Broker - Les logs communs des modules

Sommaire

Problème à la libération de ressources lors de la création d'un processus

Pour exécuter certaines tâches (*Modules*, *Workers*, ...), le Broker doit créer de nouveaux processus sur le système.

Sur les environnements UNIX et notamment Linux, la procédure pour créer un nouveau processus consiste à dupliquer le processus courant. L'appel système associé se nomme **fork**, ainsi, cette action est souvent appelé *un fork*.

Sous Linux, l'opération de **fork** est rapide, l'allocation effective de la mémoire du nouveau processus se faisant lorsque le nouveau processus veut y accéder.

Python disposant d'un *garbage collector*, un outil parcourant la mémoire pour identifier les zones inutilisés et la restituer au système, la mémoire héritée du parent finit par être intégralement copiée dans le nouveau processus.

Pour éviter une consommation inutile de ressources, après le fork, on nettoie les données inutiles issues du processus père (mémoire principalement)

Plus concrètement, le Broker va supprimer les données produites par les modules qu'il a instancié.

Les modules peuvent fournir une méthode pour gérer la libération de leur ressources.

Si cette méthode provoque une erreur, on retrouvera un log de ce style :

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] On linux system, the forking mechanism (process creation) is fast but create a copy of the father process. So we have to release unnecessary resources inherited from father. The cleaning has been performed but we encountered an error:

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] Cleanup of data from module **MODULE_NAME2** raised error IOError occurred

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] Some memory may have not been freed. Shinken will still run if enough memory remains available.

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] You can report this message to support in order to optimize Shinken memory consumption

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] ERROR stack : Traceback (most recent call last):

[YYYY-MM-DD HH:MM:SS] ERROR : [**DAEMON_NAME** (- Module: **MODULE_NAME**)] [**MODULE_NAME**] [CLEAN AFTER FORK] [pid:**PID**] [**MODULE_NAME2**] ERREUR PYTHON

```
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] On linux system, the forking mechanism (process creation) is fast
but create a copy of the father process. So we have to release unnecessary resources inherited from father.
The cleaning has been performed but we encountered an error:
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] Cleanup of data from module  sla  raised error  IOError occurred
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] Some memory may have not been freed. Shinken will still run if
enough memory remains available.
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] You can report this message to support in order to optimize Shinken
memory consumption
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] ERROR stack : Traceback (most recent call last):
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ]   File "/usr/lib/python2.7/site-packages/shinken/modulesmanager.
py", line 877, in do_after_fork_cleanup
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ]     inst_cleanup()
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ]   File "/var/lib/shinken/modules/sla/sla_module_broker.py", line
93, in after_fork_cleanup
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ]     raise IOError(u'IOError occurred')
[2021-09-29 15:05:58] ERROR  : [ shinken-broker-master ( - Module: Livestatus ) ] [ Livestatus       ] [
CLEAN AFTER FORK ] [ pid:20404 ] [ sla ] IOError: IOError occurred
```

Envoi et réception de commande entre plusieurs processus

Les logs suivants permettent de suivre l'envoi et la réception de commande se faisant entre plusieurs processus.

Il existe deux types de communications par commande :

- Communication entre le Broker et un de ses modules
- Communication entre un module et un de ses workers

Envoi d'une commande

Debug

Dans le cas d'une réussite de l'envoi de la commande, jusqu'à sa réception, ce log s'affichera :

[YYYY-MM-DD HH:MM:SS] DEBUG : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] The command call **[NOM_DE_LA_COMMANDE]** was executed by the module **NOM_DU_MODULE** in **TEMPS_D'EXECUTIONS**

```
[2020-11-17 09:12:11] DEBUG  : [ broker-master   ] [ Livestatus       ] The command call [get_module_info]
was executed by the module Livestatus in 0.143s
```

Warning

Dans le cas d'un premier timeout d'une commande, ce log s'affichera. La commande sera alors renvoyée une deuxième fois.

[YYYY-MM-DD HH:MM:SS] WARNING : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] The command call **[NOM_DE_LA_COMMANDE]** for module **NOM_DU_MODULE** did timeout (**TEMPS_TIMEOUT**s). We will retry one time.

```
[2020-11-17 09:12:11] WARNING  : [ broker-master   ] [ Livestatus       ] The command call [get_module_info]
for module Livestatus was sent but the call did timeout (1s). We will retry one time.
```

Il peut arriver qu'à l'envoi d'une commande une autre réponse soit reçue, si la précédente commande n'a pas fonctionné par exemple. Dans ce cas, ce log s'affichera. La commande sera alors renvoyée afin de récupérer la bonne réponse.

[YYYY-MM-DD HH:MM:SS] WARNING : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] The command call **[NOM_DE_LA_COMMANDE]** was called but another respond was present. Retrying.

```
[2020-11-17 09:12:11] WARNING  : [ broker-master   ] [ Livestatus       ] The command call [get_module_info]
was called but another respond was present. Retrying.
```

Error

Si l'envoi de la commande a connu un premier timeout, à son deuxième elle passera en erreur et ne sera pas renvoyée. Ce log sera affiché :

[YYYY-MM-DD HH:MM:SS] ERROR : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] Fail to send command call **[NOM_DE_LA_COMMANDE]** for module **NOM_DU_MODULE** because the module did timeout (**TEMPS_TIMEOUT**s).

```
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ] Fail to send command call
[get_module_info] for module Livestatus because the module did timeout (1s).
```

Il peut arriver que la commande échoue à cause d'un problème du côté du module/worker. Dans ce cas ce log sera affiché après l'affichage de la stack.

[YYYY-MM-DD HH:MM:SS] ERROR : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] Fail to send command call **[NOM_DE_LA_COMMANDE]** for module **NOM_DU_MODULE** because of an unknown error **MESSAGE_D'ERREUR**

```
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ] Fail to send command call
[get_module_info] for module Livestatus because of an unknown error 'int' object is not iterable
```

Réception d'une commande

Debug

Au moment de la réception de la commande, si cette dernière est exécutable, ce log sera affiché :

[YYYY-MM-DD HH:MM:SS] DEBUG : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] [PID:**PID_DU_PROCESSUS**] Executing command **[NOM_DE_LA_COMMANDE]** with param **LISTE_DES_PARAMETRES**

```
[2020-11-17 09:12:11] DEBUG  : [ broker-master   ] [ Livestatus       ] [PID:2564] Executing command
[get_module_info] with param []
```

Warning

Si une commande inconnue est reçue, ce log sera affiché :

[YYYY-MM-DD HH:MM:SS] WARNING : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] [PID:**PID_DU_PROCESSUS**] Received unknown command **[NOM_DE_LA_COMMANDE]** from father process !

```
[2020-11-17 09:12:11] WARNING  : [ broker-master   ] [ Livestatus       ] [PID:2564] Received unknown
command [get_module_info] from father process !
```

Error

Si la commande crash, ce log sera affiché :

[YYYY-MM-DD HH:MM:SS] ERROR : [**NOM_DU_BROKER**] [**NOM_DU_MODULE**] Our father process did send us the command **[NOM_DE_LA_COMMANDE]** that did fail: **TRACEBACK**

```
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ] Our father process did send us the
command [get_module_info] that did fail:
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ] Traceback (most recent call last):
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ]   File "C:\dev\workspace\shinken-
enterprise\sources\framework\shinken\shinken\basesubprocess.py", line 117, in
get_and_execute_command_from_master
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ]     result = f()
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ]   File "C:\dev\workspace\shinken-
enterprise\testing\test_command_queue_handler.py", line 43, in fail_command
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ]     raise Exception
[2020-11-17 09:12:11] ERROR  : [ broker-master   ] [ Livestatus       ] Exception
```