SysMonitor – Documentation Utilisateur

Introduction

SysMonitor.sh est un script shell permettant de gérer et surveiller les procesus système sur un ordinateur Linux via le répertoire ***/proc***. Il offre des informations détaillées sur le système, la gestion des processus (suspendre, reprendre, tuer) ainsi que des options pour sauvegarder ces informations dans un fichier *.log*.

Le script se divise en plusieurs fonctions pour différentes options :

1) ***cpu_info*** **(option « par défaut ») :** Afficher un résumé des infos sytèmes (CPU, RAM, Uptime, etc.)
2) ***list_processes*** **(option -lp) :** Lister tous les processus en cours via ***/proc***.
3) ***process_detail*** **(option -s pid) :** Afficher les détails (Nom, Etat, Mémoire, etc.) du processus pid
4) ***info_cpu (option -lcpu)*** : Afficher les informations CPU extraites de ***/proc/cpuinfo***.
5) ***save_system_info*** **(option -s ou --save) :** Sauvegarder les information systèmes pertinentes dans le fichier ***sysinf.log***.
6) ***launch_bg*** **(option -bg cmd) :** Lancer le programme « cmd » en arrière-plan ; cela un simple « ls -l »
7) ***stop_process*** **(option -stop pid): Suspendre** le processus pid
8) ***continue_process*** **(option -cont pid) : Relancer** le processus pid suspendu en arrière-plan
9) ***kill_process*** **(option -kill pid) : Terminer** le processus pid
10) ***print_help*** **(option -h ou --help) :** Afficher l'aide du script avec des exemples
11) ***invalid_option*** : Gérer une option invalide et redemander une entrée (une option) à l'utilisateur.
12) ***MAIN*** : La Fonction Principale pour lancer le programme du script.

Fonctions

Fonction *cpu_info()* : Résumé des Informations Système (par défaut)

Description

La fonction « ***cpu_info()***» permet des informations basiques de la machine (système d'exploitation, nom de la machine, uptime, ram, nom du CPU)

Code

```
cpu_info() {
    echo "./SysMonitor.sh" >> "$logfile"
    {
        echo "======= INFORMATIONS SYSTEME ======="
        echo ""
        echo "Système d'exploitation : $(uname -o -p)"
        echo "Nom de la machine      : $(hostname)"
        echo "Uptime                 : $(uptime)"
        echo "RAM utilisée / totale  : $(free -h | grep 'Mem' | awk '{print $3 "/" $2}')"
        echo "Nom du CPU             : $(grep 'model name' /proc/cpuinfo | head -n1 | awk '{for(i=4;i<=NF;i++) printf $i " "; print ""}')"
    } | tee -a "$logfile"  # Affiche à l'écran ET enregistre dans le fichier log (sysLog.txt)
}
```

uname -o -p : permet d'afficher le système d'exploitation (-o) et le type de processeur (-p)free -h : affiche la quantité de mémoire RAM libre et utilisée par le système en octets

grep "Mem" : cherche et sélectionne la ligne Mem

awk '{print $3 "/" $2}' : permet d'afficher au même endroit la mémoire utilisée sur la mémoire totale mais sans faire une division (les guillemets l'empêchent)

grep "model name" | head -n1 | awk '{for(i=4;i<NF;i++) printf $i " " ; print "" }' :

1) On affiche le contenu du fichier cpuinfo contenu dans /proc
2) On cherche la ligne qui précise le nom du CPU (comporte la chaine « model name »), on prend que la première ligne (car répétif)
3) On affiche la ligne à partir de la 4e colonne jusqu'à la fin de la ligne

Fonction *list_processes()* : Liste des Processus (-lp)

Description

La fonction « ***list_process()*** » permet de lister tous les processus actifs sur le système. Les informations affichées incluent le PID (l'identifiant du processus), le Nom du processus et son État actuel (Inactif « I », En Cours « R », Suspendu « S », Zombie « Z », etc.).

Code

```
# Fonction pour Lister tous les processus en cours via /proc (option -lp)
list_processes(){
    echo "./SysMonitor.sh -lp" >> "$logfile"
    {
        # Titre de section
        echo "=== Liste des processus en cours ==="

        # En-têtes de colonnes : PID, nom du processus, état
        printf "%-10s %-25s %-10s\n" "PID" "Nom" "État"
        echo "-----------------------------------------------"

        # Boucle sur tous les dossiers numériques de /proc (correspondant aux PIDs)
        for pid in /proc/[0-9]*; do
            # Vérifie que le fichier status existe (donc qu'il s'agit bien d'un processus)
            if [ -f "$pid/status" ]; then
                # Récupère le PID depuis le nom du dossier
                PID=$(basename "$pid")

                # Extrait le nom du processus depuis le fichier status
                NOM=$(grep -s "^Name:" "$pid/status" | awk '{print $2}')

                # Extrait l'état du processus (R = en cours, S = en veille, Z = zombie, etc.)
                ETAT=$(grep -s "^State:" "$pid/status" | awk '{print $2}')

                # Affiche les infos du processus formatées proprement
                printf "%-10s %-25s %-10s\n" "$PID" "$NOM" "$ETAT"
            fi
        done

        echo "" # Saut de ligne
    } | tee -a "$logfile"  # Affiche à l'écran ET enregistre dans le fichier log (sysLog.txt)
}
```

Exemple d'Utilisation

```
osboxes@osboxes:~/programmation_shell/projet$ ./SysMonitor.sh -lp
=== Liste des processus en cours ===
PID        Nom                         État
-----------------------------------------------
1          systemd                     S
10         kworker/0:0H-events_highpri I
1049       ModemManager                S
1060       VBoxService                 S
11         kworker/u16:0-ipv6_addrconf I
115        kworker/R-charger_manager   I
12         kworker/R-mm_percpu_wq      I
1217       unattended-upgr             S
1231       gdm3                        S
1247       kerneloops                  S
1250       kerneloops                  S
1296       rtkit-daemon                S
13         rcu_tasks_kthread           I
1394       colord                      S
```

➔ Dans cet exemple, nous avons plusieurs processus Suspendus (le processus de PID 1, par exemple) et quelques-uns Inactifs (le processus de PID 10, par exemple).

Fonction ***process_details()*** : Afficher les détails du pid (-s pid)

Description

La fonction « ***process_details()*** » permet d'afficher sur le terminal les informations du processus spécifié par le biais de son PID (nom, état et mémoire utilisée).

La fonction commence par vérifier si un argument a été saisi. Si ce n'est pas le cas, il l'indique à l'utilisateur et qui la fonction avec un code d'erreur. Une fois, le pid saisi en argument, il regarde si le PID du processus est présent dans la liste des processus actifs.

Code

```
process_details(){
    pid_saisi=$1
    if [[ -z $pid_saisi ]] #si pas d'argument mis dans la fonction (pid non saisi)
    then    echo "Vous n'avez pas saisi de pid." | tee -a "$logfile"
    return 1
    fi

    #Verification de l'existance du PID
    echo -n "PID saisi: " | tee -a "$logfile"
    ps aux | awk '{print $2}' | grep ^${pid_saisi}$ | tee -a "$logfile"

    echo -e "\n======= Détails sur le PID $pid_saisi =======\n" | tee -a "$logfile"
    retour=$?

    if [[ $retour -eq 0 ]]
    then    ps -q "$pid_saisi" -o comm=NOM -o %mem="%MEM" -o stat="Statut" 2>&1 | tee -a "$logfile"

    else    echo "Le PID n'existe pas" | tee -a "$logfile"
    fi
}
```

ps aux : Affiche la liste de tous les processus actifs avec des informations systèmes tels que la mémoire, le PID etc..

awk '{print $2}' : Permet de ne sélectionner que la 2e colonne de ps aux (correspond à la colonne PID)

grep ^${pid_saisi}$: Permet de chercher dans la liste des processus actifs une occurrence du pid saisi. Il est important de mettre les symboles ^ (commence par) et $ (finit par), pour indiquer qu'on cherche exactement le pid saisi et non pas un pid qui a une occurrence de cette chaine de caractère. On aurait pu également utilisé grep -w ${pid_saisi}

ps -q $pid_saisi -o comm=NOM -o %mem,stat : Affiche les processus en cours par PID (-q) avec les catégories choisies par l'utilisateur (-o). Dans le man ps, il va des exemples d'utilisations de ps par rapport à ce qu'on voudrait afficher, ce qui a été grandement utile pour afficher les informations qui nous intéressaient. -o est un format défini par l'utilisateur, il permet d'afficher les catégories (nom, état etc..) qu'on veut et on peut même personnaliser les noms de ces colonnes.

if [[-z $pid_saisi]] : -z est un operateur du if qui permet de vérifier si une variable est vide ou non (NULL)

Fonction *infos_cpu()* : Afficher les Informations CPU (-lcpu)

Description

La fonction « ***info_cpu()*** » fournit des informations détaillées sur le processus du système, telles que le Modèle, la Fréquence, le Nombre de Cœurs Logiques et la Taille du Cache.

Code

```
# Fonction pour afficher les infos CPU extraites de /proc/cpuinfo (option -lcpu)
infos_cpu() {
    # Trace l'utilisation de l'option -lcpu dans le fichier de log
    echo "./SysMonitor.sh -lcpu" >> "$logfile"

    {
        # Titre avec emoji pour le fun 😄
        echo "🧠 === Informations CPU ==="

        # Récupère et affiche le modèle de processeur (ex: Intel(R) Core(TM) i7...)
        modele=$(grep -m1 "model name" /proc/cpuinfo | cut -d: -f2 | sed 's/^ //')
        echo "📌 Modèle        : $modele"

        # Récupère et affiche la fréquence actuelle du CPU en MHz
        frequence=$(grep -m1 "cpu MHz" /proc/cpuinfo | cut -d: -f2 | sed 's/^ //')
        echo "🎚️ Fréquence     : $frequence MHz"

        # Compte le nombre total de cœurs logiques présents (threads)
        coeurs_logiques=$(grep -c ^processor /proc/cpuinfo)
        echo "🧩 Cœurs Logiques: $coeurs_logiques"

        # Récupère la taille du cache L2 du CPU
        cache=$(grep -m1 "cache size" /proc/cpuinfo | cut -d: -f2 | sed 's/^ //')
        echo "🗃️ Cache         : $cache"

        # Récupère l'architecture système (x86_64, i686, etc.)
        arch=$(uname -m)
        echo "🏗️ Architecture  : $arch"

        echo ""  # Saut de ligne
    } | tee -a "$logfile"  # Affiche à l'écran et sauvegarde dans le fichier log
}
```

Exemple d'Utilisation

```
osboxes@osboxes:~/programmation_shell/projet$ ./SysMonitor.sh -lcpu
🧠 Affichage des informations CPU...
🧠 === Informations CPU ===
📌 Modèle        : Intel(R) Xeon(R) CPU E31270 @ 3.40GHz
🎚️ Fréquence     : 3400.030 MHz
🧩 Cœurs logiques: 4
🗃️ Cache         : 8192 KB
🏗️ Architecture  : x86_64
```

Dans cet exemple, nous avons les informations suivantes :

- Le **Modèle** du CPU est un Intel (R) Xeon (R) CPU E31270 @ 3.40GHz
- La **Fréquence** (exacte) du CPU est de 3400.030 MHz
- Le CPU dispose de 4 **Cœurs logiques**
- L'**Architecture** du CPU est un 64 bits

Fonction ***save_system_info()*** : Sauvegarder les Informations Système (--save)

Description

La fonction « ***save_system_info()*** » permet la récupération des informations du système (Nom d'utilisateur, temps de connexion, etc) dans un fichier « ***sysinfo.log*** ».

Code

```
save_system_info(){
    echo "./SysMonitor.sh --save" >> "$logfile"

    cat <<EOF > "sysinfo.log"  # La redirection doit être placée ici, dès l'ouverture du here-do
[----- Informations Système -----]
Date : $(date)
Nom de la machine : $(hostname)
Noyau : $(uname -r)
Uptime : $(uptime)

[----- Utilisateurs connectés -----]
$(who)

[----- Espace Disque -----]
$(df -h)

[----- Mémoire -----]
$(free -h)
EOF

    # Détails des fonctions utilisées dans le programme :
    # date : Affiche la date et l'heure du système.
    # hostname : Récupère le nom de l'hôte.
    # uname -r : Affiche la version du noyau Linux.
    # uptime : Affiche depuis combien de temps la machine est allumée.
    # who : Affiche les utilisateurs actuellement connectés.
    # df -h : Affiche l'espace disque utilisé et disponible.
    # free -h : Affiche la mémoire vive utilisée/disponible en format lisible.
    echo "📁 Informations systèmes sauvegardées dans le fichier sysinfo.log."|tee -a "$logfile"
}
```

Exemple d'Utilisation

Résultat affiché dans l'invite de commande par la commande ***./Sysmonitor.sh --save*** :

```
osboxes@osboxes:~/programmation_shell/projet$ ./SysMonitor.sh --save
📁 Sauvegarde des informations système...
📁 Informations systèmes sauvegardées dans le fichier sysinfo.log.
```

Cela nous donne le fichier « ***sysinfo.log*** » suivant :

```
[----- Informations Système -----]
Date : Mon Apr 21 05:18:56 AM EDT 2025
Nom de la machine : osboxes
Noyau : 6.11.0-21-generic
Uptime :  05:18:56 up  1:14,  1 user,  load average: 0.31, 0.48, 0.42

[----- Utilisateurs connectés -----]
osboxes  seat0        2025-04-21 04:04 (login screen)
osboxes  tty2         2025-04-21 04:04 (tty2)

[----- Espace Disque -----]
Filesystem      Size  Used Avail Use% Mounted on
tmpfs           2.0G  1.6M  2.0G   1% /run
/dev/sda2       492G   16G  451G   4% /
tmpfs           9.6G   26M  9.6G   1% /dev/shm
tmpfs           5.0M  8.0K  5.0M   1% /run/lock
tmpfs           2.0G  132K  2.0G   1% /run/user/1000

[----- Mémoire -----]
               total        used        free      shared  buff/cache   available
Mem:            19Gi       2.9Gi        14Gi       110Mi       2.1Gi        16Gi
Swap:          4.0Gi          0B       4.0Gi
```

Fonction ***launch_bg()*** : Lancer le cmd en arrière-plan (-bg cmd)

Description

La fonction « ***launch_bg()*** » permet le lancement d'une commande donnée en paramètre, en arrière-plan sur le terminal.

Code

```bash
launch_bg() {
    # Vérifications des arguments donnés, il doit y avoir 2 ou plus arguments.
    if [[ $# -ge 2 ]]; then
        # # Création d'un tableau contenant tout les arguments du 2 jusqu'à la fin
        commande=("${@:2}")

        # Execution de la commande en arrière plan, redirigeant les potentielles erreurs retourn
        # "${commande[@]}" 2>/dev/null &

        {
            # echo "----- Début de la commande : ${commande[*]} -----"
            "${commande[@]}"
            # echo "----- Fin de la commande : ${commande[*]} -----"
        } 2>&1 | tee -a "$logfile" &


        # # Affichage d'un message de confirmation.
        echo "🚀 La commande '${commande[*]}' a été lancée en arrière-plan." | tee -a "$logfile"
    else
        # #Affichage d'un message d'erreur en cas de mauvaise utilisation
        echo "❌ Utilisation : -bg <commande> (ex : -bg sleep 10)" | tee -a "$logfile"
    fi
}
```

Exemple d'utilisation

```
osboxes@osboxes:~/programmation_shell/projet$ ./SysMonitor.sh -bg ls
🚀 La commande 'ls' a été lancée en arrière-plan.
SysMonitor(1).sh
SysMonitor(2).sh
SysMonitor.sh
SysMonitor_Matthieu.sh
sysinfo.log
sysinfo.txt
```

Dans cet exemple :

L'option « ***-bg ls*** » permet l'exécution en arrière-plan de la commande ls.

On peut voir sur la capture d'écran :

- Confirmation du lancement de la commande
- Le résultat de la commande

Fonction ***stop_process()*** : **Suspendre** le processus pid (-stop pid)

Description

La fonction « ***stop_process()*** » permet de mettre en pause un processus en train de fonctionner. Il suffit simplement de renseigner le pid du processus pour le mettre en pause.

Code

```
stop_process(){
    if [ -z "$1" ]; then # Vérification de l'existence de l'argument $1.
        # S'il n'existe pas, affichage d'un message d'erreur.
        echo "Sysmonitor.sh: -stop: l'option nécessite un PID." | tee -a "$logfile"
        return 1 #Puis renvoi d'une erreur.
    elif ! [[ "$1" =~ ^[0-9] ]]; then # Vérification que le PID soit bien un nombre.
        # S'il n'est pas un nombre, affichage d'un message d'erreur.
        echo "Sysmonitor.sh: -stop: l'option attend un PID, pas un nom." | tee -a "$logfile"
        return 1 #Puis renvoi d'une erreur.
    # Vérification de l'existence du PID (qu'il corresponde bien à un processus en fonctionnemen
    elif ! kill -0 "$1" 2>/dev/null; then
        # S'il ne correspond pas à un processus fonctionnel, affichage d'un message d'erreur.
        echo "Sysmonitor.sh: -stop: le PID $1 est introuvable." | tee -a "$logfile"
        return 1 #Puis renvoi d'une erreur.
    # Tentative d'exécuter la commande kill -STOP pour mettre en pause le processus associé au P
    elif kill -STOP "$1" 2>/dev/null; then
        # Message de bonne réussite du programme.
        echo "Le processus possedant le PID $1 est maintenant suspendu." | tee -a "$logfile"
    else
        # S'il n'arrive pas à l'arrêter alors message d'erreur.
        echo "Sysmonitor.sh: -stop: Permissions insuffisantes pour suspendre le processus $1." |
        return 1 # Puis renvoi d'une erreur.
    fi
}
```

Exemple d'Utilisation

```
osboxes@osboxes:~/Desktop/Exo_Shells/Projet_Shell$ ./stop.sh -stop 9165
Le processus possedant le PID  est maintenant suspendu.

[2]+  Stopped                 sleep 10000000000000000
```

Fonction ***continue_process()*** : Relancer le processus pid (-cont pid)

Descripiton

La fonction « ***continue_process()*** » s'occupe de relancer en arrière-plan un processus actuellement suspendu. Elle prend en paramètre l'identifiant du processus.

Code

```
continue_process(){
    {
        # Vérification si on a mis un argument ou non
        process=$1
            if [[ -z $process ]]
            then    echo "Vous n'avez pas saisi le pid "
            return 1
            else    echo "PID saisi: $process"

            fi
        # Vérification de l'existance du PID parmi la liste des processus suspendus (numeros de pid)

        jobs -s -p | grep -w ${process} #-w pour chercher le terme exact
        # retour=$?
        # #récupération du code de retour de la commande grep (0 si pid trouvé dans la liste -existe- sinon 1 -n'existe pas)
        if [[ $? -eq 0 ]]
        then    #jobs -l | grep -w ${process}

            # match seulement le mot exact!! ici on veut récupérer le job associé au processus.
            # Ne pas mettre des '' pour le grep -w avec variable

            # On filtre pour avoir seulement le numero de job
            n_job=$(jobs -l | grep -w ${process} | awk '{print $1}' | tr -d '[]+')

            # J'ai écrit de cette façon pour qu'on ait un message qui s'affriche si bg réussi.

            # Je ne voulais mettre de if et utiliser $? pour être sûr que je vérifiais la bonne
            # exécution de bg + pour ne pas alourdir le code
            bg %${n_job} && echo "Le processus ${process} a bien été relancé en arrière plan" || echo "Fail relancement du processus en bckgrund"
            return 0
        else    echo "Le processus saisi ne figure pas dans la liste des processus suspendus"
            return 1
        fi
    } 2>&1 | tee -a "$logfile"
}
```

jobs -s -p | grep -w ${process} : Affiche l'état des jobs (tâches). -s permet d'afficher seulement les tâches stoppées et -p permet d'afficher les identifiants des processus associés à ces tâches. Donc, la commande nous affiche les listes des identifiants des processus dont la tâche est suspendue.

grep -w ${process} : permet de chercher le processus qu'on veut tuer dans la liste évoquée précédemment

jobs -l | grep -w ${process} | awk '{print $1}' | tr -d '[]+' :

1) On affiche la liste la plus complète des jobs en terme d'info (numéro de job, pid, état etc)
2) On sélectionne la ligne qui comporte le pid qu'on veut relancer
3) On sélectionne la première colonne car c'est celle qui comporte le numéro job
4) On se débarrasse de l'encadrement []+ afin de récupérer le numéro de job

bg %${n_job} && echo "Le processus ${process} a bien été relancé en arrière plan" || echo "Fail relancement du processus en bckgrund" : On relance le processus suspendu et si la commande réussi on affiche un message de succès sinon on affiche un message d'erreur (operateur AND && et OU ||)

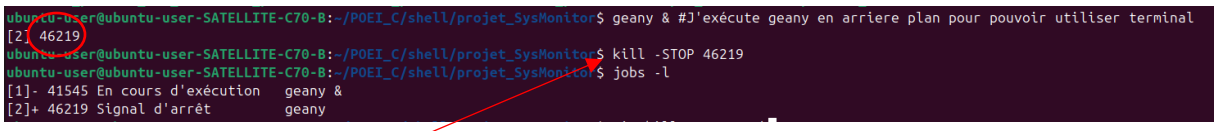
ATTENTION !

Il faut **ABSOLUMENT** lancer le script qui utilise cette fonction avec le mot-clé source dans le terminal !

Exemple : source continue_process_new.sh

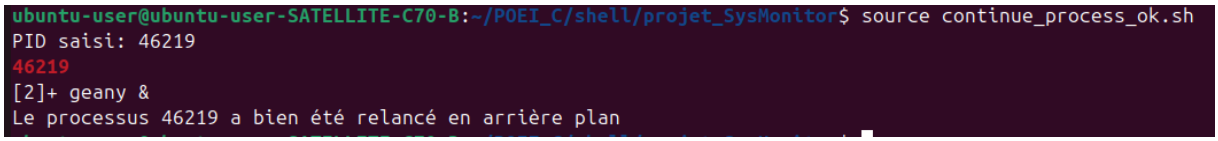
Pourquoi ? ➜ car en exécutant le script avec **./** ou **bash**, cela va créer un shell fils dans le shell père et le shell fils ne sera pas en mesure de voir les jobs du shell du père !!!

Exemple d'Utilisation



On arrête le processus 46219

Ensuite on affiche la liste des jobs avec leurs infos. Je vais dans mon script pour utiliser la fonction sur ce processus



Fonction *kill_process()* : Terminer le processus pid (-kill pid)

Description

La fonction « ***kill_process()*** » permet comme son nom l'indique de tuer un processus, c'est-à-dire y mettre fin. Cette fonction prend en paramètre un pid.

Si l'on ne saisit pas de pid en argument, la fonction indique à l'utilisateur que la saisie est vide.

Dès lors qu'on saisit l'identifiant du processus (pid), la fonction regarde dans la liste des processus actifs s'il est présent. Si c'est le cas, il l'indique à l'utilisateur, tue le processus et retourne un message de succès ou d'echec. Dans le cas où le pid ne figure pas dans la liste, il renvoie un message d'erreur à l'utilisateur et la fonction se termine en renvoyant la valeur 1 (échec).

Code

```
kill_process() {
    pid="$1"

    if [ -z "$pid" ]; then
        echo "❌ Erreur : aucun PID fourni." | tee -a "$logfile"
        return 1
    fi

    if kill -0 "$pid" 2>/dev/null; then
        kill "$pid"
        echo "🔴 Processus $pid terminé avec succès." | tee -a "$logfile"
    else
        echo "❌ Le processus $pid n'existe pas ou vous n'avez pas les droits nécessaires." | tee -a "$logfile"
    fi
}
```

kill ${victim_pid} : On tue le processus

Fonction ***print_help()*** : Afficher l'Aide (-h ou--help)

Description

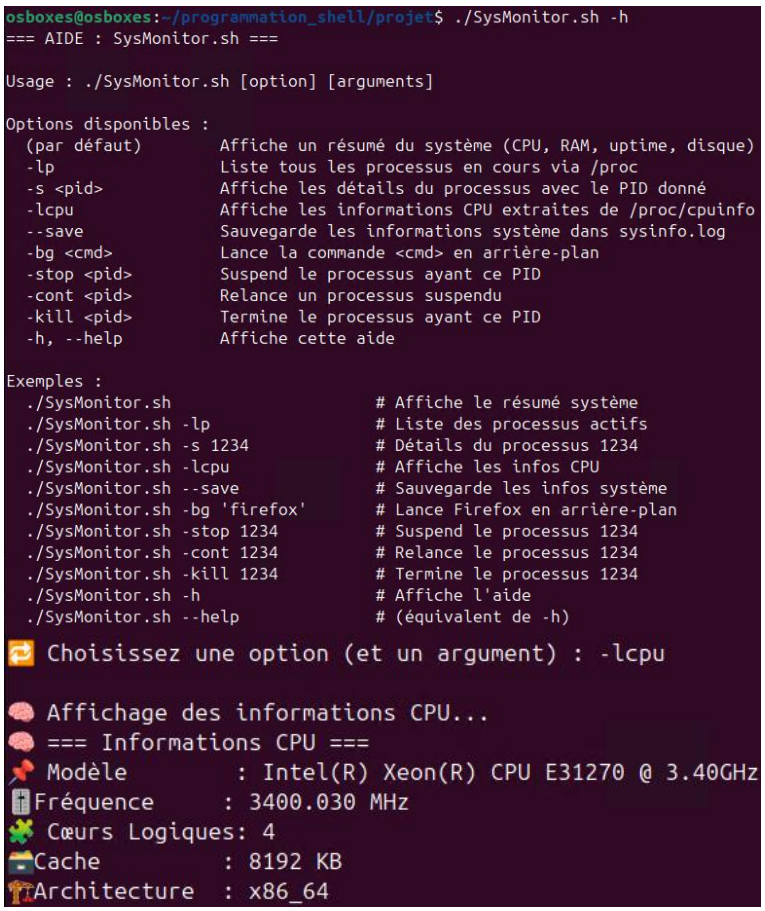
La fonction « ***print_help()*** » affiche un guide complet de l'utilisation du script avec toutes options possibles, en donne des exemples pour chaque option.

Après quoi, l'utilisateur est invité à choisir une option (-lp, -lcpu, etc.) avec un argument si nécessaire. C'est alors qu'une autre fonction se lance, en fonction de l'option (et de l'argument) choisie par l'utilisateur.

Code

```
print_help() {
    {
        # Titre de la section aide
        echo "=== AIDE : SysMonitor.sh ==="
        echo ""

        # Affiche la syntaxe de base pour exécuter le script
        echo "Usage : ./SysMonitor.sh [option] [arguments]"
        echo ""

        # Liste des options disponibles avec une brève description
        echo "Options disponibles :"
        echo "  (par défaut)        Affiche un résumé du système (CPU, RAM, uptime, disque)"
        echo "  -lp                 Liste tous les processus en cours via /proc"
        echo "  -s <pid>            Affiche les détails du processus avec le PID donné"
        echo "  -lcpu               Affiche les informations CPU extraites de /proc/cpuinfo"
        echo "  --save                Sauvegarde les informations système dans sysinfo.log"
        echo "  -bg <cmd>           Lance la commande <cmd> en arrière-plan"
        echo "  -stop <pid>         Suspend le processus ayant ce PID"
        echo "  -cont <pid>         Relance un processus suspendu"
        echo "  -kill <pid>         Termine le processus ayant ce PID"
        echo "  -h, --help          Affiche cette aide"
        echo ""
        # Section exemples pour illustrer chaque option
        echo "Exemples :"
        echo "  ./SysMonitor.sh                    # Affiche le résumé système"
        echo "  ./SysMonitor.sh -lp                # Liste des processus actifs"
        echo "  ./SysMonitor.sh -s 1234            # Détails du processus 1234"
        echo "  ./SysMonitor.sh -lcpu              # Affiche les infos CPU"
        echo "  ./SysMonitor.sh --save             # Sauvegarde les infos système"
        echo "  ./SysMonitor.sh -bg 'firefox'      # Lance Firefox en arrière-plan"
        echo "  ./SysMonitor.sh -stop 1234         # Suspend le processus 1234"
        echo "  ./SysMonitor.sh -cont 1234         # Relance le processus 1234"
        echo "  ./SysMonitor.sh -kill 1234         # Termine le processus 1234"
        echo "  ./SysMonitor.sh -h                 # Affiche l'aide"
        echo "  ./SysMonitor.sh --help             # (équivalent de -h)"
        echo ""
    } | tee -a "$logfile"  # Affiche à l'écran et enregistre dans le fichier de log

    # Invite l'utilisateur à saisir une nouvelle ligne de commande (option valide)
    read -p "🔁 Choisissez une option (et un argument) : " input
    echo "" | tee -a "$logfile"  # Saut de ligne

    # Met à jour les arguments du script avec ceux saisis par l'utilisateur
    set -- $input  # Permet de gérer des options avec arguments comme -s 1234

    # Relance le switch principal avec les nouveaux arguments
    MAIN "$@"
}
```

Exemple d'Utilisation



Fonction *invalid_option()* : Gestion des Erreurs

Description

La fonction « ***invalid_option()*** » est appelée lorsque l'utilisateur entre une option incorrecte ou inconnue dans le script ***SysMonitor.sh***.

Elle affiche un message d'erreur et propose d'entrer l'option « *-h* » ou « *--help* » pour connaître les (autres) options possibles (et les types d'arguments à donner avec).

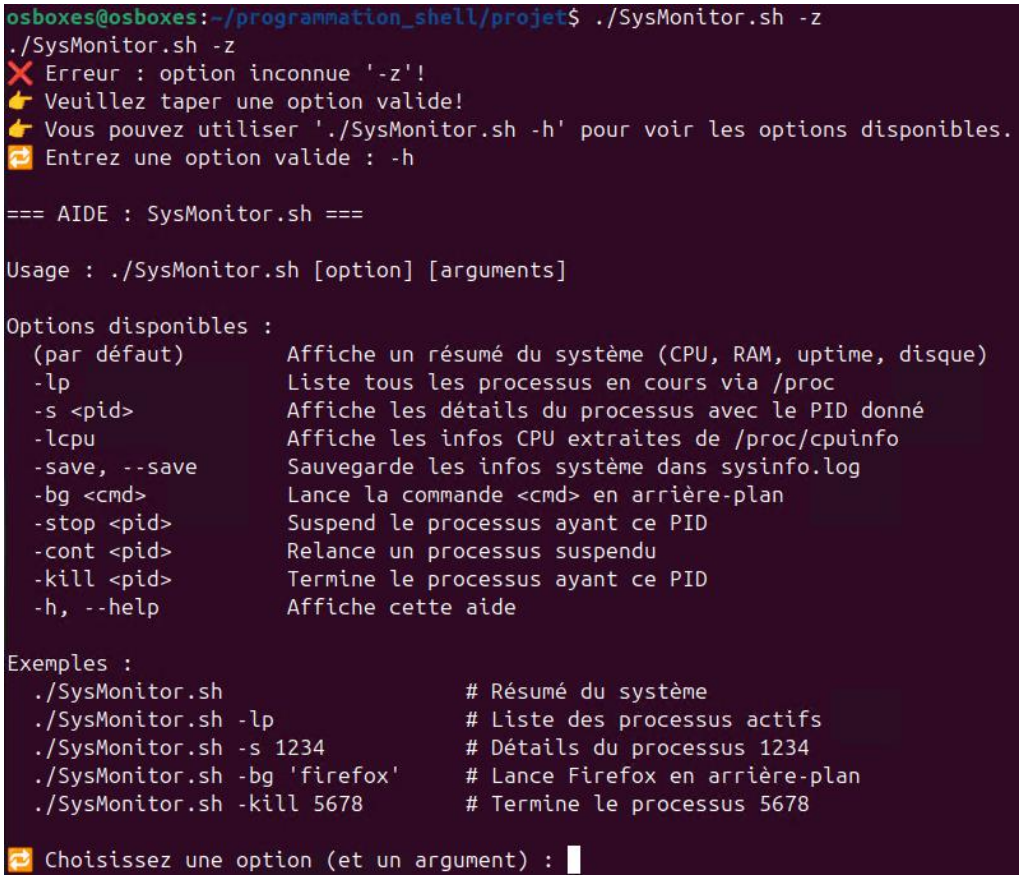
Code

```
# Fonction pour gérer une option invalide et redemander une entrée à l'utilisateur
invalid_option() {
    # Enregistre dans le fichier de log la commande entrée par l'utilisateur
    echo "./SysMonitor.sh $1" | tee -a "$logfile"

    # Affiche un message d'erreur indiquant que l'option est inconnue
    echo "❌ Erreur : option inconnue '$1'!" | tee -a "$logfile"

    # Invite l'utilisateur à saisir une option correcte
    echo "👉 Veuillez taper une option valide!" | tee -a "$logfile"
    echo "👉 Vous pouvez utiliser './SysMonitor.sh -h' pour voir les options disponibles." | tee

    # Invite l'utilisateur à saisir une nouvelle ligne de commande (option valide)
    read -p "🔁 Entrez une option valide : " input
    echo "" | tee -a "$logfile"  # Saut de ligne

    # Met à jour les arguments du script avec ceux saisis par l'utilisateur
    set -- $input  # Permet de gérer des options avec arguments comme -s 1234

    # Relance le switch principal avec les nouveaux arguments
    MAIN "$@"
}
```

Exemple d'Utilisation

```
osboxes@osboxes:~/programmation_shell/projet$ ./SysMonitor.sh -z
./SysMonitor.sh -z
❌ Erreur : option inconnue '-z'!
👉 Veuillez taper une option valide!
👉 Vous pouvez utiliser './SysMonitor.sh -h' pour voir les options disponibles.
🔁 Entrez une option valide :
```

➔ Dans cet exemple, nous avons entré l'option « *-z* » mais qui n'est pas reconnue par le script ;
➔ Le programme affiche donc un message d'erreur et propose d'utiliser l'option « *-h* » pour voir les (autres) options possibles.



➔ Nous avons entré l'option « *-h* » ; ce qui lance la fonction « Fonction ***print_help()*** : Afficher l'Aide (-h ou --help) » : voir plus haut

Fonction MAIN

Description

La fonction ***MAIN*** est la fonction principale du script pour lancer le programme.

C'est ici que sont gérés et interprétés les options et les arguments donnés lors de l'exécution du programme ; c'est dans cette fonction que le programme appel les différentes fonctions vues plus haut, en fonction de l'option (et de l'argument) donnée par l'utilisateur.

Code

```
MAIN() {
    while true; do  # Boucle pour analyser les arguments passés au script
        case "$1" in
            # Option : -lp → Liste tous les processus actifs
            -lp)
                list_processes  # Appelle la fonction qui affiche la liste des processus
                break  # Sort de la boucle après exécution
                ;;

            # Option : -s <pid> → Affiche les détails d'un processus spécifique
            -s)
                echo "./SysMonitor.sh -s $2" >> "$logfile"
                if [ -n "$2" ]; then
                    process_details "$2"  # Si un PID est fourni, affiche ses détails
                else
                    echo "❌ Erreur : veuillez fournir un PID après -s" | tee -a "$logfile"
                fi
                break
                ;;

            # Option : -lcpu → Affiche les informations CPU
            -lcpu)
                echo "🧠 Affichage des informations CPU..."
                infos_cpu  # Appelle la fonction qui affiche les infos CPU
                break
                ;;

            # Option : --save → Sauvegarde les informations système dans un fichier
            --save)
                echo "📁 Sauvegarde des informations système..."
                save_system_info  # Appelle la fonction de sauvegarde
                break
                ;;
            # Option : -bg <cmd> → Exécute une commande en arrière-plan
            -bg)
                echo "./SysMonitor.sh -bg $2" >> "$logfile"
                if [ -n "$2" ]; then
                    launch_bg "$@"  # Exécute la commande entière passée après -bg
                else
                    echo "❌ Erreur : veuillez spécifier une commande après -bg"  | tee -a "$log
                fi
                break
                ;;

            # Option : -stop <pid> → Suspend un processus en envoyant le signal STOP
            -stop)
                echo "./SysMonitor.sh -stop $2" >> "$logfile"
                if [ -n "$2" ]; then
                    stop_process "$2"  # Suspend le processus avec le PID donné
                    # stop_process $2
                else
                    echo "❌ Veuillez fournir un PID après -stop" | tee -a "$logfile"
                fi
                break
                ;;
```

```
            # Option : -cont <pid> → Relance un processus suspendu (signal CONT)
            -cont)
                echo "./SysMonitor.sh -cont $2" >> "$logfile"
                if [ -n "$2" ]; then
                    continue_process "$2"  # Relance le processus
                else
                    echo "❌ Veuillez fournir un PID après -cont" | tee -a "$logfile"
                fi
                break
                ;;

            # Option : -kill <pid> → Termine un processus en envoyant le signal KILL
            -kill)
                echo "./SysMonitor.sh -kill $2" >> "$logfile"
                if [ -n "$2" ]; then
                    kill_process "$2"  # Tue le processus spécifié
                else
                    echo "❌ Veuillez fournir un PID après -kill" | tee -a "$logfile"
                fi
                break
                ;;

            # Option : -h ou --help → Affiche le guide d'utilisation
            -h|--help)
                echo "./SysMonitor.sh $1" >> "$logfile"  # Log l'utilisation de l'aide
                print_help  # Affiche le message d'aide
                break
                ;;
            # Aucune option passée → Affiche un résumé système (ex. : infos CPU par défaut)
            "")
                cpu_info  # Affiche les informations système de base
                break
                ;;

            # Option non reconnue → Affiche une erreur
            *)
                invalid_option "$1"  # Appelle la fonction de gestion d'erreurs
                return  # Termine la fonction proprement
                ;;
        esac
    done
}

# Lancement de la fonction principale
MAIN "$@"
```

Fichier *sysinfo.txt*

À chaque exécution de la commande « ***./SysMonitor.sh [option] [arguments]*** », le résultat de la commande affiché est stocké dans un fichier « ***sysinfo.txt*** ».

Note : Le fichier ***sysinfo.txt*** est (ré)initialisé à chaque nouvelle exécution « ***./SysMonitor.sh*** ».

Code

Le fichier ***sysinfo.txt*** est (ré)intialisé au tout début du programme (juste avant l'implémentation de la fonction « par défaut » (***cpu_info***)).

```
# Initialiser le fichier de log d'exécution
logfile="sysinfo.txt"
echo "=== Résultat d'exécution - $(date) ===" > "$logfile"
echo "" >> "$logfile"
```

Quand une fonction affiche un message à l'écran (sur le terminal) ces messages ont tous (ou presque) ajouté dans le fichier ***sysinfo.txt*** ; pour ce faire, le programme utilise 2 syntaxes différentes :

1) Ajouter tout un ensemble en 1 fois (exemple avec la fonction « ***list_processes()*** ») :

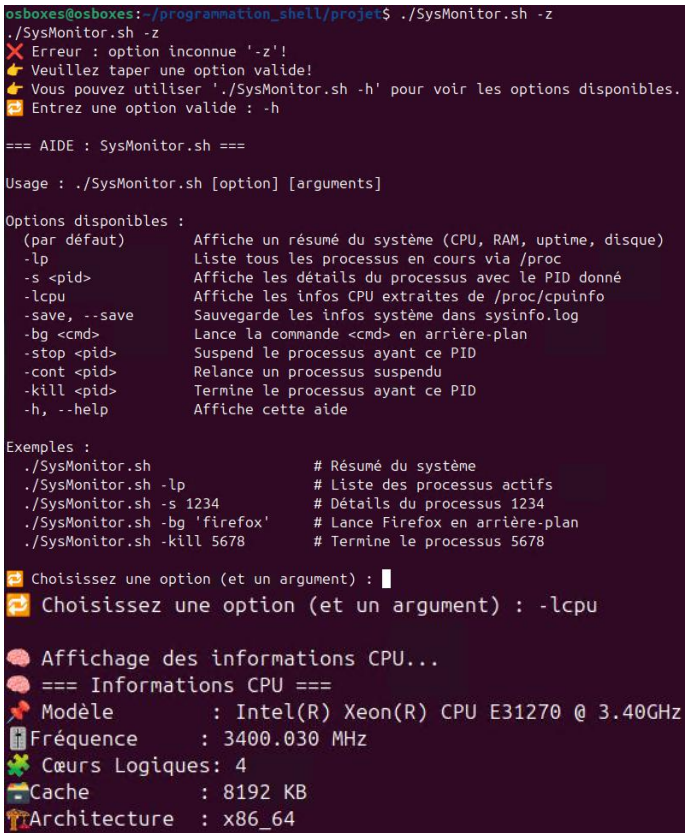
```
list_processes(){
    echo "./SysMonitor.sh -lp" >> "$logfile"
    {
        # Titre de section
        echo "=== Liste des processus en cours ==="

        # En-têtes de colonnes : PID, nom du processus, état
        printf "%-10s %-25s %-10s\n" "PID" "Nom" "État"
        echo "-------------------------------------------"

        # Boucle sur tous les dossiers numériques de /proc (correspondant aux PIDs)
        for pid in /proc/[0-9]*; do
            # Vérifie que le fichier status existe (donc qu'il s'agit bien d'un processus)
            if [ -f "$pid/status" ]; then
                # Récupère le PID depuis le nom du dossier
                PID=$(basename "$pid")

                # Extrait le nom du processus depuis le fichier status
                NOM=$(grep -s "^Name:" "$pid/status" | awk '{print $2}')

                # Extrait l'état du processus (R = en cours, S = en veille, Z = zombie, etc.)
                ETAT=$(grep -s "^State:" "$pid/status" | awk '{print $2}')

                # Affiche les informations du processus formatées proprement
                printf "%-10s %-25s %-10s\n" "$PID" "$NOM" "$ETAT"
            fi
        done

        echo "" # Saut de ligne
    } | tee -a "$logfile"  # Affiche à l'écran ET enregistre dans le fichier log (sysLog.txt)
}
```

2) Ajouter 1 ligne à la fois (exemple avec la fonction « ***invalid_option()*** ») :

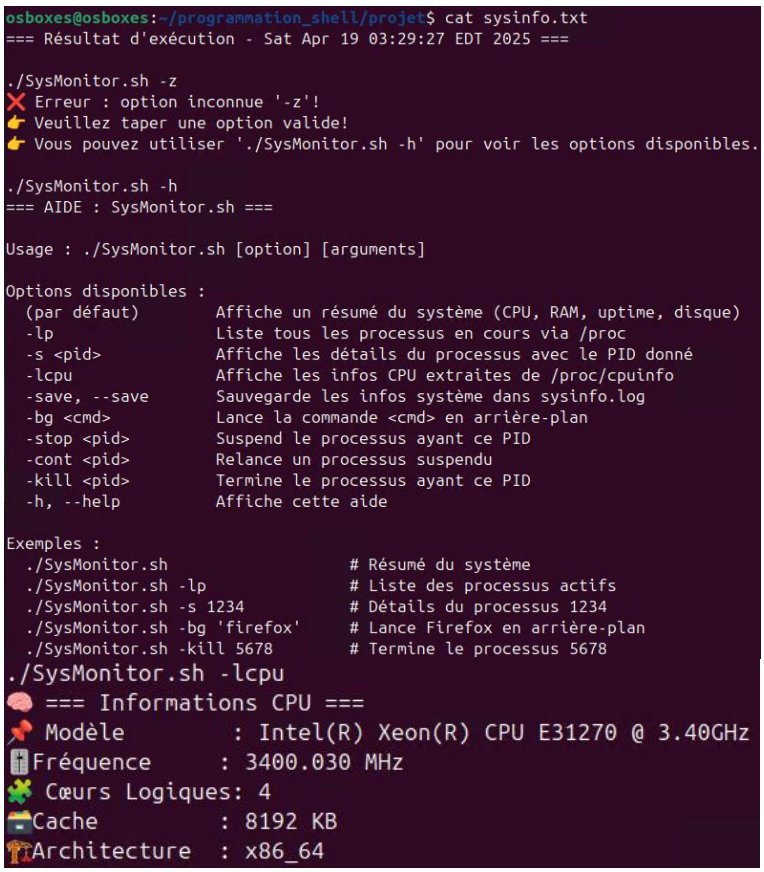
```
# Fonction pour gérer une option invalide et redemander une entrée à l'utilisateur
invalid_option() {
    # Enregistre dans le fichier de log la commande entrée par l'utilisateur
    echo "./SysMonitor.sh $1" | tee -a "$logfile"

    # Affiche un message d'erreur indiquant que l'option est inconnue
    echo "❌ Erreur : option inconnue '$1'!" | tee -a "$logfile"

    # Invite l'utilisateur à saisir une option correcte
    echo "👉 Veuillez taper une option valide!" | tee -a "$logfile"
    echo "👉 Vous pouvez utiliser './SysMonitor.sh -h' pour voir les options disponibles."

    # Invite l'utilisateur à saisir une nouvelle ligne de commande (option valide)
    read -p "🔁 Entrez une option valide : " input
    echo "" | tee -a "$logfile"  # Saut de ligne

    # Met à jour les arguments du script avec ceux saisis par l'utilisateur
    set -- $input  # Permet de gérer des options avec arguments comme -s 1234

    # Relance le switch principal avec les nouveaux arguments
    MAIN "$@"
}
```

Exemple de Fichier « *sysinfo.txt* »

Soit la commande exécutée suivante :



Cela nous donne le fichier « ***sysinfo.txt*** » suivant :



Conclusion

Le script « ***SysMonitor.sh*** » est un bon outil pour surveiller et gérer les processus système sur un ordinateur Linux ; il permet d'interagir avec les informations système de manière détaillée. Grâce aux différentes options disponibles, il permet de lister les processus, de sauvegarder des informations, ou même de manipuler les processus en arrière-plan.