Gestion des Entrées/Sorties (Noyau Linux)

Architecture Matérielle : Bus

- Bus
  - Ensemble de lignes reliant plusieurs composants
    - Tous les composants voient tout ce qui passe
    - Permet la transmission de signaux (bits) en parallel
- Architecture PC : 3 bus
  - Données
    - Pentium = 64 bits (lignes)
  - Adresse
    - Pentium = 32bits
  - Contrôle
    - Verrouillage du bus donnée
    - Sens des transferts (lecture/écriture)
    - ...

Bus d'Entrées/Sorties

- Lignes qui relient CPU <-> périph. E/S
  - Sous-ensemble des 3 bus primaires
- Sur archi. 80x86
  - Adresses : 16 lignes / 32
  - Données : 8, 16 ou 32 /64
- Connexion physique périph/Bus
  - Hiérarchie de composants
    - Ports d'E/S
    - Interfaces
    - Controlleurs de périphériques

Ports d'E/S

- Chaque périphérique a son propre ensemble d'adresses
  - Appelées "Port d'E/S"
    - Jusqu'à 65K ports 8 bits
  - 2 ports 8bits contigus = 1 port 16 bits
  - 2 ports 16 bits contigus = 1 port 32 bits
  - Instructions réservées en assembleur
    - in/out : lecture/écriture 1 octet
    - ins/outs : lecture/écriture d'une séquence d'octets
- Les ports peuvent être mappés en mémoire physique
  - Ne pas confondre avec DMA ...
  - Lecture/écriture à l'aide d'instructions standard
    - mov, and, or, ...

Organisation des Ports d'E/S

- Vue logique : 4 "registres"
  - Contrôle : CPU -> périph.
  - Etat (status) : périph -> CPU
  - Entrée (Input) : données périph -> CPU
  - Sortie :(Output) : données CPU -> périph.
- Implémentation Physique
  - Réduction des coûts = réutilisation des ports
    - Ex: même port pour Entrée et Sortie

API Noyau / Ports

- Fonctions d'entrée
  - inb()/inw()/inl() : lecture 1x8, 1x16 ou 1x32 bits
  - inb_p()/inw_p()/inbl_p() : idem + instruction nulle
  - insb()/insw()/insl() : nx8, nx16 ou nx32 bits
- Fonctions de sortie
  - outb()/outw()/outl()
  - outb_p()/outw_p()/outl_p()
  - outsb()/outsw()/outsl()

La Notion de Ressource d'E/S

- Problème : quels ports pour quels périphériques ?
  - Détection difficile dans certains cas
    - Bus ISA
  - Conflits possibles entre pilotes
    - En croyant s'adresser à un périph. un pilote peut envoyer une commande à un autre...
- Solution : gestion rigoureuse des affectations
  - Ressource = plage affectée **exclusivement** à un pilote
    - Plage de ports (adresses si ports mappés en mémoire)
  - Ressource organisées de façon **hiérarchique**
    - Ex: partage des ressources IDE entre master et slave
      - ressource "master IDE1" fille de ressource IDE1, fille de IDE...

Organisation Hiérachique des Ressources d'E/S

- Structure ressource dans le noyau
  - début/fin plage de ports
  - parent, frères, fils
  - Drapeaux d'état
- Racine : variable **`ioport_resource`**
- API offerte aux pilotes
  - {request|release|check}_resource() :
    - affectation/libération/disponibilité de ressources
  - {request|release|check}_region() :
    - visent plus directement les ports d'E/S
    - Voir `/proc/ioports`

Interfaces d'E/S

- Circuit spécialisé
- Intermédiaire entre port et périphérique
  - Traduction registres -> commandes_periph
  - Détection changements états -> registres
  - Peut aussi être connecté à une **ligne d'interruption**
    - branchée à un PIC/APIC (Contrôleur d'Intr. Progr.)
    - PIC/APIC déclenche interruption matérielle
- Deux types d'Interfaces
  - Dédiée à un matériel
    - Souvent embarquée sur la même carte que le **contrôleur**
  - Généraliste
    - Permettent de connecter des matériels (très) différents
    - Connecteur vers périphérique externe (RS232, ...)

Exemples d'Interfaces d'E/S

- Dédiées
  - Clavier : connectée à un contrôleur qui possède son propre CPU
  - Graphique : vers contrôleur avec CPU, mémoire (frame buffer), et mémoire morte
  - Disque : connexion par câble au disque, qui contient le contrôleur, souvent assez élaboré (mémoire cache, ...)
  - Réseau : Interface et contrôleur sont généralement embarqués dans la "carte réseau"
- Généralistes
  - Parallèle (8bits à la fois), série (1 bit à la fois), USB
  - PCMCIA (portables), SCSI, Firewire, ...

Contrôleurs de Périphériques

- Simplifient l'utilisation de périphériques complexes
  - Interpètent les commandes (simples) de l'interface
    - Supervisent la réalisation des actions correspondantes
    - Exemple: disque
      - Traduire "écrire bloc X" en :
        - Placer les têtes sur le cylindre C (attendre)
        - Attendre rotation pour être au-dessus de secteur S1
        - Ecrire données (jusqu'à secteur S2)
        - Mettre à jour cache interne pour futures lectures
        - Signaler la fin de l'opération
  - Convertissent les signaux électriques émis par périphérique en valeurs pour registre d'état

Zones Mémoire d'E/S Partagées

- Trois zones de partage sur architecture PC
  - Bus ISA : 0xA000 à 0xFFFFF
    - "Trou" de 640 Ko à 1 Mo
      - Complique init table des pages
  - Bus VLB (VESA) : 0xE00000 à 0xFFFFFF
    - Adresses de 14Mo à 16 Mo
      - Complique encore init table des pages
  - Bus PCI
    - Adresses physiques très élevées
      - Moins difficile à gérer car @ supérieures à la quantité de mémoire physique installée
    - Extension AGP
      - Capable d'utiliser directement la mémoire centrale du CPU
        - Grâce au circuit GART (Graphic Address Remapping Table)

Accès aux Zones de Mémoire d'E/S partagées

- Rappel : le noyau utilise adresses linéaires du 4e Go
  - Début = PAGE_OFFSET (0xC0000000)
  - Cas simple : E/S partagent des adresses physiques basses
    - il suffit d'un OU binaire avec la valeur PAGE_OFFSET
      - Premier octet @lineaire noyau = premier octet mem physique
  - Problème avec les adresses physiques hautes
    - Les @linéaires ne sont pas en vis-à-vis des @phys.
      - Limite des 896 Mo
      - Donc Pas d'association permanente @phys / @linéaire ...
    - Solution : ioremap() / ioremap_nocache()
      - Construit un mapping entre @linéaires et @physiques E/S

API Noyau pour Manipuler Adresses E/S Partagées

- Mémoire partagée mais ...
  - ... utilisation (potentiellement) différente
- Sur certaines architecture, accès par inst. spéciales
  - Utilisation **systématique** de fonctions génériques d'E/S
    - Pour que le code des pilotes soit indépendant de l'architecture cible
      - Et donc portable
  - API noyau Linux
    - readb, readw, readl, writeb, writew, writel
    - memcpy_fromio, memcpy_toio, memset_io

Accès Mémoire Direct (DMA)

- Tous les PC ont un processeur DMAC
  - Direct Memory Access Controller
- Permet le transfert **asynchrone** entre RAM et périph.
  - Le pilote construit une réquête décrivant le transfert
  - Le pilote demande au CPU de soumettre la requête au DMAC
    - LE DMAC s'occupe du transfert sans bloquer le CPU
  - Le pilote place le processus client dans l'état bloqué
    - Le scheduler élit un autre processus
  - Quand le transfert est terminé, le DMAC déclenche une intr. matérielle
  - Le traitant réveille le pilote, qui réveille le proc. client

Espace d'Adressage DMA

- 4e espace d'adressage
  - En plus de logique, linéaire et physique ...
  - Appelée "Adresses de Bus"
  - Adresses manipulées par tous les périphériques
    - **Sauf** le CPU
    - Permettent de désigner les données à placer sur le bus de données
    - Sur architecture PC : identiques @physiques du CPU
- Adresses de bus incontournables avec DMA
  - Communication directe entre DMAC et périphérique
  - Le noyau (cad le CPU) doit parler leur "langue"
    - Conversion avec virt_to_bus()/bus_to_virt()

Mies en Oeuvre des Accès DMA

- Accès au périphérique via fichier spécial
  - Allocation d'une ligne IRQ lors de la première utilisation
    - Surcharge de la méthode open() du fichier
      - Incrémentation d'un compteur de référence
  - Utilisation d'un port pour programmer le DMAC
    - Adresse et taille du tampon DMA
    - Direction du transfert
  - Le DMAC déclenche une interruption en fin d'opération
    - Réveil du processus
  - Libération de la ligne IRQ
    - Quand le dernier processus ferme le fichier spécial
      - Surcharge de la méthode release()

Niveaux de Support des Périphériques par le Noyau

- Trois niveaux de support possibles
  - Aucun (!)
    - Le processus client utilise directement les ports d'E/S
      - Appels systèmes iopl() et ioperm() + assembleur (in/out)
      - Nécessite privilège (root)
      - Exemple : certaines implémentations du serveur X11
  - Minimal
    - Le noyau connait l'interface d'E/S, mais ne sait pas ce qui s'y trouve
      - Le noyau permet aux processus clients d'envoyer des séquences de caractères vers cette interface (via fichier spécial)
  - Etendu
    - Le noyau sait exactement comment utiliser l'interface du périphérique : il se charge de tout

Exemples de Supports Possibles

- Cartes graphiques : plusieurs possibilités
  - Aucun Support : serveur X11 spécialisé
  - Support total : serveur X11 générique "frame buffer"
    - Le noyau fournit un service "frame buffer" générique
      - Fichier /dev/fb
    - Depuis 2.4, support extension DRI
      - Direct Rendering Interface : extensions 3D de certaines cartes
- Interfaces généralistes (série, parallèle, USB)
  - Série, parallèle : Support minimal
    - Le noyau ne sait pas ce qu'on branche sur l'interface
  - SCSI, USB, PCMCIA : Support total
    - Indispensable car ces périphériques ont besoin d'accéder directement au bus de données (ex: transfert disque)

Stratégies de Gestion des Tampons

- Problème de synchronisation
  - D'un coté les processus utilisateurs
    - Couche haute du pilote (Top Half)
    - Peuvent mettre du temps à réagir à un évènement
      - Latence d'aiguillage
  - De l'autre les périphériques
    - Couche basse du pilote (Bottom Half)
    - Réagit très rapidement aux évènements matériels
      - Interruption matérielle déclenche un traitant spécialisé
  - Entre les deux ??
    - Il faut stocker provisoirement des données
      - Exemple : paquets réseau, caractères d'une liaison série, ...
    - Solution classique : algo. producteur/consommateur
      - tampons circulaires + synchro. par sémaphores ...

Effets de l'Utilisation de Tampons

- Deux situations
  - Périph. à accès séquentiels (cartes son, réseau, ...)
    - Une appli (un processus) ne peut pas redemander la même donnée
    - Les tampons permettent d'adoucir les pics de charge
      - Quand le CPU est très chargé, le dialogue entre Top et Bottom est moins rapide
      - Les tampons permettent d'amortir le délai de réaction
  - Périph. à accès direct (disques essentiellement)
    - Un ou plusieurs processus peuvent réclamer la même donnée
    - Les tampons ont un double rôle (plus complexe)
      - Amortir la latence d'aiguillage (**un peu**, car accès souvent lent)
      - Mémoire cache (**beaucoup**, car accès souvent lent)

Techniques de Surveillance des Opérations d'E/S

- Une opération d'E/S a été lancée ...
  - Comment sait-on qu'elle est terminée ?
    - Ou a échoué si on a fixé un délai max (timeout) ...
- Deux techniques
  - Sondage (polling)
    - Le pilote consulte sans cesse (régulièrement) l'état du périphérique
      - Stratégie 1 : vérification à chaque tick d'horloge (int. horloge)
      - Stratégie 2: vérifier + schedule (attente active)
  - Interruption
    - Exige un minimum du périphérique (son contrôleur)
      - Ligne d'interruption matérielle (IRQ)
    - Utile avec périph. aléatoirement lent

Architecture Type d'un Pilote Utilisant les Interruptions

- Le pilote comporte deux fonctions
  - Une pour la partie haute du pilote
    - Généralement exécution en exclusion mutuelle
      - protection par semaphore/mutex
    - Programme la requête d'E/S
      - Données/commande écrite en mémoire
      - Transmission commande au pilote (instruction du type outb())
      - Attente sur wait_queue (ex: wait_event_interruptible())
      - Au réveil : copie du résulat depuis noyau vers espace utilisateur
  - Une pour la partie basse
    - Déclenchée par interruption matérielle
      - Programmation du vecteur d'interruption
    - Lors de l'interruption :
      - Lecture état pilote + réveil partie haute (wake_up_interruptible)

Exemples de Pilote ?

- Les sources du noyau !!
- http://perso.ens-lyon.fr/brice.goglin/progsyst/
  - TP6 + correction : "E/S : IRQ et commande du HP"
    - Utilisation des IRQ
    - Thread noyau
    - Completions, spinlocks
    - ...