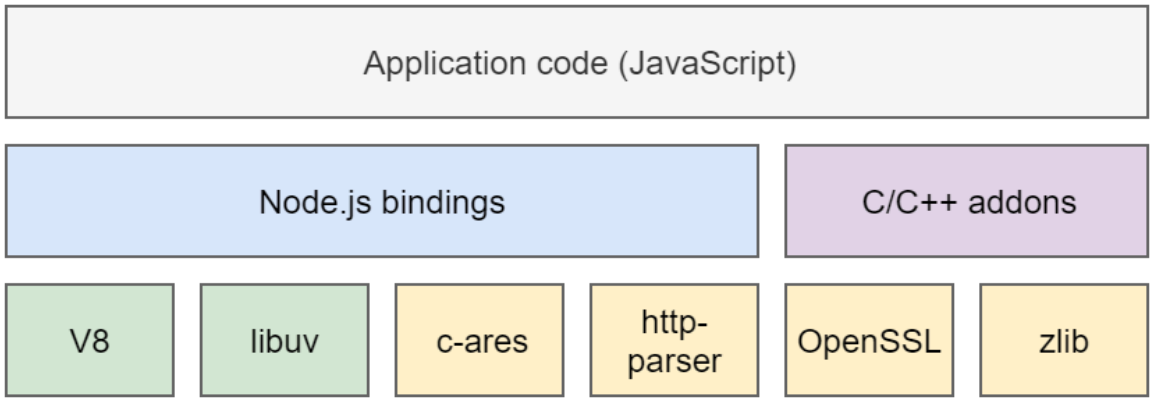
Les évènements systèmes avec LIBUV

Temps de lecture : 5 minutes

Architecture de `Node.js`

Nous allons maintenant voir globalement l'architecture de `Node.js` à un plus haut niveau :



`V8` : il s'agit du moteur `JavaScript` développé en `C++` par Google qui est ultra-performant. Il rend votre code `JavaScript` extrêmement performant. `Node.js` le contrôle en utilisant une API écrite en `C++`, il lui dit quoi exécuter et quand l'exécuter.

`libuv` : il s'agit d'une librairie écrite en `C` et qui a été développée à l'origine pour `Node.js` à partir de librairies existantes. Nous y reviendrons mais en résumé : c'est la librairie qui permet d'effectuer des opérations I/O non bloquantes quel que soit le système d'exploitation.

`c-ares` : librairie écrite en `C` pour certaines opérations de DNS.

`http-parser` : librairie écrite en `C` permettant de parser les requêtes HTTP de manière optimale avec une empreinte mémoire extrêmement faible pour chaque requête.

`openSSl` : librairie écrite en `C` et en `Perl` très connue pour toutes les opérations cryptographiques.

`zlib` : librairie écrite en `C` pour toutes les opérations de compression et de décompression.

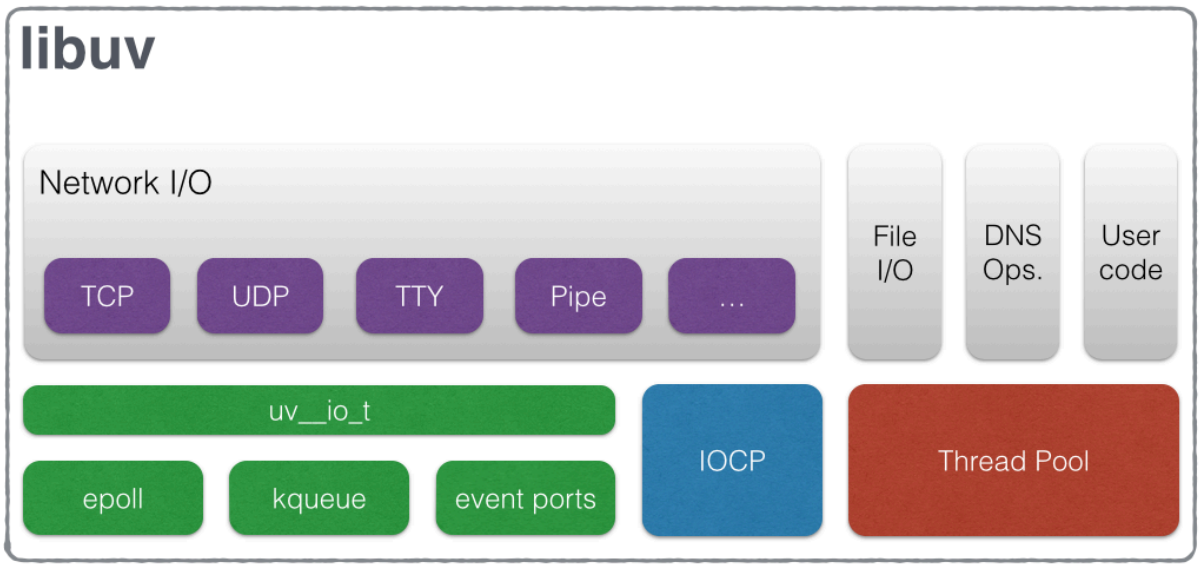
La librairie `libuv`

`libuv` est une couche permettant d'abstraire les API systèmes afin de pouvoir utiliser les opérations d'I/O de manière non bloquantes grâce à une seule API sur laquelle vient se brancher `Node.js`.

Dans le cas contraire, il faudrait utiliser l'API d'`epoll` sur Linux, l'API `kqueue` sur macOS, l'API `IOCP` sur Windows, l'API `even ports` sur les systèmes Solaris et tout recoder pour tous les systèmes d'exploitation.

`libuv` permet de gérer le système de fichiers, les opérations DNS, le réseau, les processus enfants, les pipes, le traitement du signal, le `polling`.

`libuv` permet également d'utiliser un pool de threads pour le travail pour certaines tâches qui ne peuvent pas être effectuées de manière asynchrone au niveau du système d'exploitation.



Interactions entre `Node.js`, `libuv` et le système d'exploitation

Retenez que `Node.js` veut absolument maintenir l'`event loop` sans la bloquer car elle tourne sur un unique `thread`.

Si l'`event loop` est bloquée votre application ne peut plus rien faire, elle va devoir attendre que l'opération bloquante se termine.

Comment empêcher qu'elle se bloque ?

`Node.js` va transférer toutes les tâches à `libuv` qui va transférer tout ce qu'il peut au meilleur système asynchrone disponible sur le système d'exploitation lorsque c'est possible

(pour tout ce qui est I/O réseau par exemple).

`libuv` va ensuite faire du `polling` pour savoir dès qu'un évènement système se produit et exécuter le `callback` correspondant.

Lorsqu'il n'existe pas de système asynchrone, il va prendre un `worker` dans la `thread pool` qui est de 4 par défaut et va lui faire exécuter du code bloquant (système de fichiers, certaines opérations DNS, opérations cryptographiques, calculs complexes).

Les `threads` du `pool` effectuent les tâches bloquantes et `libuv` sait quand les `threads` ont terminé.

Ensuite il va exécuter le `callback` correspondant.

Donc vous voyez pourquoi les évènements systèmes sont très importants : ils permettent à `Node.js` de savoir quand des tâches de fond effectuées par le système d'exploitation sont terminées et d'exécuter le code correspondant en exécutant les `callback` liées.

Rappels sur le point faible de `Node.js`

`Node.js` est une architecture prévue pour **le plus de tâches concurrentes possibles**.

Du moment que les tâches ne sont pas gourmandes en CPU, vous pourrez gérer un nombre extrêmement importants de connexions car l'empreinte mémoire des connexions est très faible.

Lorsque vous faites des opérations nécessitant du calcul constant, comme une encryption synchrone, `Node.js` va prendre un `thread` dans les 4 disponibles par défaut et exécuter les opérations dessus pour ne pas bloquer le `thread` principal sur lequel tourne l'`event loop`.

Retenez que si vous devez faire des tâches comportant beaucoup de calculs, il faut essayer de les déléguer à autre chose que `Node.js`, car son architecture n'est pas optimisée pour.