M2 Informatique Embarquée

Examens du 27 Mars 2009 et du 3 Avril 2009

Documents autorisés

Questions indépendantes les unes des autres. Regardez le barème avant de vous lancer.

1. Donnez une définition d'un système temps-réel. Qu'est-ce qui différencie un système dit temps-réel d'un système n'ayant pas cette caractéristique? Quelle est la différence entre système temps-réel « mous » et systèmes temps-réels « durs »? Donnez 2 exemples de systèmes temps-réel.
   - Un système temps-réel doit réagir aux stimuli de son environnement dans un délai temporel imposé par l'environnement, l'application. Pour un système temps-réel, il ne suffit pas de fournir des résultats corrects, il faut aussi les fournir en temps et en heure.
     - Résultats corrects au sens temporel
     - Des résultats corrects mais tardifs sont faux.
   - Dures (Hard real-time)
     - Une contrainte est dite "dure", si son non respect résulterait en une catastrophe (Kopetz 1997)
   - Lâches (Soft real-time)
     - Toutes les autres contraintes temporelles
   - Exemples:
     - système de contrôle de procédés industriels,
     - Système de freinage du metro 14.

2. Quelle est la différence entre un noyau Linux et une distribution Linux? Linux est-il adapté aux systèmes embarqués? Détaillez. Utiliseriez-vous Linux pour réaliser un système temps-réel dur? Pourquoi? Qu'est-ce que BusyBox?
   - Système:
     - Système d'exploitation, Noyau
     - Linux: http://kernel.org/
     - Arbre des sources des différentes versions du noyau
     - Fournit juste les services utiles aux applications...
     - Entité plutôt passive
   - Distribution:
     - Ensemble des logiciels nécessaires pour une fonctionnalité complète et du système
     - Inclus: outils d'administrations et applications
       - Mount, fsck, ifconfig, init,...
       - Éditeurs, compilateurs,..
   - Il existe des distributions Linux pour système embarqués avec petit encombrement mémoire et file système allégé (Monta-Vista Linux, µCLinux, DSL,...). Par ailleurs le noyau Linux tourne sur un grand nombre de processeur dont beaucoup adaptés aux besoins des systèmes embarqués.
   - Même si Linux (en particulier le noyau 2.6) a été modifié pour être plus à même de supporter des applications temps-réel, il est encore un peu prématuré de considérer qu'il puisse être utilisé tel quel pour des systèmes TR durs. Avec des extensions comme RTLinux, on peut par contre envisager ce type d'applications.

- BusyBox:
  - Créer une commande unique fournissant les services des commandes usuelles
    - Exemples: ls, cp, sed, diff, mkdir...
  - Gains:
    - Réutilisation de code commun non fourni par des librairies
    - Évite les copies bibliothèques statiques sur disque/mémoire, mais on peut utiliser les librairies dynamiques
    - Fournir des versions réduites de ces commandes
    - Voir aussi FreeBSD crunchgen

3. Décrivez ce que sont une API et une ABI, donnez la signification de ces 2 sigles. Quelle différence y a-t-il entre les deux? Sont-elles dépendantes des processeurs? Que définit la spécifcation »Single Unix Specification v3.0 »? Que définit le standard « Linux Standard Base » (LSB)?
   - API: Application Programming Intreface
     - Définit le prototype des fonctions
       - Ex: int open (const char * name, int oflags,...)
     - Définit les fichiers d'inclusions (header files)
     - Définit le comportement de la fonction
     - Permet la portabilité source des applications
     - Ne permet pas la portabilité binaire des applications => ABI.
   - ABI: Application Binary Interface
     - Permet la portabilité binaire des applications
       - Et donc dépendant du processeur ABI x86, ABI Sparc,...
     - Format de fichiers binaires exécutables
       - ELF, format de debug, description des symboles, librairies,...
     - Conventions invocations de fonctions
       - Passage de paramètres (registre / pile), Retours de fonction
     - Invocation d'appels systèmes
       - Numéro associé, Passage paramètres entrée / sortie
       - Handler de signaux, etc...
   - LSB: Linux Standard Base
     - Résoudre les problèmes d'interopérabilité et de portabilité entre systèmes Linux un ensemble de bibliothèques standards,
       - S'appuie sur des spécifications d'ABI spécifiques aux processeurs
     - un nombre de commandes et d'utilitaires qui étendent le standard POSIX,
     - la structure de la hiérarchie du système de fichiers,
     - les différents run levels,
     - Le « packaging »: rpm (et pas deb)
     - et plusieurs extensions à X Window System.
   - Sus définit une API
     - Application Programming Interface
4. Quelle différence y a-t-il entre un appel système fork et un appel vfork? Pourquoi vfork est-il particulièrement adapté à certains systèmes embarqués? Lesquels?
   - Un fork crée un clone du processus appelant, père et fils ont une copie de l'espace d'adressage.
   - Un vfork crée aussi un clone, mais le fils s'exécute en premier... dans l'espace d'adressage du père...jusqu'à ce que le fils fasse
     - soit un appel à exit auquel cas, le père peut reprendre son son exécution,
     - soit un appel à exec, auquel cas il dispose alors de son propre espace d'adressage et

« libère » le père.
- Vfork est très utile sur les processeurs où l'on ne dispose pas de MMU (ARM7, DSP...)

5. De manière succincte, quelles sont les différences entre un OS, un micro-noyau et un OS temps-réel? Citez un exemple de chaque. Quels sont les grands services rendus par chacun de ces « logiciels »?
   - OS: (Linux, Windows,...)
     - tous services
       - Gestion mémoire
       - gestion processus, ordonnancement,
       - Réseau, IPC
       - gestion fichiers
       - Supports pilotes de périphériques
       - Gestion des utilisateurs
       - Comptabilité (accounting)
     - Implémentation monolithique espace superviseur, sujet aux défaillances de chacun de ses constituants
   - Micro-noyau:
     - Garder un minimum en espace privilégié: le micro-noyau
     - Reporter en espace utilisateur le reste des services, dans des serveurs dédiés
     - Services:
       - Mémoire (sous-ensemble lié aux mécanismes, politiques en mode utilisateur)
       - Ordonnancement (et minimum gestion processus)
       - IPC (communication entre serveurs)
       - I/O: minimum permettant écriture pilote de périph en mode user
   - OS Temps-réel
     - Proche d'un micro-noyau avec des services permettant programmation d'applications plus que de serveurs.
6. Décrivez les 3 manières de parvenir à virtualiser un système (OS)? En quoi diffèrent-elles? Quels sont leurs mérites et complexités respectifs? Qu'a apporté la technologie Intel-VT (AMD SVM) ? Parmi ces 3 manières quelles sont celles qui sont utilisables dans le monde embarqué, temps-réel et celles qui ne le sont pas? Étayez votre proposition.
   - Virtualisation transparente :
     - Traduction binaire dynamique (ex: VmWare)
       - Exécute le système à un niveau de privilège inférieur à celui pour lequel le système a été conçu. Les instructions de l'OS invité sont lues et éventuellement ré-écrites avant exécution pour éliminer les instructions sensibles
     - Assistée par matérielle
       - Suppose un processeur totalement virtualisable: Toutes les instructions sensibles exécutées par l'OS invité déclenchent une exception matérielle qui transfère le contrôle d'exécution à la couche de virtualisation (hyperviseur ou VMM)
   - Paravirtualisation
     - Quand aucune des 2 méthodes ci-dessus n'est utilisable (processeurs embarqués par ex)
     - Quand on dispose des sources de l'OS invité
     - On peut adapter les couches basses de l'OS invité à la couche de virtualisation
     - Permet en général une meilleure synergie entre VMM et OS invité
   - La virtualisation assistée par matérielle et la paravirtualisation sont compatibles avec les besoins temps-réel et embarqués. La traduction dynamique binaire requiert une

empreinte mémoire trop grande et a une influence négative sur le déterminisme.

7. Listez le plus exhaustivement possible les éléments qui interviennent dans le temps d'exécution d'un programme, en expliquant pour chacun d'eux quelle est leur influence.
   - Gestion mémoire :
     - la gestion du swap/pagination peut entrainer un surcoût,
     - Allocation mémoire peut ralentir un programme
     - Le verrouillage d'un processus en mémoire peut accélérer ses performances
     - Les mécanismes de ramasse-miettes sont des surcoûts cachés
   - Vitesse du processeur
   - Taille des caches du processeur
   - Charge du système
     - ordonnancement peut tenir un processus éloigné du processeur
     - L'ordonnancement d'autres processus peut perturber le cache en le vidant des éléments utilisés par le programme dont on cherche à mesurer les performances
   - Capacité/Débit/latence du système d'Entrées/Sorties. De même la concurrence de plusieurs processus peut ralentir un programme
   - Options du compilateur
     - Utilisation de code optimisé
     - utilisation de code PIC tend à ralentir un programme
8. Quels éléments peuvent intervenir sur l'empreinte mémoire d'un programme? Quelle(s) différence(s) y a-t-il entre empreinte sur disque et empreinte en mémoire? Comment réduire l'une, l'autre? Quels effets cela peut-il avoir sur le temps d'exécution de ce programme?
   - Empreinte sur disque:
     - Taille de l'exécutable sur le disque, inclus souvent les tables de symboles et des sections ELF inutiles pendant l'exécution elle-même
   - Empreinte mémoire:
     - Taille du code + taille des données statiques et dynamiques
     - Données dynamiques:
       - liées aux appels de procédures – attention à la récursivité -
       - liées aux allocations de données dans le tas,
     - Taille (code, données statiques et dynamiques) des bibliothèques partagées dynamiques
   - Librairies (cf ci-dessus Q5)
   - Options de compilation (code optimisé en général plus réduit)
   - Réduction sur disque:
     - « stripper » le binaire, utiliser des bibliothèques dynamiques
   - Un programme de taille réduite a plus de chance de tenir dans le cache du processeur et d'aller plus vite.
9. Qu'est-ce qu'une latence d'interruption? En quoi est-ce important? Qu'est-ce que la gigue sur l'arrivée d'un événement périodique? En quoi gigue et latence sont-elles, ou ne sont-elles pas corrélées? Faites un graphique si nécessaire. Quelles caractéristiques doivent avoir latence et gigue dans un système temps-réel? Comment peut-on « garantir » ces caractéristiques? Décrivez brièvement un algorithme permettant de mesurer la gigue d'un événement périodique (min, max, moyenne).
   - Latence:
     - délai entre l'arrivée de l'évènement et sa prise en compte (exécution de la première instruction associée à cet événement)
   - Plus le délai est long, moins le système a de temps pour produire le résultat correspondant dans les délais impartis. Les systèmes temps-réels tendent à avoir des

latences faibles, voire très faibles

- Gigue:
  - Variation sur le délai d'arrivée de l'évènement, ou sur le délai de sa prise en compte,
  - La gigue va (en général) accroître la latence de l'évènement,
  - Un système avec beaucoup de gigue est peu déterministe et donc peu propice à supporter des applications temps-réel.
- Un système temps-réel va minimiser la gigue de prise en compte des interruptions, en réduisant au maximum la durée de ses sections critiques (durant lesquelles les interruptions sont masquées). Idéalement ces sections critiques sont toutes bornées dans le temps et courtes. La latence maximum peut donc être déduite de la durée de la plus longue section critique.