Systèmes d'exploitation centralisés

Deuxième Année Informatique et Mathématiques Appliquées
Durée : 1 heure 45 (Les documents distribués en cours et TD sont autorisés)

6 juin 2012

1 Principes généraux (10 points)

1.1 Notion de binaire objet

Un système d'exploitation exécute des programmes représentés dans un format binaire. Ce format, appelé binaire objet, caractérise toute famille de système.

Questions (1 point par question)

1. Quelle est la conséquence (négative) de l'existence de plusieurs formats de binaire objet vis-à-vis de la portabilité des programmes exécutables ? Considérer le cas de systèmes à format binaire différent s'exécutant sur des processeurs matériels identiques.

   Réponse *Chaque famille de systèmes (Unix, Windows, Linux,...) ayant son propre format de fichier binaire, les binaires exécutables ne sont acceptés que par le système d'exploitation correspondant à leur format. Autrement dit, deux binaires exécutables sur un même processeur mais de format distinct ne peuvent être exploitables que si le système d'exploitation leur correspondant est utilisé.*
   De toute façon, les noyaux de systèmes d'exploitation ne sont pas non plus normalisés. Les appels de primitives ne sont pas les mêmes de l'un à l'autre. Les binaires contenant des appels aux primitives du noyau, le même problème se pose vis-à-vis de ces appels. POSIX définit bien une « norme » en la matière, mais elle n'est valable que dans le monde Unix (ce qui n'est déjà pas si mal).

2. Donner les principales sections structurant un binaire objet.

   Réponse *Un format binaire contient un en-tête qui définit les principales composantes qui suivent ainsi qu'une identification du binaire lui-même (appelé parfois magic number). Ensuite, on trouve un ensemble de sections de données initialisées (data) et de code (text) représentant le programme binaire. Enfin, un ensemble de tables essentiellement utiles à l'éditeur de liens, au chargeur et au metteur au point :*
   - *La table de relocation repère les instructions ou pointeurs à translater et est donc utilisée par l'éditeur de liens ;*
   - *La table des symboles contient la définition de tous les symboles, variables et points d'entrée, du programme ;*
   - *La table des identificateurs et chaînes constantes rassemble tous les noms de symboles et toutes les constantes de type chaîne.*

   Ces deux dernières tables sont très utiles pour le metteur au point.

3. On distingue en fait deux types de binaire : le format binaire objet ré-éditable et le format binaire objet exécutable. Pour quelle raison et quelles différences existent entre ces deux formats ?

   Réponse *La raison essentielle est qu'un compilateur ne peut traduire que le programme qui lui est fourni. Or, ce programme contient de nombreuses références externes à des sous-programmes utilitaires (par exemple d'entrée/sortie comme scan,printf,...) ou aux primitives du noyau (exit, fork, open,...). Il*

s'agit donc d'un programme binaire incomplet (il manque des sections de données et de code) qualifié de binaire objet ré-éditable.

Par ailleurs, la programmation modulaire conduit naturellement au développement de modules de programmes, développés et compilés séparément, qu'il faudra ensuite « assembler » pour constituer un programme applicatif.

Le binaire exécutable doit être un programme complet dans lequel toutes les références externes ont été satisfaites : les binaires objets ré-éditables de tous les modules de programmes référencés ont été trouvés et édités ensemble.

4. Préciser les composants logiciels d'un système d'exploitation qui produisent l'un ou l'autre des formats précédents.

   Réponse *Les compilateurs produisent du binaire objet ré-éditables puisqu'ils traduisent un programme source qui contient des références externes à définir ultérieurement.*

   L'éditeur de liens assemble des binaires ré-éditables souvent issues de librairies (archives) pour produire un binaire exécutable. C'est à partir de ce binaire que le chargeur du système d'exploitation pourra implanter une image mémoire exécutable du programme en mémoire virtuelle.

5. Sous quel type de format binaire objet sont représentées les bibliothèques de sous-programmes associés à des langages compilés tels que C ou Ada ?

   Réponse *Jadis, l'édition de liens était statique vis-à-vis des bibliothèques : en clair, le code des bibliothèques était intégré dans le binaire exécutable. Elles étaient donc en format ré-éditables. Les bibliothèques étaient des binaires objets ré-éditables.*

   Aujourd'hui, les éditeurs de liens peuvent traiter des bibliothèques dynamiques. Autrement-dit, une bibliothèque dynamique est un « exécutable » qui peut être partagé par plusieurs processus qui exécutent des programmes référençant les modules contenus dans cette bibliothèque. C'est alors au chargeur (et non pas à l'éditeur de liens) de trouver les bibliothèques dynamiques nécessaires au programme exécutable[1]*.*

   On trouve ainsi les bibliothèques dynamiques dll *sous* Windows*,* so *sous* Unix*,* dylib *sous* Mac OS X*, etc.*

1.2 Confinement des erreurs et robustesse d'un noyau

Un noyau de système d'exploitation exécutant plusieurs programmes à la fois, des mécanismes sont nécessaires pour garantir que les interférences entre programmes via les ressources matérielles ne rendent pas leur exécution incorrecte et sont bien contrôlées.

Questions (1 point par question)

6. Sur quel concept repose la supervision de l'exécution d'un programme dans un système d'exploitation ?

   Réponse *Le concept de processus.*

7. Préciser quelques ressources **matérielles** dont on doit contrôler l'usage par allocation/libération durant l'exécution d'un programme.

   Réponse *L'exécution d'un programme requiert dynamiquement l'allocation des ressources matérielles de l'architecture support. Pour les traitements, il faut évidemment allouer un* ***processeur****. Pour le programme lui-même, de l'espace* ***mémoire****. Au cours de l'exécution d'un programme, des ressources spécifiques peuvent aussi être sollicitées, par exemple, une clé USB, un scanner, etc.*

8. Pourquoi le noyau d'un système d'exploitation doit être protégé vis-à-vis de l'exécution des programmes applicatifs ? Préciser les différentes formes de protection nécessaires.

1. L'éditeur de liens peut intégrer le module qui provoquera le chargement des bibliothèques nécessaires.

Réponse *Le noyau doit être protégé car il est partagé par tous les programmes qui l'utilisent comme un service, comme une machine abstraite support grâce aux primitives qu'il implante et que les programmes appellent.*

Il doit donc être protégé contre les écritures et branchements intempestifs grâce à la mise en place d'un mécanisme de protection mémoire.

Il faut par contre pouvoir appeler les primitives de manière sûre et contrôlée. Pour ce faire, on utilise le mécanisme d'appel superviseur encore appelé déroutement programmé (TRAP).

Enfin, certaines instructions du processeurs doivent être interdites aux programmes applicatifs : par exemple, celles qui permettent de programmer les contrôleurs d'entrées/sorties ou l'unité de gestion mémoire. On introduit donc le mécanisme de mode d'exécution à cette fin. Les programmes applicatifs s'exécutent en mode programme et ne peuvent, dans ce mode, exécuter les instructions dîtes privilégiées. Le noyau s'exécute en mode superviseur et peut exécuter le jeu complet d'instructions du processeur.

9. Un programme peut comporter une instruction inexécutable. Comment traite-t-on ce genre d'événement de façon à assurer le confinement de l'erreur et la poursuite de l'exécution du système et des autres programmes de façon transparente. En particulier, à quel mécanisme matériel fait-on appel ?

   Réponse *Le processeur ne pouvant exécuter l'instruction, le mécanisme de déroutement provoque un branchement immédiat dans une routine du noyau. Le noyau reprend ainsi le contrôle et peut :*
   - *soit arrêter définitivement le programme fautif en détruisant le processus support,*
   - *soit redonner le contrôle à une routine du programme fautif qui pourra récupérer l'erreur au niveau applicatif.*

10. Préciser deux types de ressources **logiques** offertes par les noyaux de système d'exploitation pour que des programmes puissent échanger et/ou enregistrer des données.

    Réponse *Deux ressources logiques essentielles : les fichiers pour les données rémanentes, les pipes pour une communication selon un mode producteur/consommateur.*

2 Ordonnancement des processus(10 points)

On considère le problème d'ordonnancement des processus dans un noyau de système d'exploitation c'est-à-dire les stratégies d'allocation de temps processeur aux processus prêts à s'exécuter. Il peut exister plusieurs processeurs réels (architecture multi-processeurs ou multi-cœurs).

On envisage plusieurs solutions toutes fondées sur l'affectation d'une **priorité** aux processus et sur le partage du temps processeur **par quantum**. Les processus prêts les plus prioritaires reçoivent des quata sans préemption c'est-à-dire que la création d'un nouveau processus prêt de priorité supérieure à celle des processus actifs n'interrompt pas l'un d'eux dans l'exécution de son quantum.

De nouveaux processus prêts sont sporadiquement créés et leur **profil d'exécution** peut évoluer dynamiquement passant de phases **interactives** (entrées/sorties fréquentes) à des phases **calculatoires**.

2.1 Stratégie à quantum de durée unique

Dans cette stratégie, la durée d'un quantum est unique, quelle que soit la priorité attribuée au processus. Chaque processus est créé avec une priorité prise sur un intervalle $[0..PRIO[$: la priorité 0 est maximale et $PRIO - 1$ est minimale. Une file d'attente des processus prêts est associée à chaque priorité.

Lorsqu'un processeur se libère, un processus prêt de priorité pr reçoit un quantum de temps **si et seulement si** aucun processus prêt de priorité supérieure n'existe **et** s'il est en tête de la file d'attente pr des processus prêts à s'exécuter ($\equiv$ il est en tête de la file la plus prioritaire non vide).

En fin de quantum, l'ordonnanceur remet le processus actif en fin de la file d'attente correspondant à sa priorité (stratégie du tourniquet) et il choisit alors le processus prêt le plus prioritaire pour le prochain quantum. Si un processus actif se bloque avant la fin de son quantum, le processus sera de façon similaire réinséré en fin de file d'attente selon sa priorité mais seulement **après son déblocage**.

Questions (2 points par question)

11. Expliquer pourquoi cette stratégie présente un risque de famine en expliquant un scénario précis conduisant à une situation de famine pour certains processus qui ne deviennent jamais actifs.

    Réponse *La famine se produit si un ensemble de files de forte priorité contient toujours au moins un processus prêt. Par exemple, partitionnons les files en deux ensembles de part et d'autre d'une priorité pivot* $p, 0 < p < PRIO$ *:*
    – *un ensemble* $\mathcal{L}_{0..p-1}$ *constitué des files les plus prioritaires de la priorité 0 à la priorité* $p-1$
    – *un ensemble* $\mathcal{L}_{p..PRIO-1}$ *constitué des files strictement moins prioritaires.*
    S'il existe toujours au moins un processus prêt dans l'ensemble $\mathcal{L}_{0..p-1}$*, les processus dans les files de l'ensemble* $\mathcal{L}_{p..PRIO-1}$ *sont en situation de famine. Or, cette situation est tout à fait possible. Il suffit que des processus bloqués ou nouvellement créés reviennent s'insérer régulièrement dans les files de l'ensemble* $\mathcal{L}_{0..p-1}$ *pour en maintenir toujours au moins une non vide.*

12. On propose une variante de cette stratégie : **sur fin de quantum**, le processus actif de priorité pr est réinséré en fin de la file de priorité supérieure $(pr-1)$ par l'ordonnanceur si $pr > 0$. On augmente donc si possible sa priorité au fur et à mesure de son exécution. Quel profil de processus est favorisé par cette stratégie et pourquoi ? Cette stratégie comporte-t-elle encore un risque de famine ?

    Réponse *Cette stratégie est pire que la précédente puisqu'elle favorise les processus de profil calculatoire qui peuvent voir leur priorité augmenter peu à peu. Les processus qui sont donc dans l'état prêt le plus longtemps sont aussi les plus prioritaires. Les processus de profil interactif ont donc très peu de chance de s'exécuter si des processus à profil calculatoire existent de façon continue dans le système. Le risque de famine est donc encore augmenté.*

13. On propose une autre variante : tout processus actif de priorité pr **qui se bloque** est réinséré en fin de la file de priorité supérieure $(pr-1)$ par l'ordonnanceur si $pr > 0$. Quel profil de processus est favorisé par cette stratégie ? Cette stratégie comporte-t-elle encore un risque de famine ?

    Réponse *Cette stratégie favorise les processus interactifs qui peuvent voir leur priorité augmenter peu à peu. Or, ces processus se bloquent souvent et ne sont donc présents dans les files de processus prêts que de façon très passagère. Le risque d'une présence persistante de processus interactifs dans les files les plus prioritaires est donc peu probable. Le risque de famine est fortement réduit même s'il existe encore à cause des processus de profil calculatoire qui peuvent continuer à se coaliser dans les files de forte priorité.*

2.2 Stratégie à quantum de durées distinctes

Dans cette stratégie, on conserve la gestion des priorités mais, à chacune des priorités, est associé un quantum de durée distincte. À la priorité maximale 0 correspond un quantum de durée minimale d et à la priorité minimale $PRIO$ correspond un quantum de durée maximale $PRIO \times d$. La stratégie de l'ordonnanceur consiste à :

– réinsérer un processus actif de priorité pr en fin de file de priorité $(pr+1)$ plus faible **si et seulement si** il consomme son quantum **et** n'est pas encore de priorité minimale $PRIO-1$;
– réinsérer, lorsqu'il redeviendra prêt, un processus de priorité pr en fin de file de priorité $(pr-1)$ plus forte **si et seulement si** il se bloque avant la fin de son quantum et n'est pas déjà de priorité 0.

Questions (2 points par question)

14. Expliquez comment cette stratégie essaie d'adapter de manière optimale la durée du quantum selon l'évolution dynamique du profil d'exécution du processus. ?

    Réponse *Cette stratégie a un double objectif :*
    – *minimiser le nombre de commutations de contextes de processus : garder un quantum trop petit pour les processus de profil calculatoire entraîne des commutation inutiles. Il est donc important d'allonger la durée du quantum alloué à de tels processus. En contre partie, leur priorité doit être diminuée progressivement pour éviter qu'ils ne monopolisent le(s) processeur(s) ;*

– *optimiser l'usage des ressources matérielles pour l'exécution des processus : donner une plus grande priorité aux processus de profil interactif leur donne l'occasion d'accéder plus rapidement aux ressources qu'ils utilisent, disques par exemple, et donc de maintenir un taux d'occupation élevé de ces ressources. L'augmentation de leur priorité est néanmoins sans risque, car comme nous l'avons vu dans la question précédente* (13)*, ceux-ci sont peu présents dans les files de processus prêts.*

Cette stratégie présente donc l'avantage de s'adapter au profil des processus même si ce profil évolue au cours du temps. Un processus peut en effet passer par des phases interactives et calculatoires. Grâce à la stratégie proposée, sa priorité et la durée de son quantum s'adaptera automatiquement à son profil courant.

15. Comporte-t-elle un risque de famine ?

    Réponse *En pratique, le risque de famine est négligeable. Il faudrait le maintien de processus de profil interactif permanent pour occuper les files de forte priorité mais se bloquant peu pour être présents dans ces files le plus longtemps possible : ce qui est, pour le moins, peu probable voire contradictoire.*

    La stratégie proposée est donc bien adaptée à une exécution efficace des processus alliant équité d'accès à la ressource processeur et optimisation du taux d'usage des autres ressources.