Série d'exercice #2

Solution

January 22, 2019

2.1 Système distribué

Expliquer la différence entre le modèle client-serveur et le modèle décentralisé *peer-to-peer* de systèmes distribués.

2.1 Système distribué

Expliquer la différence entre le modèle client-serveur et le modèle décentralisé *peer-to-peer* de systèmes distribués.

Solution

Dans le model P2P il n'y a pas de distinction entre les clients et les servers.

- Toutes les noeuds sont considérés des *peers*.
- Ils peuvent soit agir en tant que client, serveur ou les deux.

Alors que dans le modèle client-serveur on distingue la partie client du serveur.

2.2 Programme vs Processus

Expliquer ce qui distingue un programme (*program*) d'un processus (*process*)?
Indice: Pensez à la relation entre les deux et où ils résident.

2.2 Programme vs Processus

Expliquer ce qui distingue un programme (*program*) d'un processus (*process*)?
Indice: Pensez à la relation entre les deux et où ils résident.

Solution

Un programme est un entité passive (e.g. sur le disque). Un processus est un programme en cours d'exécution. Un programme peut avoir plusieurs processus.

2.3 Ordonnanceur

Donner une différence entre ordonnanceur à court terme et à long terme (*longterm* and *short-term* schedulers).

2.3 Ordonnanceur

Donner une différence entre ordonnanceur à court terme et à long terme (*longterm* and *short-term* schedulers).

Solution

Il y a une différence dans la fréquence d'exécution. Une ordonnanceur à court terme s'exécute beaucoup plus souvent pour beaucoup moins de temps qu'un ordonnanceur à long terme.
Un LTS (*long-term scheduleur*) sert généralement à choisir quelles «Jobs» à charger en mémoire pour l'exécution. Se trouve beaucoup plus dans le contexte de batch processing (traitement par lot). Sert à assurer une balance entre les processus cpu-bound (liés au processeur) et memory-bound (liés à la mémoire).
Un STS (*short-term scheduleur*) choisit plutôt les processus en mémoire (et prêt) auxquels seront alloué du temps de processeur.

2.4 Context switch

Décrire les actions entreprises par le noyau lorsqu'il change d'un processus à un autre (*context switch*).

2.4 Context switch

Décrire les actions entreprises par le noyau lorsqu'il change d'un processus à un autre (*context switch*).

Solution

- Sauvegarder l'état du processus sortant (PCB_0);
- Restaurer l'état du processus entrant (PCB_1);
- . . . plus les caches.

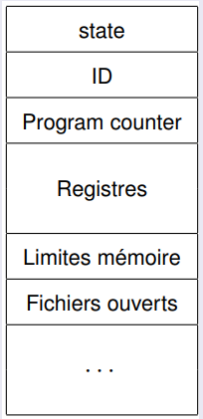

Figure: Information du PCB (note de cours)

2.5 Fork

Quand un processus crée un nouveau processus avec l'opération `fork`, quelles parties, parmis les suivantes, sont-elles partagées entre le processus parent et le processus enfant:

- La pile
- Le tas
- Les segments de mémoire partagés

2.5 Fork

Quand un processus crée un nouveau processus avec l'opération `fork`, quelles parties, parmis les suivantes, sont-elles partagées entre le processus parent et le processus enfant:

- La pile
- Le tas
- Les segments de mémoire partagés

Solution

- La pile (en lecture seul)
- Le tas (en lecture seul)
- Les segments de mémoire partagés.

2.6 Fork^2

Soit le code suivant:

```
#include <stdio.h>
int value = 5;
int main () {
   if (fork () == 0)
      value += 15;
   else {
      wait (NULL);
      printf ("Value = %d\n", value);
   }
   return 0;
}
```

Quelle valeur va-t-il renvoyer à l'écran?

2.6 Fork^2

Soit le code suivant:

```
#include <stdio.h>
int value = 5;
int main () {
   if (fork () == 0)
      value += 15;
   else {
      wait (NULL);
      printf ("Value = %d\n", value);
   }
   return 0;
}
```

Quelle valeur va-t-il renvoyer à l'écran?
La valeur affichée à l'écran est 5.

Fork^3

Soit le code suivant:

```
int main () {
  fork (); fork (); fork (); return 1;
}
```

Incluant le processus initial, combien de processus sont créés par ce programme?

Fork^3

Soit le code suivant:

```
int main () {
  fork (); fork (); fork (); return 1;
}
```

Incluant le processus initial, combien de processus sont créés par ce programme? Il y aura $2 * 2 * 2 = 8$ processus.