotnikov, A.M. Modeling Processes in Semiconductor Structures under Radiation Exposure // Modeling Systems and Processes. – 2022. – Vol. 15, №3. – pp. 106-127.

5. Zhaksabayev, D.O., Bakiev, M.N. Algorithms for Text Document Classification Considering Proximity in Feature Space // Modeling Systems and Processes. – 2022. – Vol. 15, №1. – pp. 36-43.

6. Zhuravleva, I.V., Popova, E.A. Semiconductor Technologies for Implementing Radiation-Resistant Integrated Circuits // Modeling Systems and Processes. – 2022. – Vol. 15, №1. – pp. 44-52.