**TD Unix**

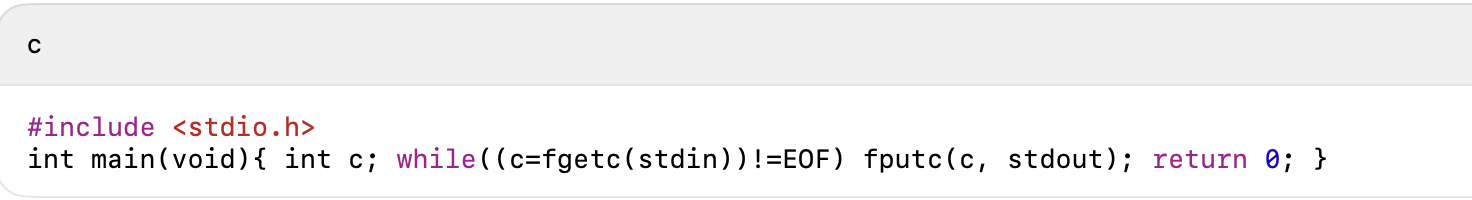
**Système de fichiers**

**INFO - 2éme année**

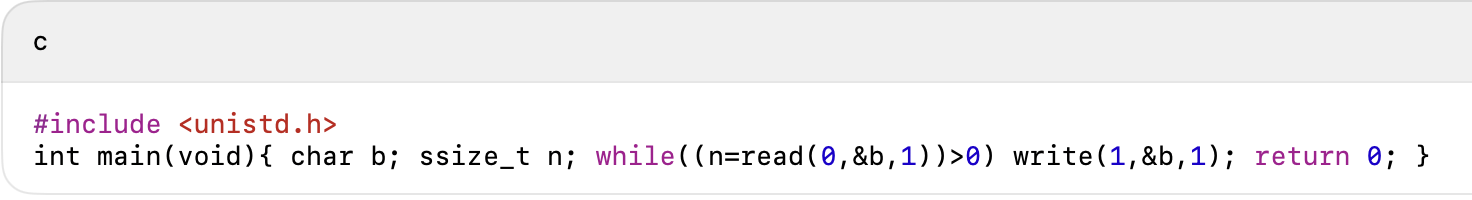
1 Entrées/sorties

**1) Deux versions de cat**

* **Version stdio (fread/fwrite, 1 octet) – principe :**

****

* (équivalente à lire/écrire des blocs de 1 caractère avec fread/fwrite).
* **Version syscalls (read/write, 1 octet)** – principe :



* (strictement en appels système).
* **2) Mesure avec time et fichier > 1 Mo**
* La commande proposée dans l’énoncé convient ; tu compares les deux binaires sur le même fichier redirigé en entrée/sortie : time ./my\_cat < gros\_fichier > /dev/null.

**3) Interprétation des résultats**

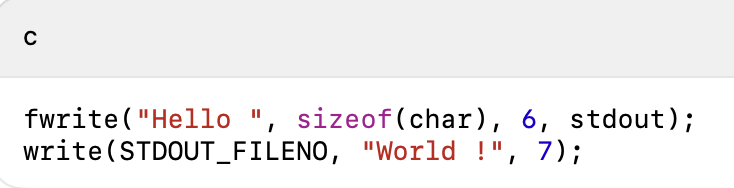
* En lisant/écrivant **un octet à la fois**, **fread/fwrite** (stdio) **vont en général être plus rapides** que **read/write** car la bibliothèque C **tamponne** en user-space et **réduit le nombre d’appels système** (donc moins de basculements noyau).
* **Commandes de plus bas niveau ?** **read/write** sont **plus bas niveau** que fread/fwrite (qui sont des enveloppes tamponnées au-dessus).

Conclusion attendue : **stdio > syscalls** pour 1 octet/bloc, à cause du buffering ; mais si tu écrivais toi-même des **gros blocs** avec read/write (p. ex. 4–64 KiB), les perfs seraient similaires voire meilleures que fread/fwrite.

**4) Programme mixte**

**fwrite + write**

Code fourni :



**Observation attendue :**

* Sur **terminal** (stdout souvent *line-buffered*), tu verras typiquement Hello World ! (dans l’ordre logique).
* **Redirigé vers un fichier/pipe**, write() écrit **tout de suite** dans le noyau alors que fwrite() peut **rester en tampon** tant qu’on ne fflush(stdout)/exit. Il est donc possible d’observer **World !Hello** (ordre inversé) si le tampon stdio n’est pas vidé avant la fin.
* Si tu ajoutes fflush(stdout); après fwrite, l’ordre redevient déterministe.

**Est-ce cohérent avec l’interprétation précédente ?** Oui : la différence d’ordre/perfs **vient précisément du tamponnage stdio** vs. l’écriture immédiate des syscalls.

2 Tampons