# 

LAB4 - La gestion des processus

**Programmation système**

**Thierry Khamphousone – TC-44**

Table des matières

[Objectifs 2](#__RefHeading__317_62609081)

[Rappels et mise en place de l’environnement 2](#__RefHeading__742_1331557722)

[Gestion des processus 2](#__RefHeading__9_62609081)

[1Shell 2](#__RefHeading__35634_1331557722)

[2Création de processus en C 3](#__RefHeading__35636_1331557722)

[3Schéma « gestion des processus » 4](#__RefHeading__1262_1596899261)

By Rexy



# Objectifs

Ce « LAB » présente les notions de base de gestions de processus. Il est basé sur la mise en parallèle de notions relatives aux commandes Shell (et Scripts Shell) avec leur équivalent en langage C.

* Documents : «Linux : la gestion des processus ».
* Notions abordées : le « scheduler », les signaux, la gestion synchrone et asynchrone, /proc
* Commandes et fichiers exploités : ps, pstree, top, trap, kill, fork(), exec().
* Travail à rendre : Vous devrez répondre directement à plusieurs questions au sein de ce document. Vous le copierez sur Moodle sous le nom : « LAB3\_noms.pdf ».

# Gestion des processus

## Shell

Lisez le document « Linux – gestion des processus »

Répondez aux questions suivantes en utilisant **une couleur distincte**.

* Dans un terminal : lancez un éditeur de texte simple (« gedit », « Vim », « Emacs », etc.). En se lançant, votre éditeur affiche (ou pas...) des informations sur le canal des erreurs. Comment feriez-vous pour qu’il n’affiche plus ces erreurs ? **=> Utilisation de la redirection de la sortie d’erreur vers un fichier erreur.log** (commande : ﻿**vim GigaPanda 2>> erreur.log).** Testez. Info : il est toujours intéressant de lancer les applications graphiques à partir d’un terminal, car cela permet de voir des dysfonctionnements souvent cachés par les interfaces graphiques (idem sous Windows).
* Pourquoi n’avez-vous plus la main sur votre terminal ? (**Je n’ai plus la main sur mon terminal car le processus de l’éditeur de texte est en cours sur le terminal courant**). Lancez en aveugle la commande « id ». Fermez votre éditeur via ses menus graphiques. Cela est-il normal ? (**La commande « id » a bien été entrée dans l’entrée standard et attend que le processus en cours concernant l’éditeur de texte se termine avant de pouvoir se lancer**). Relancez-le de manière asynchrone et sans affichage des messages d’erreur (commande : **﻿gedit GigaPanda 2>/dev/null &).**
* Définissez les 13 champs affichés par la commande « ps -faxl » (« F » : **indicateur sur l’état du processus [1] : mémoire vive et [2] : terminé** « UID » : **Identifiant unique de l’user**, « PID » : **numéro identifiant du processus**, etc.). Quels sont le PID et le PPID de votre éditeur ? (**Mon éditeur de texte gedit possède le numéro de processus PID 2061 et son PPID est 1783)**. Quel est le processus PPID ? (**Le PPID 1783 de mon éditeur correspond au PID du processus bash, PPID correspond enfaite au numéro de processus parent, le processus qui l’a créé.**)
* Affichez la liste des signaux disponible sur votre système (**kill -l**). En tant que père de votre éditeur, votre shell peut lui envoyer des signaux. Fermer votre éditeur en lui envoyant le signal d’interruption « SIGINT » (commande : **kill -2 2061**).
* Le PPID d'un shell correspond au processus qui gère le terminal et les entrées clavier. Quel est-il sur votre système ? (**PPID bash : 1778**). Quelle commande permet de connaître les caractéristiques du terminal : (**stty -a**). En analysant cette commande, vous pourrez connaître les séquences de touches associées à certains envois de signaux. Quelle séquence pour le signal SIGINT ? (**CTRL + C**). Testez.
* Lancez votre éditeur de manière synchrone (vous n’avez donc plus la main). Lancez-lui le signal de suspension (SIGSTOP) de deux manières différentes (1 : **CTRL + Z**) (2 : **kill -19 2061**). Vous reprenez ainsi la main. Votre éditeur est bien toujours présent, mais comme le processus ne passe plus dans la boucle du microprocesseur, il est devenu inerte (sans activité). Vérifiez. Vous pouvez le réactiver en lui envoyant le signal (SIGCONT) de deux manières différentes (1 : **kill -18 3806**) (2 : **kill -SIGCONT 3806**) (**bg et fg fonctionnent aussi**). Testez. Vous pouvez réactiver tous les processus stoppés via la commande « bg » (background). INFO : vous connaissez maintenant un bon moyen de reprendre la main sur votre terminal quand vous lancez un processus graphique.
* Qu’arrivera-t-il à votre éditeur si son père meurt ?(**en synchrone, gedit se ferme au moment où le terminal se ferme car le terminal est le PID père de gedit**). Testez en fermant le terminal. Rouvrez un terminal et lancez-le de manière asynchrone afin qu’il ne dépende plus de son père en cas de mort subite de ce dernier (commande : **nohup gedit &**). Vérifiez en fermant le terminal. Qui est devenu PPID de votre éditeur ? (**Le PPID de mon éditeur a été récupéré par le processus 997 qui correspond au processus de systemd (OS). Le processus parent PPID de 997 est le processus 1 qui correspond au processus init**).
* Les signaux sont traités par les processus via les standards de programmation. Le « shell » est un processus comme un autre. Ses programmeurs ont prévu que l’utilisateur puisse intercepter les signaux afin de modifier l’action prévue normalement. La commande « trap » qui est interne au « shell » permet ainsi d’intercepter et de reprogrammer l’action des signaux. Ainsi, dans un « shell », la commande « trap ‘ls’ nro\_signal » permet d’intercepter le signal numéro ‘nro\_signal’ quand il est reçu afin de lancer la commande « ls » au lieu du traitement normalement prévu. À quelle séquence de touche correspond le signal « SIGINT » ? (**CTRL+C**). Reprogrammer l’action liée à ce signal afin de lancer la commande « ps ». (**trap ”ps” 2**)Testez. Comment pouvez-vous réinitialiser cette programmation à sa valeur par défaut ? (**trap 2**).
* Créez un « script shell » qui affiche la liste des usagers de votre système. Cette liste est dans le fichier « /etc/passwd » (vous pouvez judicieusement exploiter la commande « cut »). Dans un terminal (et donc un shell), modifiez le comportement du signal numéro 3 afin de lancer votre script. Ouvrez un deuxième terminal. Déterminez le PID du premier terminal et envoyez-lui le signal numéro 3. Faites une capture d’écran des 2 terminaux. Supprimez l’interception du signal numéro 3.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

* Quand un processus reçoit le signal Nro 1 (SIGHUP), il se ferme normalement (HUP = Hang UP = raccrocher). À partir du deuxième terminal, lancez le signal Nro 1 sur le premier terminal. Rouvrez un nouveau terminal et interceptez le signal 1 afin qu’il ne se ferme plus lors de la réception de ce signal. Testez. Quel signal faudrait-il envoyer pour fermer un processus qui ne veut plus rien savoir ? (**SIGINT** **ou kill -2 [process\_id]**). Qui traite en fait ce signal ? (**Arrêt par CTRL+C, traite l’interruption du processus)**. INFO : C’est parfois le seul moyen de supprimer un processus planté (boucle infinie par exemple).
* Gestion interactive : Sous KDE, GNOME : <CTRL> + <ECHAP>. Sous Windows : <CTRL> + <ALT> + <Suppr>. Sous l’environnement minimaliste LXDE : commande « lxtask ». Dans un terminal texte : commande « top » (puis « h » pour connaître les interactions possibles).

## Création de processus en C

Dans les LAB précédents, vous avez exploité la fonction « system (commande) » de la « librairie C standard (stdlib) » qui lance un « shell » à l’intérieur duquel votre commande est exécutée. Votre programme appelant attend la fin de cet appel pour poursuivre son exécution. L’enchaînement est donc synchrone.

Vous allez maintenant exploiter les possibilités du système multiprocessus qui permet de créer un nouveau processus géré de manière asynchrone et donc traité en parallèle (fonction « fork » et « exec »).

Appuyez-vous sur le guide suivant :

cf. <http://mtodorovic.developpez.com/linux/programmation-avancee/?page=page_3#L3>

Réalisez un programme en C qui crée un processus fils. Le père affiche son PID et le PID de son fils et attend que vous tapiez la touche « p ». Le processus fils affiche son PID et son PPID. Il ne fait qu’attendre la réception d’un signal de son père (que vous définirez) pour mourir. Le père envoie ce signal quand vous tapez la touche « p ». Quand il s’est assuré que son fils est bien mort, il quitte aussi cette dure vie.

Réalisez une capture d'écran de son exécution.

A screenshot of a cell phone

Description automatically generated

Une fois le fils créé, le père et le fils veulent écrire « je suis le (père/fils) et mon PID est ... » dans un même fichier. Ils ne peuvent cependant pas le faire en même temps... Cherchez et développez une méthode de résolution de ce problème d’accès concurrent à une ressource (sémaphore, mutex, etc.).

A screenshot of a cell phone

Description automatically generated

**Rendez ce code C sur Moodle dans un fichier part-2.c.**

**Une solution serait d’utiliser les threads et d’effectuer les ecritures dans le fichier en asynchrone.**

**Le code C a bien été déposé sur le Moodle au nom de part-2.c**

## Schéma « gestion des processus »

Réalisez ci-dessous (ou sur un document annexe) le schéma de gestion des processus sous Linux. Vous tenterez de faire apparaître les termes suivants : « microprocesseur », « mémoire vive », « zone de swap », « scheduler » « inerte » « en attente », « en sommeil », « actif », « mort », « zombie », « priorité ». Dans un court texte, vous expliquerez la vie trépidante d'un processus.

**Le schéma et sa description ont été réalisé sur un document annexe déposé sur le Moodle au nom de : « vie\_trepidante\_processus.pdf »**