========================PHP Fiber Cheat sheet============================

Here's a PHP Fiber cheat sheet for different types of multitasking. This includes basic usage, handling multiple tasks, exception handling, and integration with I/O-bound operations.

PHP Fiber Cheat Sheet

1. Basic Fiber Operations

Create a Fiber
php
Copy code
```
$fiber = new Fiber(function() {

    // Your code here

});
```

-

Start a Fiber
php
Copy code
```
$result = $fiber->start();
```

-

Suspend a Fiber
php
Copy code
```
$value = Fiber::suspend($someValue);
```

-

Resume a Fiber
php
Copy code
```
$result = $fiber->resume($someValue);
```

-

Check Fiber Status
php

Copy code
```
$fiber->isStarted();   // Returns true if the fiber has been started

$fiber->isSuspended(); // Returns true if the fiber is currently suspended

$fiber->isRunning();   // Returns true if the fiber is currently running

$fiber->isTerminated();// Returns true if the fiber has finished execution
```

-

Get Fiber Return Value
php
Copy code
```
$returnValue = $fiber->getReturn();
```

-

2. Handling Multiple Tasks

Creating Multiple Fibers
php
Copy code
```
$fiber1 = new Fiber(function() {

    // Task 1

});



$fiber2 = new Fiber(function() {

    // Task 2

});
```

-

Starting Multiple Fibers
php

Copy code
```
$fiber1->start();

$fiber2->start();
```

-

Resuming Multiple Fibers
php
Copy code
```
$result1 = $fiber1->resume();

$result2 = $fiber2->resume();
```

-

3. Exception Handling in Fibers

Catching Exceptions within a Fiber
php
Copy code
```
$fiber = new Fiber(function() {

    try {

        // Your code here

    } catch (Throwable $e) {

        echo "Caught exception: " . $e->getMessage();

    }

});
```

-

Handling Exceptions When Resuming
php
Copy code
```
try {

    $fiber->start();

    $fiber->resume();
```

```php
} catch (Throwable $e) {

    echo "Error: " . $e->getMessage();

}
```

-

4. Non-Blocking I/O with Fibers

Example: Simulating Asynchronous Sleep
php
Copy code
```php
function asyncSleep(int $seconds) {

    return new Fiber(function() use ($seconds) {

        sleep($seconds);

        return "Slept for $seconds seconds";

    });

}



$fiber = asyncSleep(3);

$fiber->start();

echo $fiber->resume();  // Output: Slept for 3 seconds
```

-

5. Cooperative Multitasking with Fibers

Pausing Execution and Resuming Later
php
Copy code
```php
$fiber = new Fiber(function() {

    echo "Task 1 started\n";

    Fiber::suspend();  // Pause here
    echo "Task 1 resumed\n";

});



$fiber->start();  // Output: Task 1 started

// Do other work here...

$fiber->resume();  // Output: Task 1 resumed
```

-

Running Multiple Fibers Cooperatively
php
Copy code

```
$fiber1 = new Fiber(function() {

    echo "Fiber 1\n";

    Fiber::suspend();

    echo "Fiber 1 Resumed\n";

});



$fiber2 = new Fiber(function() {

    echo "Fiber 2\n";

    Fiber::suspend();

    echo "Fiber 2 Resumed\n";

});



$fiber1->start();

$fiber2->start();
```

```
// Resume fibers at some later point

$fiber1->resume();

$fiber2->resume();
```

-

6. Integration with Event Loops

ReactPHP Example
php
Copy code
```
$loop = React\EventLoop\Factory::create();



$fiber = new Fiber(function() use ($loop) {

    $loop->addTimer(1, function() {

        echo "Timer fired\n";

        Fiber::suspend();

    });

    $loop->run();

    echo "Fiber resumed after timer\n";

});



$fiber->start();

$fiber->resume();  // This will be called after the timer fires
```

-

Amp Example
php

Copy code
```php
use Amp\Loop;


$fiber = new Fiber(function() {

    Loop::run(function() {

        Loop::delay(1000, function() {

            echo "1 second later\n";

            Fiber::suspend();

        });

    });

});


$fiber->start();

$fiber->resume();  // Resumes after 1 second delay
```

-

7. Advanced Fiber Operations

Combining Results from Multiple Fibers
php
Copy code
```php
$fiber1 = new Fiber(function() {

    sleep(2);

    return "Result 1";

});


$fiber2 = new Fiber(function() {
    sleep(1);

    return "Result 2";

});


$fiber1->start();

$fiber2->start();


$result1 = $fiber1->resume();

$result2 = $fiber2->resume();


echo "Combined results: $result1, $result2";  // Output: Combined
results: Result 1, Result 2
```

-

Cancelling a Fiber (Not directly supported, but can be simulated)
php
Copy code

```php
$fiber = new Fiber(function() {

    while (true) {

        // Perform task

        Fiber::suspend();

    }

});


$fiber->start();
```

```
// To "cancel", simply do not resume the fiber or exit the loop
conditionally
```

-

Summary

PHP Fibers provide a powerful mechanism for handling concurrency, allowing you to write non-blocking, asynchronous code in a more synchronous style. This cheat sheet covers the essential operations you need to manage multitasking effectively using Fibers in PHP.

By mastering these concepts, you can optimize your applications for performance, especially when dealing with multiple I/O-bound operations or other tasks that can benefit from concurrent execution.

========================Multi Tasking with PHP Fiber =======================

Handling multiple form requests concurrently using PHP Fibers can be a powerful way to optimize the performance of your application, especially when dealing with I/O-bound operations like database queries, external API calls, or file uploads. Here's how you can approach handling multiple form requests using PHP Fibers.

Scenario: Handling Multiple Form Submissions Concurrently

Imagine you have a web application where users submit multiple forms that trigger different operations, such as saving data to a database, sending emails, and making API requests. Using PHP Fibers, you can handle these operations concurrently, reducing the total execution time.

Example: Using PHP Fibers to Handle Multiple Form Requests

Let's assume you have a form where users can submit data that triggers three different operations:

1. **Saving data to a database**
2. **Sending a confirmation email**
3. **Making an API request**

Here's how you can use PHP Fibers to handle these operations concurrently:

Step 1: Setup Fibers for Each Operation

php

Copy code
```
function saveToDatabase($formData) {
    return new Fiber(function() use ($formData) {
        // Simulate a database save operation
        echo "Saving to database...\n";
        // Simulate a delay, representing a blocking I/O operation
        sleep(2);
        echo "Database save completed\n";
        return "Database ID: 123";  // Return some result
    });
}

function sendEmail($emailAddress) {
    return new Fiber(function() use ($emailAddress) {
        // Simulate sending an email
        echo "Sending email to $emailAddress...\n";
        sleep(1);
        echo "Email sent\n";
        return "Email Status: Sent";  // Return some result
    });
}

function makeApiRequest($endpoint, $data) {
    return new Fiber(function() use ($endpoint, $data) {
        // Simulate making an API request
        echo "Making API request to $endpoint...\n";
        sleep(3);
        echo "API request completed\n";
        return "API Response: Success";  // Return some result
    });
}
```

Step 2: Start Fibers Concurrently

Now, when a form is submitted, you can start these fibers concurrently:

php
Copy code
```
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $formData = $_POST['formData'];
    $emailAddress = $_POST['emailAddress'];
    $apiEndpoint = "https://api.example.com/endpoint";

    // Create and start fibers
    $dbFiber = saveToDatabase($formData);
    $emailFiber = sendEmail($emailAddress);
    $apiFiber = makeApiRequest($apiEndpoint, $formData);

    $dbFiber->start();
    $emailFiber->start();
    $apiFiber->start();

    // You can do other work here while the fibers are running

    // Resume fibers to get their results
    $dbResult = $dbFiber->resume();
    $emailResult = $emailFiber->resume();
    $apiResult = $apiFiber->resume();

    // Handle the results
    echo "Database Result: $dbResult\n";
    echo "Email Result: $emailResult\n";
    echo "API Result: $apiResult\n";

    // Send a response back to the user
    echo "All operations completed successfully!";
}
```

Explanation

- **Fibers for Each Task:** Each task (database save, email send, API request) is encapsulated in a separate fiber.
- **Starting Fibers:** All fibers are started concurrently using the `start()` method.
- **Resuming Fibers:** After starting the fibers, they are resumed to get their results once the I/O operations are complete.
- **Concurrent Execution:** Although PHP executes code synchronously by default, fibers allow you to manage multiple tasks concurrently, simulating an asynchronous environment.

Step 3: Handling Fiber Exceptions

Fibers can throw exceptions, just like regular PHP code. It’s important to handle these exceptions properly:

php
Copy code
```php
try {
    $dbFiber->start();
    $emailFiber->start();
    $apiFiber->start();

    $dbResult = $dbFiber->resume();
    $emailResult = $emailFiber->resume();
    $apiResult = $apiFiber->resume();

    echo "Database Result: $dbResult\n";
    echo "Email Result: $emailResult\n";
    echo "API Result: $apiResult\n";
} catch (Throwable $e) {
    echo "An error occurred: " . $e->getMessage();
}
```

Advantages of Using Fibers for Form Handling

1. **Concurrency:** Fibers allow you to handle multiple form submissions concurrently without blocking other operations.
2. **Improved Performance:** By avoiding blocking I/O operations, your server can handle more requests in the same amount of time.
3. **Simplicity:** Fibers allow you to write asynchronous code in a synchronous style, making it easier to maintain and debug.

Conclusion

Using PHP Fibers to handle multiple form requests concurrently can greatly improve the performance of your application, especially when dealing with I/O-bound operations. This approach enables your application to efficiently manage multiple tasks in parallel, reducing overall execution time and improving the user experience.