Programmation Système
TP n° 5 : projection en mémoire
22-23 février 2016

Exercice 1 : « `cat` » et « `cp` »avec `mmap()`

1. Écrire un programme qui projette en mémoire (avec `mmap()`) un fichier dont le nom lui est passé en paramètre, puis affiche son contenu sur sa sortie standard ***sans*** utiliser de tampon *(buffer)*.
2. Écrire un programme qui prend deux paramètres *src* et *dest*, et qui copie le fichier de nom *src* vers le fichier de nom *dest* sans utiliser de tampons ni les appels système `read()` et `write()`. Attention aux tailles relatives des fichiers – `ftruncate()` est votre ami.
3. Comparer les vitesses d'exécution de ces deux programmes avec la version « usuelle » à base de `read()` et `write()` bufferisés, pour un fichier source de grande taille (créé par exemple à l'aide de « `dd if=/dev/zero of=/tmp/grosFichier bs=1m count=100` »).

Exercice 2 : communication par mémoire partagée

1. Écrire un programme qui crée un fichier `synchro` de 4 octets de zéros, qui le `mmap()`pe, puis qui crée un nouveau processus. Le fils attend alors 10 secondes, puis écrit son *pid* sous forme binaire dans le fichier `synchro` sans utiliser l'appel `write()` (il faudra utiliser un pointeur de type `volatile int*`, qu'on supposera atomique). Le père attend qu'une valeur différente de 0 apparaisse dans le fichier, puis affiche celle-ci et termine. N'oubliez pas de faire un appel à `msync` au bon moment.
2. Modifier votre programme pour qu'aucun fichier ne soit créé.

Exercice 3 : `mmap()` avec déplacement

1. Écrire un programme qui affiche le nombre d'occurrences de l'octet 42 dans le fichier passé en paramètre de ligne de commande, sans utiliser de tampon.
2. Modifier votre programme pour qu'il ne `mmap()`pe jamais plus de 128 ko à un moment donné, quelle que soit la taille du fichier analysé [1].

1. Les architectures à 32 bits limitent l'espace mémoire d'un processus à 2 ou 4 Go selon le système d'exploitation. Sur ces architectures, `mmap()` est donc limité à quelques gigaoctets par processus, et les programmes qui manipulent de gros fichiers doivent faire plusieurs appels à `mmap()`.