Université de Paris 2021/2022

« Saturnd / Cassini »

PROJET DE
BRAGINA Natalia, SAAD David et TEIXEIRA Gabriel

Architecture du projet

Les fonctions main se trouvent dans les fichiers **saturnd.c** et **cassini.c**.

Les fonctions desservant l'exécution de tasks pour Cassini sont fournies par le ficher **client-request.c**

Les fonctions desservant l'exécution de tasks pour Saturnd sont fournies par **daemon-proc-tasks.c**, lancement du mode de démon - **run-daemon.c**, les fonctions auxiliaires pour effectuer l'exécution de tasks coté saturnd **daemon-aux-functions.c** et **listd.c** (bibliothèque générique)

Cassini

La partie client cassini est composée de plusieurs fonctions permettant à l'utilisateur d'interagir avec l'ensemble des tâches qu'il aura défini. L'utilisateur dispose notamment de fonction pour la création, la suppression, l'affichage d'un historique et le listage de tâches. Il pourra également récupérer les sorties standards ainsi que terminer le programme.

Saturnd

Structures créées

Afin de transférer et restaurer le contexte, nous avons créé une structure **stContext**, qui a les champs suivants :

struct stContext

struct listElements_t	*tasks	La liste des tâches
int	pipeRequest	Les descripteurs de tubes
int	pipeReply	
uint64_t	lastTaskId	Le dernier ID attribué
int	exit	Le drapeau de sortie utilisé pour savoir s'il faut de terminer le processus
struct stString	*pipeReqName	Les deux pointeurs vers les chemins d'accès aux tubes
struct stString	*pipeRepName	

Nous transmettons cette structure à chaque fonction qui fait le travail.

Pour stocker les informations sur les tâches, nous avons créé une structure **stTask** avec les champs suivants :

struct stTask

uint64_t	taskId	Le numéro de la tâche
unsigned char	minutes[60]	Le timing sous forme d'une chaine de caractère composé de 0 et de 1
unsigned char	hours[24]	
unsigned char	daysOfWeek[7]	
uint64_t	min	Le timing de la tâche
uint32_t	heu	
uint8_t	day	
size_t	argC	Le nombre d'arguments de la tâche
struct stString	**argV	Les Arguments de la tâches
int	stdOut	Les descripteurs de la sortie standard et sortie d'erreur standard
int	stdErr	
struct tm	stCreated	Date de création de la tâche
struct tm	stExecuted	Dernière date d'exécution de la tâche
pid_t	lastPid	Le PID du dernier processus ayant exécuté la tâche
struct listElements_t	*runs	L'ensemble des dates d'exécution et leurs valeurs de retour

Quelques précisions :

- Le timing composé de 0 et de 1 permet de faciliter la surveillance pour le lancement des tâches. En effet, dans les tableaux ci-dessus, les cases marquées par le caractère 1 signifient : les heures, minutes et les jours auxquels la tâche doit être exécutée.
- Dans l'idéal il aurait fallu pouvoir éviter la redondance d'informations mais par manque de temps ; deux champs pour les heures, les minutes et les jours sont présents pour chaque tâche.

Schéma de fonctionnement du démon

Dans un premier temps, nous ouvrons le pipeRequest en lecture/écriture avec une option non-bloquante (afin de ne pas recevoir POLLHUP lorsque Cassini ferme le tube après écriture). Ensuite, nous chargeons des données de dernière session à partir du disque.

Et enfin nous exécutons un cycle composé de deux étapes :

- Sondage sur pipeRequest en attendant l'écriture dessus (fréquence de vérification 2 fois/sec). Si une requête arrive, il lit son OPCODE et délègue aux fonctions auxiliaires la lecture de la partie restante et l'exécution ultérieure.
- Vérification des horaires d'exécution des tâches sur la liste des tâches et relancement si nécessaire.

Pour sortir de ce cycle, il suffit que soit une des fonctions termine avec une erreur, soit que la commande « terminate » ai été reçu. Dans ce cas, nous réalisons une sauvegarde de la session courante, fermons des descripteurs et libérons la mémoire des structures.

Concernant la maintenance, nous vérifions si une tâche est terminée ou bien encore si elle doit être lancée. Dans le cas où une tâche doit s'exécuter nous vérifions bien que sa dernière exécution n'est pas récente. Une fois que la tâche est terminée, nous récupérons les informations sur l'achèvement du travail par le biais de la fonction waitpid puis, nous enregistrons les informations dans la structure stRunStat de ce processus.

Lors de la création d'une tâche, nous lisons les informations envoyées par cassini puis remplissons la structure de la tâche. Après avoir rempli le champ concernant l'heure de création de la tâche, il est important de noter que nous modifions le champ concernant l'heure d'exécution de la tâche en lui retranchant 59 secondes afin que la fonction de maintenance qui surveille l'exécution des tâches puisse la prendre et l'exécuter au plus vite possible. Une fois cela fait, nous écrivons les données nécessaires sur le tube de réponse.

- Ouverture de pipe en lecture et écriture
- Chargement des données de la dernière session depuis le disque
- Vérifier si une tâche doit être exécutée avec le timing
  Attente écriture du message sur le pipe de requette
  Sortie : . Une erreur ; . Signal terminate → Sauvegarde de la session courante, Fermeture des descripteurs, Libération de la mémoire
- Maintenance - Exécution
  Vérifier si la tâche est terminée
  → Si c'est le cas grâce au waitPid nous mettons à jour la structure stRunStat
  Si sa dernière exécution n'est pas récente (inférieure à 5 secondes) la tâche doit être exécutée
- tube de requête → Cassini → tube de réponse
  Remplissage de la structure de tâche
  Petite précision: le champ concernant la date de création est rempli avec le temps actuel, de même avec le champ date concernant la d'exécution mais en cette fois-ci en retranchant 59s pour que le Mainteneur puisse exécuter la tâche au plus vite

Les tâches qu'il faut exécuter possèdent leur champ tasl.lastPid à 0. Si elles sont en cours d'exécution, le champ lastPid est supérieur à 0 car il possède le PID du processus qui l'exécute.

On utilise Waitpid avec l'option WNOHANG qui permet au père de continuer ce qu'il a à faire lorsque le fils n'a pas terminé. L'option permet que dès lors que le fils se termine d'envoyer un signal au père qui va mettre à jour le statut de la tâche lastPid à 0. Ce qui permettra de réaliser à nouveau la tâche le moment venu.

Notre démon crée des enfants ; pour ne pas s'y perdre, nous avons mis en place plusieurs marqueurs pour mettre en évidence la présence des processus et leurs exécutions. Par exemple, lastPid permet de garder une trace du processus qui exécute la tâche courante, une fois que la tâche est terminée, lastPid passe à 0. Nous gardons également l'historique de chaque exécution des tâches dans la struct listElement_t run ce qui nous permet de garder une trace du nombre de fois qu'elle a été exécutée et du bon déroulement de son travail.

Autre :

Nous utilisons une petite bibliothèque de liste (générique) pour travailler avec la liste des tâches et l'historique des exécutions. De plus, de nombreuses petites fonctions ont été créées pour permettre de clarifier le code (construire des chemins, libérer des ressources et fermer des fichiers). Par ailleurs, nous avons également décidé d'utiliser des fonctions écrites sous forme de macros.