

TP Systèmes d'Exploitation : processus et ordonnancement

Responsable

Philippe SWARTVAGHER
philippe.swartvagher@enseirb-matmeca.fr

Intervenante

Fadwa ABAKARIM
fadwa.abakarim@enseirb-matmeca.fr

2025 – 2026

Grandement inspiré du TP de Brice GOGLIN

1 Inspectons les processeurs et les processus

1. Combien a-t-on de processeurs / cœurs ?
Essayez les commandes suivantes :
 - `nproc`
 - `cat /proc/cpuinfo`
 - `lscpu`
 - `top` (saisissez `1` (le chiffre, pas la lettre) pour faire apparaître tous les cœurs, `q` pour quitter)
 - `htop` (`q` pour quitter)
2. Que trouve-t-on dans `/proc` ?
Explorez quelques dossiers et fichiers, puis consultez `man proc`
Expliquez à quoi correspondent en particulier `/proc/<pid>/stat` et `/proc/self`
3. Lancez la commande `top` ou `htop`
À quoi correspondent les différentes colonnes ?
Pour n'afficher que vos processus, vous pouvez ajouter le paramètre `-u $USER` à chaque commande.
Faites apparaître la colonne qui indique sur quel cœur chaque processus est exécuté :
 - `top` : appuyez sur `f`, descendez sur `P`, appuyez sur `Espace`, puis `Echap`
 - `htop` : appuyez sur `F2`, descendez sur `Screen`, avec la flèche de droite allez dans `Available Columns`, descendez sur `Processor`, appuyez sur `Entrée` ou `F5`, puis `F10` pour valider et quitter)
4. À quoi correspondent les trois valeurs `Load average` ?
Indice : la commande `uptime` donne aussi cette information.
5. En parlant de `uptime`, peut-on vraiment se vanter d'avoir un « uptime » important ?

2 Partage du temps processeur entre processus

1. Écrivez un programme en C qui fait une attente active infinie. Le programme n'attendra en réalité rien, il fera une boucle infinie vide.
2. Qu'observez-vous dans `top` (ou `htop`) quand vous lancez ce programme ? Vous pouvez n'afficher que les processus dont la ligne de commande contient `loop` avec `htop -F loop`.
Si vous lancez plusieurs fois ce programme simultanément ?
Comment varie la charge du système ?
Profitez-en pour observer également la hiérarchie des processus (`pstree`, `V` avec `top`, `F5` avec `htop`).
3. Modifiez votre programme pour que si on lui passe un paramètre, il fasse un appel à `printf()` dans la boucle infinie pour afficher quelque chose sur la sortie standard. Qu'observez-vous maintenant si vous lancez le programme pour qu'il affiche quelque chose à chaque tour de boucle ?
Redirigez la sortie standard du programme vers `/dev/null`. Qu'observez-vous ?

Par défaut, un processus peut s'exécuter sur n'importe quel cœur. Pour restreindre sur quel(s) cœur(s) un processus peut s'exécuter, on peut utiliser la commande `taskset` (ou `numactl`, ou `hwloc-bind`) :

```
# programme ne peut s'exécuter que sur les cœurs 0 et 2 :  
taskset -c 0,2 programme  
  
# restreindre le processus avec le PID 7456 aux cœurs 0 et 2 :  
taskset -p -c 0,2 7456
```

4. Lancez plusieurs fois votre programme sur le même cœur.
Comment le cœur est-il partagé entre les processus ?
5. Changez la priorité des processus avec la commande `nice` (ou `renice`)
Quel est l'impact sur le partage du cœur ?
6. Écrivez un programme qui dort le nombre de secondes passé en paramètre du programme (fonction `sleep()`).
Exécutez le programme pour qu'il dorme une dizaine de secondes. Comment le noyau gère les tâches qui dorment ?

3 Temps utilisateur, système et réel

Dans cet exercice, on va utiliser le programme `/usr/bin/time` pour mesurer la durée d'exécution d'un programme. Votre shell peut avoir sa propre implémentation de la commande `time`, c'est pour ça qu'on va préférer utiliser le chemin absolu vers le programme.

1. Utilisez `/usr/bin/time` pour mesurer la durée de la commande

```
curl -o /dev/null -s https://www.bordeaux-inp.fr/
```

Que fait cette commande ? À quoi servent les paramètres `-o /dev/null` et `-s` ?
À quoi correspondent les différentes valeurs affichées ?
Expliquez les valeurs obtenues.

2. Utilisez `/usr/bin/time` pour mesurer la durée de votre programme qui dort pendant plusieurs secondes.
Expliquez les valeurs obtenues.

4 Durée d'un appel système, d'un changement de contexte, d'une timeslice

Pour mesurer une durée en C, on peut utiliser le code suivant :

```
#include <sys/time.h>

unsigned long usec;
struct timeval tv1, tv2;

gettimeofday(&tv1, NULL);
/* ... instructions à mesurer ... */
gettimeofday(&tv2, NULL);

usec = (tv2.tv_sec - tv1.tv_sec) * 1000000 + (tv2.tv_usec - tv1.tv_usec);
```

Note : là où il est indiqué un nombre d'itérations pour faire une moyenne, adaptez le nombre pour que l'exécution ne dure pas trop longtemps, mais pour que ce soit suffisamment long pour lisser le bruit système (une moyenne affichée toutes les secondes et un bon ordre de grandeur).

1. Confirmez que votre code fonctionne correctement en mesurant la durée de la fonction `sleep(2)`.
Est-il possible que cet appel système dure un peu moins de 2 secondes ?
Comment faire pour dormir précisément 2 secondes ?
2. Mesurez la durée de l'appel système `getpid()`.
Mesurez la durée de l'appel système `gettimeofday()`.
Comparez les résultats obtenus en utilisant ce code pour forcer de vrais appels système :

```
#include <unistd.h>
#include <asm/unistd.h>

pid_t pid = syscall(__NR_getpid);
syscall(__NR_gettimeofday, &tv, NULL);
```

Pour obtenir des résultats relativement fiables, faites la moyenne en mesurant la durée de 10 000 000 appels.

Qu'observez-vous ? *Indice* : `man vdso`

3. Écrivez un programme qui exécute dans une boucle infinie : 100 000 appels à `sched_yield()` puis affiche la durée d'un appel.
Lancez votre programme en le restreignant à un unique cœur. À quoi correspondent

les durées affichées ?

Lancez de nouveaux processus qui exécutent ce programme sur le même cœur. Comment évoluent les durées affichées ?

Pour que les performances du système restent correctes, comment doit se comparer la durée du changement de contexte à celle d'une timeslice ?

4. (a) Écrivez un programme qui fait des `gettimeofday()` en boucle et affiche un message si la durée entre deux appels est inhabituellement longue. Le programme prendra un paramètre pour indiquer la valeur du seuil pour définir une durée « inhabituellement longue ». Le message devra contenir la durée suspecte et le PID du processus.

Restreignez l'exécution du programme sur un unique cœur et testez différentes valeurs de seuil. Qu'observez-vous ?

Lancez un deuxième processus sur le même cœur. Comment les valeurs varient ? Pourquoi ?

- (b) Adaptez votre programme pour qu'il affiche maintenant en boucle combien de temps il a été exécuté (donc la durée de sa dernière timeslice) et combien de temps il a été désordonné. On utilisera les variations de durée de `gettimeofday()` pour cela.

Qu'observez-vous lorsque vous lancez un puis deux processus sur le même cœur ?