

SR01 - TD 7

Premiers appels système

Exercice1: getpid() et getppid()

Créer un programme C, qui utilise les appels système `getpid()` et `getppid()`. ajouter ensuite les appels `getuid()`, `getgid()`, ...

Exercice2: E/S bas niveau [`STDIN_FILENO`, `STDOUT_FILENO`, `/dev/tty`, `dup()`, `dup2()`, *table des descripteurs*]

cp ~sr01/td/closep.c .

Examiner le source, le compiler et l'exécuter.

Ajouter des lignes (**sans en supprimer**) pour que le dernier `printf()` (le 4^{ème}) se fasse bien à l'écran.

Vous réaliserez ce travail **deux fois** en utilisant deux méthodes différentes :

- avec `open()` -> **closep1.c**
- avec `dup()` -> **closep2.c**

Exercice3: L'espace virtuel d'un process

Sur une machine à mémoire virtuelle paginée la mémoire est vue par les programmes comme un grand tableau linéaire.

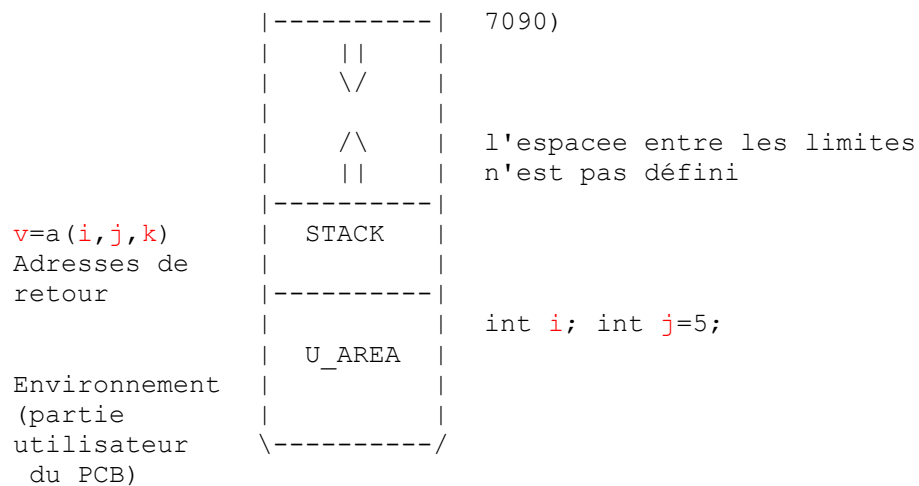
Attention : ce tableau comporte en général des TROUS : des portions de l'espace virtuel NE SONT PAS DÉFINIS.

Tout accès dans ces portions se traduit sur Unix par un "segmentation fault" suivi en général par la génération d'un "core".

Organisation en mémoire d'un programme :

	/-----\	
	-----	*
main(), ...	TEXT	aussi appelé CODE

int j=5;	DATA	Y sont aussi stockées les
		chaines de caract. de
	-----	printf, ...
static int i;	BSS	
	-----	(Block Started by Symbol,
malloc()	HEAP	Terme du
		langage assembleur de l'IBM



❓* Les trois sections en couleur orange sont celles qui composent un programme exécutable sur disque. Elles sont générées par les phases successives de compilation puis Edition de liens (link). Elles sont chargées dans l'espace virtuel d'un process lorsqu'on exécute le programme. Ceci est illustré par l'exemple ci-dessous :

- Ecrire le programme C suivant :

```
#include <stdio.h>
main()
{
printf("Bonjour a tous !\n");
}
```

- Le compiler avec le compilateur "cc" (le compilateur C standard du système). *Il ne faut pas utiliser le compilateur "domaine public gcc" pour illustrer cet exemple.*
En effet, "gcc" réalise une optimisation du code généré qui place dans la partie .TEXT certaines constantes simples, au lieu de les placer dans .DATA.
- Visualiser le contenu du programme "hello" obtenu avec la commande `objdump` :
 - `objdump -h hello` pour connaître les adresses des différentes sections (.TEXT, .DATA, .BSS).

- o `objdump -s hello | more` pour visualiser le contenu des sections.
- o `objdump -d hello | more` pour "désassembler" le contenu de la section `.TEXT` et visualiser les instructions qui la composent.

Dans quelle section est située la chaîne "Bonjour a tous !" ?

- Utiliser la commande `size` (`size -A`) pour visualiser les tailles des différentes sections

Variables d'environnement

Variables d'environnement et du shell : référencées par `$nom`
 voir la liste des variables : `echo ${^D}`
 Voir le contenu d'une variable : `echo $nom`

Variables "shell"

- Noms en minuscule par convention
- N'existent QUE SOUS le shell
- Etablir : `"set var=val"`, détruire : `"unset var"`

Variable d'environnement

- Noms en majuscules par convention
- Font partie du process
- Sont héritées par les process fils créés par un process père (==> récupérables dans les programmes)
- Cas de `DISPLAY` (test) et de `PATH` (att. aux noms de vos exec !)
- Etablir : `"setenv VAR val"`, détruire : `"unsetenv VAR"`
- Voir l'environnement : `"printenv"`

Exercice4: Les arguments de la ligne de commande Unix


Ecrire un programme qui affiche le nombre et la liste des variables d'environnement;

- **"arg1.c"** : qui utilise l'argument `envp` de la fonction `main()`
- **"arg2.c"** : qui utilise la variable globale `environ`

Exercice5: `getenv()` `putenv()`

Créer dans le shell une variable d'environnement (par exemple "SR01TD8") ayant pour valeur "Un_exemple_de_variable".

écrire un programme "env1.c" qui va successivement :

- demander le nom de la variable  l'utilisateur
- lire la valeur de cette variable à l'aide de la fonction système `getenv()` ,
- modifier cette variable par `putenv()`, en lui donnant la valeur "Un_exemple_de_variable_modifiee" ,
- lire à nouveau la valeur de la variable à l'aide de la fonction système `getenv()`.

à la sortie du programme on vérifiera dans le shell la valeur de la variable.