EX01 – Introduction / Processus

Systèmes d'exploitation / Classes T-2adfg

– Solutions –

1. L'une des raisons pour lesquelles les IHM ont mis longtemps à s'imposer était le coût du matériel nécessaire pour les supporter. Combien de RAM vidéo faut-il pour gérer un terminal texte de 25 lignes et 80 colonnes ? Combien pour un écran graphique de 1024 × 768 pixels, avec un bitmap de couleurs sur 24 bits ? Combien cela coûtait-il en 1980 (prix de la RAM : 5 dollars le KiB[1]) ? Combien cela coûte-t-il maintenant ?

   Solution : Un terminal texte de 80 × 25 nécessite 2'000 octets. Un écran graphique de 1024 × 768 avec 24 bits de couleur par pixel nécessite 2'359'296 octets. En 1980, ces options coûtaient 10 dollars respectivement 11'520 dollars. Aujourd'hui, la RAM coûte entre 10 et 20 dollars par GiB ce qui fait environ 2 cents par MiB. ces options coûtent aujourd'hui 0.004 cents et 4.5 cents.

2. Considérez un ordinateur doté de deux processeurs, chacun disposant de deux threads (hyper-threading). Supposez que trois programmes, *P0*, *P1* et *P2*, sont lancés avec des temps d'exécution de 5, 10 et 20 ms. Combien de temps mettra cet ordinateur pour exécuter globalement ces trois programmes ? Considérez que ces programmes sont 100% en mémoire, qu'il n'y a pas de blocage pendant l'exécution et qu'il n'y a pas de changement de processeur une fois qu'il est assigné.

   Solution : L'ordinateur peut mettre 20, 25, 30 ou 35 ms pour exécuter globalement les programmes selon l'organisation de leur exécution par le système d'exploitation.

   - Si *P0* et *P1* sont lancés sur le même processeur et que *P2* est lancé sur l'autre, il mettra 20 ms.
   - Si *P0* et *P2* sont lancés sur le même processeur et que *P1* est lancé sur l'autre, il mettra 25 ms.
   - Si *P1* et *P2* sont lancés sur le même processeur et que *P0* est lancé sur l'autre, il mettra 30 ms.
   - Si les trois sont lancés sur le même processeur, cela mettra 35 ms.

3. Considérez un ordinateur qui dispose d'un système de cache, d'une mémoire centrale (RAM) et d'un disque. Son système d'exploitation met en œuvre la mémoire virtuelle. Il faut 2 ns pour accéder à un mot dans le cache, 10 ns pour accéder à un mot en mémoire centrale et 10 ms pour accéder à un mot sur le disque. Sachant que le taux de succès du cache est de 95%, et celui de la mémoire centrale de 99%, quel est le temps moyen d'accès à un mot ?

   Solution : Le temps moyen d'accès à un mot est de :

   $0.95 \times 2\,ns$ (le mot est dans le cache)
   $+0.05 \times 0.99 \times 10\,ns$ (le mot est dans la RAM et non dans le cache)
   $+0.05 \times 0.01 \times 10\,000\,000\,ns$ (le mot est uniquement sur le disque)
   $= 5002.395\,ns$
   $= 5.002395\,\mu s$

[1] 1KiB = 1 kibibyte = 1024 byte. A ne pas confondre avec 1KB = 1 kilobyte = 1000 bytes. Plus d'info sur `https://en.wikipedia.org/wiki/Binary_prefix`

4. Quel est le rôle d'un appel système dans un système d'exploitation ?

   Solution : Un appel système permet à un processus utilisateur d'accéder à une ou plusieurs fonctions internes au noyau du système d'exploitation et de les exécuter. Ainsi, un programme peut invoquer un ou plusieurs services du système d'exploitation.

5. L'instruction

```
cpt write (df, buffer, nb_octets);
```

   peut-elle retourner une valeur différente de `nb_octets` dans `cpt` ? Si oui, pourquoi ?

   Solution : Si l'appel échoue, par exemple lorsque `df` est incorrect, elle peut retourner -1. Elle peut également échouer quand le disque est plein et qu'il est impossible d'écrire le nombre d'octets demandés. En cas d'achèvement normal, elle retourne toujours `nb_octets`.

6. Pour un programmeur, un appel système ressemble à n'importe quel appel à une procédure. Est-ce important pour lui de savoir quelles procédures déclenchent effectivement un appel système ? Dans quels cas et pourquoi ?

   Solution : Dans le cadre de la logique du programme, il n'est pas important de savoir si un appel à une procédure de bibliothèque est un appel système. Mais dans les cas où les performances comptent beaucoup, le programme s'exécutera plus rapidement si l'on accomplit la tâche sans appel système. Tout système exige du temps système pour permuter entre le contexte utilisateur et le contexte du noyau. En outre, sur un système multi-utilisateur, le système d'exploitation peut planifier l'exécution d'un autre processus lorsque l'appel système est terminé, ralentissant davantage la progression en temps réel d'un processus appelant.

7. Dans l'exemple sur slide 8 de votre cours «I2 - concepts de bases», la procédure et l'appel système sont tous deux appelés `read`. Est-ce nécessaire ? Sinon, lequel est le plus important ?

   Solution : Les appels système ne possèdent pas réellement de nom, autrement qu'à titre informatif. Lorsque la procédure de bibliothèque `read` déroute le noyau, elle place le numéro de l'appel système dans un registre ou dans la pile. Ce numéro sert à l'indexation dans une table. On n'utilise aucun nom. En revanche, le nom de la procédure de bibliothèque est très important puisqu'il apparaît dans le programme.

8. Quelques questions sur les conversions d'unités :

   (a) Combien de secondes y a-t-il dans une micro-année ?

   Solution : Une micro-année contient $10^{-6} \times 365 \times 24 \times 3600 = 31.536\,s$.

   (b) Les micromètres sont souvent appelés microns. Combien mesure un gigamicron ?

   Solution : $1000\,m$ ou $1\,km$.

   (c) Combien d'octets y a-t-il dans 1 To de mémoire ?

   Solution : Il y a $2^{40}\,octets$, soit $1\,099\,511\,627\,776\,octets$.

   (d) La masse de la terre est de 6000 yottagrammes. Combien cela fait-il en kilogrammes ?

   Solution : $6 \times 10^{24}\,k$.

9. À la figure qui décrit les états des processus, trois états apparaissent. En théorie, avec trois états, on pourrait avoir six transitions, deux en sortie de chaque état. Cependant, on ne voit que quatre transitions. Existe-t-il des circonstances dans lesquelles l'une ou l'autre des transitions manquantes pourraient se produire ?

Solution : La transition entre *bloqué* et *en cours d'exécution* est envisageable. Supposons qu'un processus soit bloqué sur l'E/S et que celle-ci se termine. Si le processeur est inoccupé, le processus peut passer directement de *bloqué* à *en cours d'exécution*. L'autre transition manquante, de *prêt* à *bloqué*, est impossible. Un processus *prêt* ne peut pas faire d'E/S ou quelque action que ce soit susceptible de le bloquer. Seul un processus *en cours d'exécution* peut se bloquer.

10. Sur tous les ordinateurs actuels, au moins une partie des gestionnaires d'interruption est écrite en langage d'assemblage. Pourquoi ?

Solution : Généralement, les langages évolués n'autorisent pas le type d'accès requis au matériel du processeur. Par exemple, il peut être nécessaire de disposer d'un «handler» d'interruption pour activer et désactiver l'interruption servant un périphérique particulier, ou pour manipuler les données dans la zone de pile d'un processus. En outre, les routines de service d'interruption doivent s'exécuter aussi rapidement que possible.

11. Lorsque plusieurs jobs s'exécutent en parallèle, ils peuvent se terminer plus vite que s'ils s'exécutent en série. Supposons que deux jobs qui nécessitent chacun 10 min de temps UC démarrent simultanément. Combien de temps cela va-t-il prendre s'ils s'exécutent séquentiellement ? Et s'ils s'exécutent en parallèle en supposant qu'il y ait 50% de temps d'attente E/S ?

Solution : Si chaque job a 50% de temps d'attente d'E/S, il leur faudra 20 minutes pour se terminer en l'absence de parallélisme. Si les deux jobs s'exécutent séquentiellement, le second se terminera 40 minutes après le commencement de l'exécution du premier. Avec deux jobs, le taux d'utilisation de l'UC est d'environ $1 - 0.5^2$. Donc, chaque job prend $\frac{1-0.5^2}{2}$ soit 0.375 minute de temps UC par minute de temps réel. Pour accumuler 10 minutes de temps UC, un job doit s'exécuter environ $10/0.375$ soit 26.67 minutes. Résumons : en séquentiel il faut 40 minutes pour finir ; en parallèle il n'en faut que 26.67.