УДК 004.415.3
Приймачук В.О., Пех П.А.
Луцький національний технічний університет

АСИНХРОННЕ ПРОГРАМУВАННЯ В NODE.JS

В даній статті розглянуті концепції управління асинхронним потоком виконання в Node.js. Спочатку наводиться короткий опис асинхронного програмування, його переваги та недоліки, потім розглядаються основні способи організації потоку виконання: функції зворотного виклику (callback), обіцянки (promise), співпрограми(coroutine).
Ключові слова: *Асинхронне програмування, Node,js, callback, promise, deferred*.

Приймачук Василий Александрович, Пех Петр Антонович. Асинхронное программирование в Node.js. *В этой статье описываются концепции управления асинхронним потоком исполнения в Node.js.В начале приводится краткий обзор асинхронного программирования, его преимущества и недостатки, далее разбираются основные способа организации потока исполнения: функции обратного вызова (callback), обещания (promise), сопрограммы (coroutine) .*
Ключевые слова: *Асинхронное программирование, Node,js, callback, promise, deferred.*

Priymachuk Vasyl, Pekh Petro. Asynchronous programming in Node.js. *This article describes the concept of asynchronous control flow of execution in Node.js. At the beginning of a brief overview of asynchronous programming, its advantages and disadvantages, further understand the basic way of organizing the flow of execution: the callback function,, promise, coroutine.*
Keywords: *Asynchronous programming, Node,js, callback, promise, deferred.*

Вступ

Для збільшення продуктивності сучасних комп'ютерів вже недостатньо просто збільшувати тактову частоту процесора. Тим більше з кожним роком це стає все складніше і все більш неефективно. Сьогодні більшість учасників ринку апаратного забезпечення розвивають інший перспективний напрям - збільшення кількості процесорів, процесорних ядер. Для ефективного використання таких потужностей необхідно реалізовувати паралельні обчислення.

Паралельна обробка операцій - одна з найбільш складних дисциплін в області розробки програмного забезпечення [1]. Сучасні операційні системи та середовища програмування забезпечують паралельні обчислення через багатозадачність. Реалізації істинної багатозадачності можлива лише в розподілених системах.

Загальноприйнятий спосіб реалізації багатозадачності - багатопоточність. Багатопоточность дозволяє спрощувати написання програм в деяких випадках, за рахунок використання спільної пам'яті.

Основні проблеми багатозадачності [2]: голодування, гонка, інверсія пріоритету. Багатопоточність не вирішує цих питань, причому додає ще й ряд нових.

Асинхронність - інший підхід до реалізації концепції паралелізму та багатозадачності. Він не виключає використання багатопоточності, проте в більшості випадків асинхронні системи однопоточні.

Особливості асинхронного програмування полягають в тому, що результат виконання функції доступний не відразу, а через деякий час у вигляді деякого асинхронного виклику. В синхронній моделі нам не важливо коли завдання почало виконуватись і коли надійшов результат виконання цього завдання, оскільки середовище робить для нас це все одним процесом. В цьому випаду програміст контролює час отримання результату. В асинхронній моделі ці два процеси незалежні один від одного. Після запуску завдання не має потреби чекати результат. В це й час можна виконати запустити інші завдання, обробити результат попередніх завдань, результати виконання яких вже готові. Внаслідок цього порядок виконання коду є нелінійним, в будь-який момент може бути викликана будь-яка функція. Це не дозволяє легко прослідкувати хід подій в

програмі, що створює великі проблеми при відлагодженні програми. Для великих програм критично важливо контролювати всі аспекти виконання програми.

Моделі виконання програми

Для початку розглянемо дві схожі моделі для того щоб порівняти з ними асинхронну модель.

Синхронна однопоточна модель

Це найпростіший стиль програмування. Одночасно виконується тільки одне завдання, наступне завдання запускається лише після виконання попереднього. Якщо завдання виконуються в деякому порядку, то запуск наступного завдання можливий лише якщо всі попередні завдання виконались без помилок.

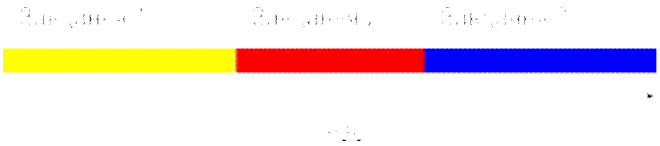

Рис. 1 - Синхронна модель

Багатопоточна модель

В багатопоточній моделі кожне завдання виконується в окремому потоці. Потоки керуються операційною системою, і у випадку наявності кількох процесорів і/або ядер, можуть виконуватись паралельно, або їх виконання може чередуватись у випадку одного процесора. Суть полягає в тому, що в потоковій моделі деталями керує операційна система, програміст думає в термінах незалежних потоків інструкцій, які можуть виконуватись одночасно. Хоч діаграма і проста, на практиці потокові програми можуть бути досить складними, оскільки необхідно координувати потоки між собою. Потокова координація та взаємодія - окрема, достатньо складна тема.

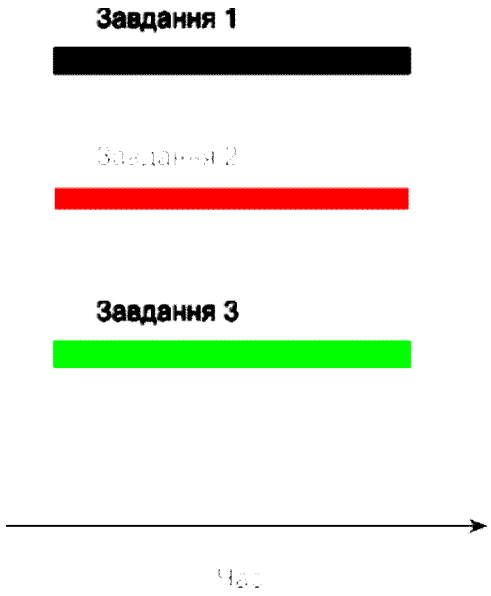

Рис. 2 - Багатопоточна модель

Асинхронна однопоточна модель

В асинхронній моделі завдання виконуються почергово в одному потоці. Це простіше, ніж у випадку потоків, оскільки програміст впевнений, що коли одне завдання виконується, в цей же час інше завдання чекає своєї черги. Хоча в однопроцесорній системі в потоковій моделі завдання все-одно будуть виконуватись почергово, програміст, що використовує потоки, має мислити в термінах кількох незалежно виконуваних процесів. Натомість однопоточна асинхронна система завжди буде чередувати виконання завдань навіть на багатопроцесорній системі.

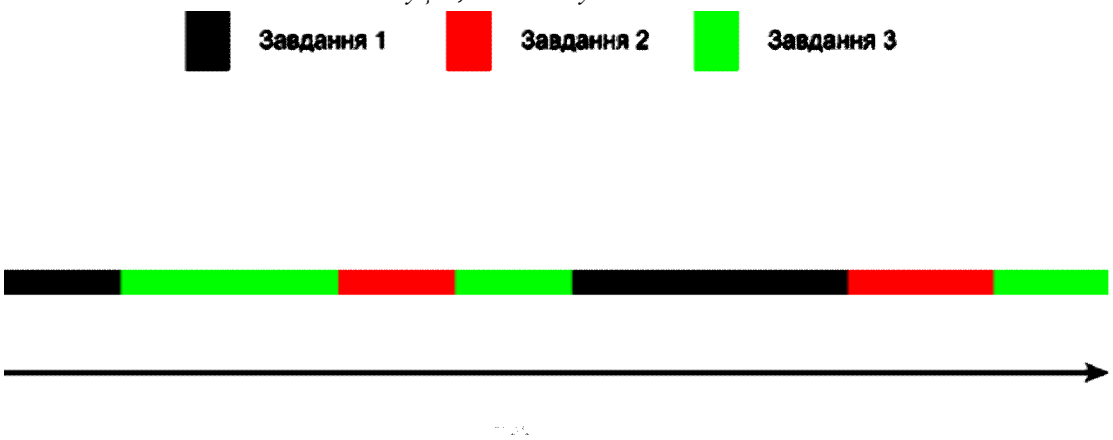

Рис. 3 - Асинхронна однопоточна модель

Існує ще одна суттєва відмінність між асинхронною та потоковою моделями. В потоковій моделі рішення про призупинення одного потоку і виконання іншого зазвичай не контролюється програмістом. Цей процес конролює операційна система, через що програміст повинен передбачати можливість зупинки процесу практично в будь-який момент. Це відрізняється від завдань асинхронної моделі, які виконуються до того моменту, доки явно не передадуть контроль іншим завданням. Ці дві моделі реалізують різні моделі багатозадачності: потокова - пріоритетну багатозадачність, асинхронна - кооперативну багатозадачність.

Відмітимо, що можливе поєднання асинхронної та потокової моделі і використання обох варіантів в одній системі.

Реалізація асинхронної моделі в Node.js

В основі асинхронної моделі Node.js стоять глобальний цикл подій (Event loop) та шаблон "реактор".

Глобальний цикл подій - це нескінченний цикл, який опитує "джерела подій" (дескриптори) на предмет появи від них якоїсь події. Опитування забезпечує бібліотека синхронного вводу/виводу libuv, яка абстрагує засоби операційної системи для роботи з неблокуючим вводом/виводом (IOCP для Windows, libev для Unix систем). Це опитування буде неблокуючим. Тобто, під час наступного витка глобального циклу подій, система проходить по всім дескрипторам, і намагається прочитати з них події: якщо такі наявні, тоді вони повертаються функцією читання в нашу систему; інакше глобальний цикл подій не буде блокуватись і чекати появи події, а відразу відповість - нових подій нема.

Подією може бути наявність нової порції даних на мережевому сокеті, зчитування нової порції даних з жорсткого диску: в загальному випадку будь-який ввід/вивід.

Другий компонент нової моделі - шаблон реактор. Зміст цього шаблону в тому, що код сервера пишеться не одним великим монолітним шматком, який виконується послідовно, а невеликими блоками, кожен з яких викликається тоді, коли відбувається подія пов'язана з цим блоком. Таким чином код - це набір сукупності блоків, завдання яких полягає в т ому, щоб реагувати на деякі події.

Переваги та недоліки асинхронної моделі

Асинхронна модель краще підходить для програм, де багато користувачів одночасно виконують деякі дії, що не навантажують процесор. Наприклад, програми, що отримують температуру з датчиків в режимі реального часу, зображення з відеокамер, пишуть нові повідомлення в чат, отримують нові повідомлення з чату тощо.

Вимога дій, які не навантажують процесор, пояснюється тим, що система має один глобальний потік подій, і, якщо в цей потік додати тяжке обчислювальне завдання, тоді система не зможе обслуговувати нових користувачів допоки це завдання не буде виконано, зникнуть всі переваги асинхронної моделі.

Тому сервера, написані на Node.js, підходять тільки для виконання завдань, які не є тяжкими обчислювальними процесами. Також вони підходять як сервери для обслуговування клієнтів з вузьким каналом даних, там де великий час відклику на дії, для повільних запитів. Сервера на Node.js ідеально виконують роль посередників в клієнт-серверній системі.

Наприклад відправка електронної пошти. Оскільки робота по відправці електронної пошти пов'язана з роботою через досить повільні мережі, і може зайняти досить багато часу, то доцільно відати цю роботу якомусь внутрішньому процесу, який працює незалежно від веб-сервера. В цьому випадку Node.js сервер обслуговує клієнта: приймає запити на відправку повідомлення, надсилає завдання внутрішньому сервісу, і не чекаючи відповіді. відразу може обслуговувати іншого користувача. Коли повідомлення буде відправлено, сервіс сповістить веб-сервер на Node.js про результат операції, а той вже в свою чергу сповістить користувача. Це дозволяє обслуговувати декілька клієнтів незалежно від того, наскільки тривалу операцію запросив користувач.

Сервери з асинхронною моделлю запущені в одному системному потоці, що на практиці породжує декілька проблем. Перша проблема - так звані "memory leak", коли деяка частина пам'яті не віддається операційній системі, і не використовується програмою. Наприклад, Apache Web Server створює по одному системному потоці на кожен запит, і після виконання цього запиту цей потік знищується, а займана ним пам'ять повертається ОС. У випадку Node.js розробнику слід бути уважним, не залишати слідів від попереднього запиту, оскільки вони будуть накопичуватись. Друга проблема - обробка помилок програми. Знову ж, оскільки Apache Web Server створює окремий потік для обробки вхідного запиту, то у випадку помилки в PHP коді, системний потік просто тихо помре, користувач отримає сторінку типу "500. Internal Server Error". На сам Apache це не вплине і він спокійно буде обслуговувати нові запити. У випадку Node.js, така помилка спровокує крах цілого сервера. Через це Node.js сервери необхідно постійно моніторити та перезапускати.

Ще один недолік асинхронної моделі - іноді код програми може бути дуже складним і заплутаним через переплетіння функцій зворотнього виклику. Такий код називають "спагетті-код". Саме цю проблему і намагаються вирішити, або хоча б згладити, запроваджуючи деякі типові практики, підходи до написання асинхронного коду. Розглянемо основні з них.

Основні практики

Розглянемо основні концепції управління асинхронним потоком виконання в Node.js на прикладі бібліотеки для роботи з віддаленими ресурсами. Ця бібліотека призначена для запитів до ресурсів, що знаходяться на віддалених серверах. API сервера реалізовано в REST стилі.

Для прикладу візьмемо декілька пов'язаних ресурсів: користувач, список документів користувача, останній документ. Спробуємо отримати останній документ користувач. Для цього нам спочатку необхідно отримати інформацію про користувача, далі список його документів, далі останній документ в цьому списку.

Розглянемо як би виглядав код в синхронній моделі:

```
// synchronous
var getLastDoc = function () {
        var user = hypersource('/user/123').get();
        var documents = user.follow().documents.get();
        var lastDoc = documents.follow().last.get();
        return lastDoc;
}
try {
        var lastDoc = getLastDoc();
        console.log(lastDoc);
} catch (err) {
        console.log('Error:', err);
}
```

Як бачимо, код простий, зрозумілий і не потребує детального пояснення. Недоліки такого підходу в тому, що кожен наступний запит до віддалених ресурсів блокує потік виконання, а оскільки час доступу до цих ресурсів досить великий, то блокування буде продовжуватись на досить великий термін.

Callback

Даний стиль асинхронного програмування є стандартним для Node.js, і його рекомендують при написанні власних програм. Функції, що виконуються асинхронно, приймають функцію останнім аргументом, яка буде викликана в разі завершення роботи та готовності результату, або у випадку помилка. Ця функція, яку називають функцією зворотнього виклику обов'язково приймає першим аргументом значення помилки. У разі успішного результату це значення рівне null, undefined або false.

```
// callback-style
var getLastDoc = function (callback) {
	hypersource('/user/123').get(function (err, user) {
		if (err) return callback(err);
		user.follow().documents.get(function (err, documents) {
			if (err) return callback(err);
			documents.follow().last.get(function (err, lastDoc) {
				if (!err) {
					callback(null, lastDoc);
				} else {
					callback(err)
				}
			});
		});
	});
}

getLastDoc(function (err, lastDoc) {
	if (!err) {
		console.log(lastDoc);
	} else {
		console.log('Error:', err);
	}
});
```

Promise/deferred

В цьому випадку асинхронна функція повертає спеціальний об'єкт – promise. Цей об'єкт інкапсулює в собі результат виконання функції, та повідомлення про помилки. Він зручний, коли необхідно виконувати кілька послідовних асинхронних завдань, що потребують результат попереднього завдання(як в нашому випадку).

```
// promise/deferred
var getLastDoc = function () {
	var result = hypersource('/user/123').get()
		.pipe(function (user) {
			return user.follow().documents.get();
		})
		.pipe(function (documents) {
			return documents.follow().last.get();
		});
	return result;
}

getLastDoc()
	.done(function (lastDoc) {
		console.log(lastDoc);
	})
	.fail(function (err) {
		console.log('Error:', err);
})
```

Coroutines

Співпрограми – компонент програми, що узагальнює поняття підпрограми, додатковою підтримкою безлічі точок входу (а не однієї, як підпрограма) і зупинку та продовження виконання із збереженням певного положення. Співпрограми дозволяють писати асинхронний код в синхронному стилі. Вся робота по забезпеченню асинхронності проводиться всередині віртуальної машини. Для Node.js створено бібліотеку node-fibers [3], яка реалізує співпрограми. Але оскільки по замовчуванню в Node.js, і зокрема у V8, відсутня підтримка співпрограм, і ця підтримка не планується [4], вони не будуть розглянуті.

Висновки

В даній статті було розглянуто асинхронне програмування як ще один спосіб реалізації багатозадачності. Асинхронне програмування дозволяє обходити такі проблеми багатозадачності як голодування, гонку за ресурси, інверсію пріоритету. Асинхронне програмування вимагає від програміста додаткових зусиль на організацію коду. Для невеликих проектів та для бібліотек загального використання кращим варіантом буде використання функцій зворотнього виклику. Якщо проект великий, в ньому багато модулів, багато зв'язків, багато складних процесів, тоді доцільно використовувати обіцянки (promise) як механізм для роботи з відкладеними результатами.

1. Мартин Фаулер. Шаблоны корпоративных приложений. : Пер. с англ. – М. : Издательский дом «Вильямс», 2011. – 544 с. ШЫИТ 978-5-8459-1611-2(рус).
2. http://en.wikipedia.org/wiki/Computer_multitasking.
3. https://github.com/laverdet/node-fibers.
4. https://groups.google.com/forum/?fromgroups#!topic/nodejs/BNs3OsDYsYw.
5. Что такое «асинхронная событийная модель», и почему сейчас она «в моде» http://habrahabr.ru/post/128772/
6. Асинхронность: почему это никак не сделают правильно? http://habrahabr.ru/post/99792/