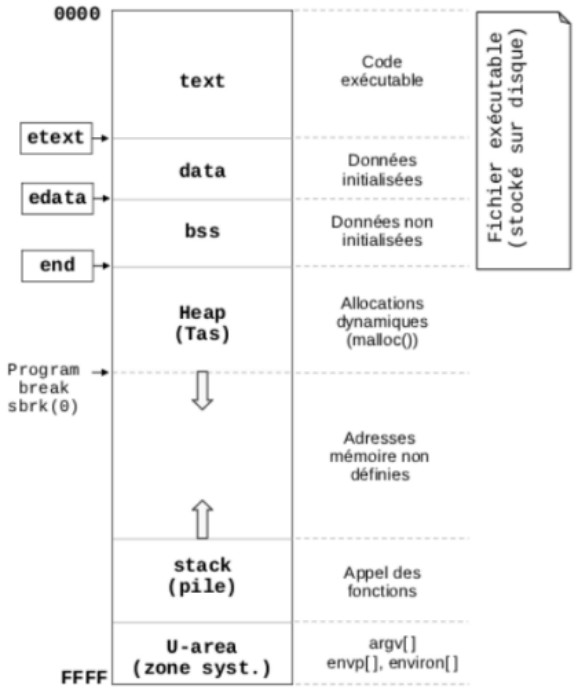
# Processus et programmation système

lundi 28 novembre 2022

1. Processus
   - -1- Programme : passif, statique, inerte
   - -2- Processus : dynamique, cycle de vie, accès au CPU et à la mémoire
   - -3- Commande PS
     1- PID : identification
     2- TYY : terminal
     3- STAT : état du processus
     4- COMMAND
   - -4- Priorité d'un processus
     1- La valeur est entre -20 et 19 (faible)
   - -5- Mémoire virtuelle
     1- Permet d'exécuter un programme partiellement qui est chargé en mémoire centrale
     2-




     3- Segment text (0-etext) : code
     4- Segment data (etext-edata) : Données initialisées au chargement du processus et Données locales statiques des fonctions.
     5- Segment BSS (block started by symbol) (edata-end) : Données non-initialisées
     6- Head (end-sbrk()) : Allocation dynamique, malloc()
     7- Stack (...-adresse max) : Pile d'exécution, rappel

2. Programmation système

-1- **pid_t getpid()**

-2- pid_t getppid() : pid de parent

-3- uid_t getuid()

-4- gid_t getgid()

-5- pid_t getsid() : numéro de session

-6- pid_t getpgid() : numéro du groupe du processus

-7- char* getenv(const char* var)

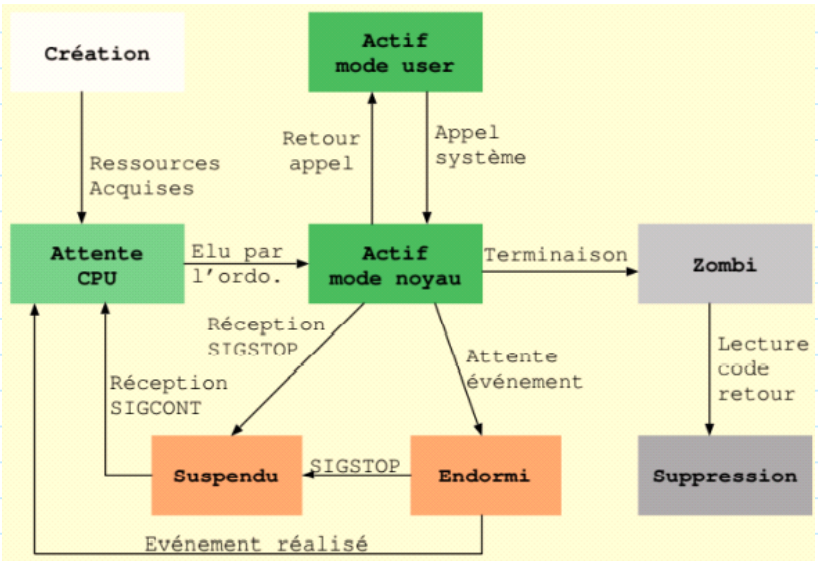
-8- int putenv(chonst char * chaîne), int setenv(const char* nom, const char * valeur, int écraser) : créer une variable d'environnement

1- Le troisième argument pour indiquer si la variable doit être écrasée (écraser!=null) dans le cas où elle existe déjà.

-9- unsetenv() : supprimer

-10- Appel système

1-




2- Deux modes d'exécution

1> Mode utilisateur

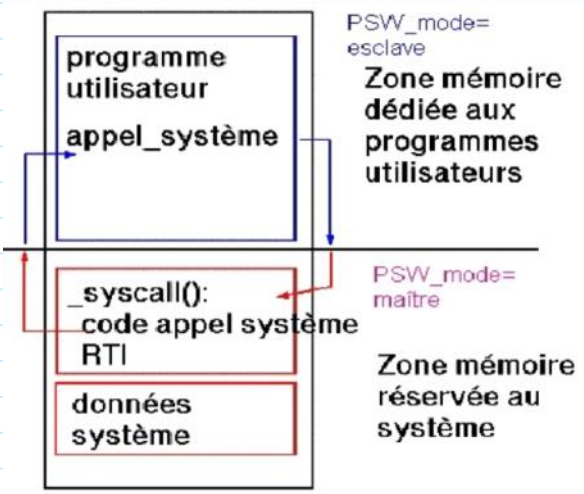
2> Mode système

a) Dans ce mode, le processus utilise des **instructions privilégiées** et accède à l'ensemble du système

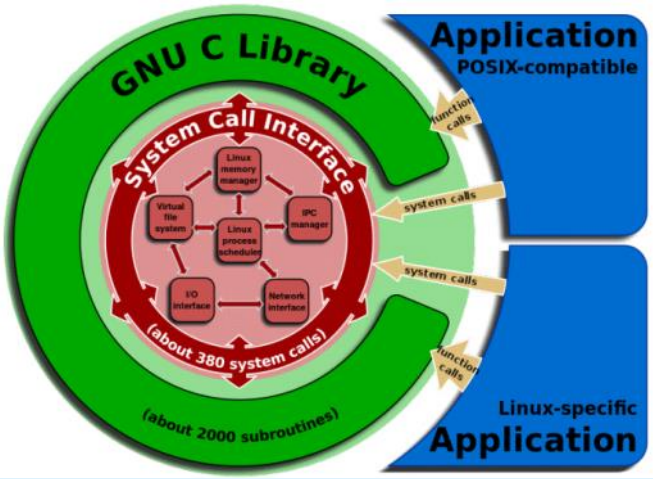
3- Appel système est réalisée par une paire d'instruction

1> system_call() // appel au système

2> RTI // retour au programme utilisateur

3> 


4- Librairies

1> 


3. Les signaux

-1- La plupart des signaux sont émis par le noyau en réponse à des conditions logicielles ou matérielles particulières

1- Contrôle des processus

2- Gestion asynchrone des entrées-sorties

3- Communication (simple) inter-processus

-2- Kill

1- STOP Signal : STOP

1> kill -STOP $(pidof xeyes)

2- CONT Signal : pour reprendre l'exécution qui est déjà suspendu

1> kill -CONT $(pidof xeyes)

-3- int kill(pid_t pid, int signal), void raise(int signal), int killpg(pid_t pgid, int signal) : **émission** un signal

1- Si *signal* est nul, le signal est envoyé à tous les processus du groupe auquel appartient le processus appelant

2- Si *signal* est négatif et sauf pour –1, le signal est envoyé à tous les processus du groupe dont le PGID est égal à la valeur absolue de cet argument *signal*

-4- int pause(void) : **attente** un signal

-5- int sigaction(int sig, struct sigaction *new_handler, struct sigaction *old_handler) : **installation** d'un nouveau handler

1- *sig* : désigne le signal pour lequel on veut installer un nouveau handler

2- *new_handler* : pointe sur la structure sigaction à utiliser. La prise en compte du signal entraînera l'exécution de la fonction $new_{handler} \rightarrow sa_{handler}$. Si de plus, la fonction n'est ni SIG_DFL, ni SIG_IGN, ce signal ainsi que ceux contenus dans la liste $new_{handler} \rightarrow sa_{mask}$, seront masqués pendant le traitement.

3- *old_handler* : pointe sur une structure sigaction qui contient les anciennes options du traitement du signal.

4- Structure de sigaction

1>
```
struct sigaction {
        void (*sa_handle)(); // pointeur sur handler ou SIG_DFL ou
SIG_IGN
        sigset_t sa_mask; // liste signaux supplémentaires à bloquer
éventuellement
        int sa_flags; // indicateurs optionnels
}
```