Les aspects techniques du projet système

Yves Denneulin, Jacques Mossière, Grégory Mounié,
Simon Nieuviarts, Franck Rousseau, Sébastien Viardot

2007-2008

Résumé

Ce document est complémentaire des transparents présentant les difficultés techniques de programmation d'un système d'exploitation sur PC. Il présente successivement la segmentation, les niveaux de privilège, les interruptions, la programmation de l'horloge et enfin la programmation de la console de l'utilisateur.

1 Introduction

On rassemble dans cette note tous les éléments de l'architecture des PC qui sont utiles (et seulement ceux-ci) pour la réalisation du projet système. Autrement dit, cette documentation est très incomplète. En cas de doute, c'est toujours la documentation des constructeurs qu'il faut utiliser, et tout particulièrement la documentation Intel.

Certaines des caractéristiques présentées sont fixées dans l'architecture IA 32 des processeurs utilisés, d'autres notions sont plus globales comme la connexion et la programmation des périphériques, ou encore l'utilisation d'horloges externes pour décompter des intervalles de temps.

Cette note est organisée comme suit : le paragraphe 2 contient tout ce qu'il faut savoir sur les segments pour les utiliser dans le projet ; le paragraphe 3 décrit les niveaux de privilège et leur utilisation dans le projet ; le paragraphe 4 décrit les interruptions ; le paragraphe 5 la programmation de l'horloge ; enfin le paragraphe 6 est consacré aux entrées-sorties sur la console de l'utilisateur.

2 Segments et adressage

2.1 Généralités

Toutes les informations accessibles se trouvent dans un espace d'adressage plat dont la taille est de 4 Goctets, soit 32 bits d'adresse. Tout élément de cet espace est désigné par une adresse appelée **adresse linéaire**. Pour faciliter la programmation, cet espace plat peut être structuré en un ensemble d'entités logiques de taille quelconque, les **segments**. Chaque segment occupe des emplacements consécutifs dans l'espace plat. Un octet dans un segment est désigné par une **adresse segmentée** composée d'un couple (désignation d'un segment, déplacement dans le segment).

Nous donnons maintenant quelques éléments sur la façon dont le processeur interprète une adresse segmentée pour construire une adresse linéaire. Le paragraphe suivant précise comment les mécanismes de segmentation sont utilisés dans le projet.

Remarque. Une adresse linéaire peut être utilisée sans transformation pour accéder à la mémoire physique ou, ce qui est le plus fréquent, elle peut être traduite en une adresse physique par des mécanismes de pagination.

2.1.1 Segments et descripteurs

Toute information utilisable est logiquement contenue dans un segment, ensemble d'octets contigus. Un segment est accessible par l'intermédiaire d'un descripteur de segments (Figure 1) contenant trois champs principaux : son adresse de base dans l'espace d'adressage linéaire, sa taille et des droits d'accès.

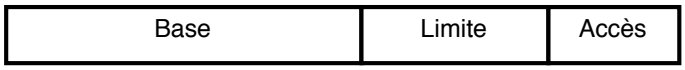

FIG. 1 – Format d'un descripteur de segment

Les descripteurs de segments sont regroupés dans des tables de descripteurs ; l'une de ces tables, la table globale des descripteurs (GDT) est toujours accessible et permet de désigner, outre des segments de code ou de données, d'autres tables de descripteurs. À un instant donné, une seule de ces tables secondaires, la table locale des descripteurs (LDT) est accessible.

Deux registres invisibles au programmeur, GDTR et LDTR, pointent respectivement vers la GDT et la LDT.

Une adresse segmentée se compose de deux parties, la sélection d'un descripteur de segment et un déplacement dans le segment sélectionné. En pratique, la sélection d'un descripteur se fait par l'intermédiaire d'un sélecteur contenu dans un registre de segment.

2.1.2 Registres de segments et sélecteurs

Six registres de segments sont disponibles :
- CS (code segment) désigne le segment contenant le programme en cours d'exécution
- DS (data segment) désigne le segment des données courantes
- SS (stack segment) désigne un segment de pile
- ES, FS et GS permettent d'adresser trois segments supplémentaires.

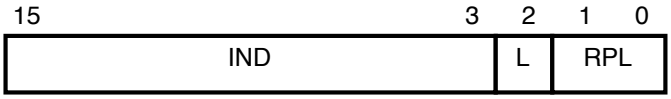

FIG. 2 – Format d'un registre de segment

Un registre de segment (figure 2) contient, sur 16 bits, un sélecteur composé de trois champs :
- L précisant si le descripteur est à prendre dans la GDT ou la LDT
- IND indice du descripteur dans la GDT ou la LDT

- RPL, « requested privilege level », utilisé pour le contrôle d'accès

2.1.3 Transformation d'une adresse segmentée en adresse linéaire

Soit l'adresse segmentée RS,dep où RS est un registre de segment et dep un déplacement.

RS contient un sélecteur dont les champs IND et L permettent d'atteindre un descripteur du segment adressé. Soit B et L les valeurs des champs Base et Limite de ce descripteur de segment. Si $dep > L$, alors il y a une erreur d'adressage. Sinon, l'adresse linéaire correspondant à l'adresse segmentée est $B + dep$.

2.2 Utilisation dans le projet

Avant l'exécution de votre noyau, un programme d'initialisation (crt0) s'exécute. Il a entre autres pour rôle de préparer des descripteurs de segments. Vous n'aurez donc pas à vous préoccuper des tables de descripteurs de segments.

Pratiquement, quatre segments sont définis :
- segment de code du système, indice 2 dans la GDT,
- segment de données du système, indice 3 dans la GDT,
- segment de code de l'utilisateur, indice 8 dans la GDT,
- segment de données de l'utilisateur, indice 9 dans la GDT.

Chacun de ces quatre segments recouvre l'ensemble de l'espace plat (adresses de 0 à $4Go-1$). Un segment de données contient aussi bien les données proprement dites que la pile.

Nous reviendrons à la section suivante sur les sélecteurs à placer dans les registres de segment en fonction du niveau de protection.

Remarque.

Le fait que les segments couvrent tout l'espace linéaire permet d'ignorer la segmentation dans la programmation. c'est un choix imposé par le compilateur : un même déplacement appliqué à SS et DS doit désigner le même octet. Ainsi, dans nos programmes, nous pouvons considérer qu'une adresse est complètement définie par la seule valeur du déplacement. C'est ce qui a été enseigné dans les cours de logiciel de base de première année.

3 Protection

3.1 Les niveaux de privilège de l'IA 32

Le système de protection de l'IA 32, inspiré du système Multics, généralise les modes maître et esclave pour permettre à un programme de s'exécuter à un niveau de privilège parmi quatre possibles. Le niveau 0, ou superviseur est le plus privilégié : un programme doit s'exécuter au niveau de privilège 0 pour faire des entrées-sorties ou manipuler les interruptions. Il existe trois autres niveaux, numérotés de 1 à 3, permettant théoriquement de concevoir des applications faisant intervenir des sous-systèmes à diverses possibilités.

Le niveau courant de privilège est celui qui figure dans le sélecteur CS du segment de programme en cours d'exécution. Un niveau de privilège est attaché à chaque segment. L'idée est de n'accéder à un segment que si le niveau de privilège du programme en cours est inférieur ou égal à celui du segment auquel il souhaite accéder.

L'IA 32 utilise une pile pour la gestion des procédures et les passages de paramètres. Pour garantir un accès valide aux données lorsqu'un programme change de niveau de privilège, il existe en fait quatre piles, une par niveau de privilège.

Les règles de changement de niveau de protection, inspirées du système Multics, sont riches et complexes, si complexes qu'aucun système d'exploitation ne les utilise à plein. Nous allons donc faire comme tout le monde et n'utiliser qu'un petit sous-ensemble des possibilités offertes.

3.2 Utilisation dans le projet

Nous utilisons uniquement les niveaux 3 et 0 : le niveau 3 est le niveau d'exécution d'un processus utilisateur, le niveau 0 est réservé au superviseur. Un processus en mode utilisateur pourra appeler un programme en mode superviseur en effectuant une interruption programmée (instruction int $49). Le retour du mode superviseur au mode utilisateur est fait par l'exécution d'une instruction iret de retour d'interruption.

Nous avons vu plus haut (2.1.2) qu'un segment était sélectionné par un registre de segment. On trouve dans ces registres trois champs

- le champ RPL(bits 0-1) contient le niveau de privilège ;on y placera 0 pour les segments à accéder en mode superviseur et 3 pour les programmes utilisateurs,
- le champ L sera toujours à 0 car on n'utilise pas de table locale de descripteur,
- le champ IND contiendra l'indice dans la GDT d'un des descripteurs préparés à l'initialisation.

On utilisera en définitive les contenus suivants pour les registres de segment :

- en mode superviseur : CS = 0x10, autres 0x18
- en mode utilisateur : CS = 0x43, autres 0x4b

Votre système devra assurer que les registres de segment sont bien chargés avec les valeurs qui correspondent au mode d'utilisation courant.

4 Interruptions

4.1 Principe de fonctionnement

Le mécanisme d'interruption réalise de façon indivisible la sauvegarde du contexte d'exécution du programme en cours pour aller exécuter un programme associé à l'interruption qui est appelé le traitant de l'interruption. On distingue différentes causes d'interruptions :

- des causes externes au programme en cours, provenant en particulier de composants matériels distincts du processeur (organe d'entrée-sortie, horloge),
- des causes internes au programme (erreurs à l'exécution ou interruption programmée).

L'IA 32 comporte 256 numéros d'interruptions. La description des traitants figure dans une table appelée IDT (« interrupt description table ») placée à l'initialisation à l'adresse $4K$.

Nous utilisons les numéros de 32 à 47 pour les interruptions matérielles et le numéro 49 pour les appels au superviseur. Chaque entrée de l'IDT occupe 8 octets et a le format défini sur la figure 3.

Une telle entrée spécifie l'adresse d'un traitant par le couple (sélecteur de segment, adresse de traitant) ; elle précise en outre que l'exécution du traitant se fait interruptions masquées (champ 01110).

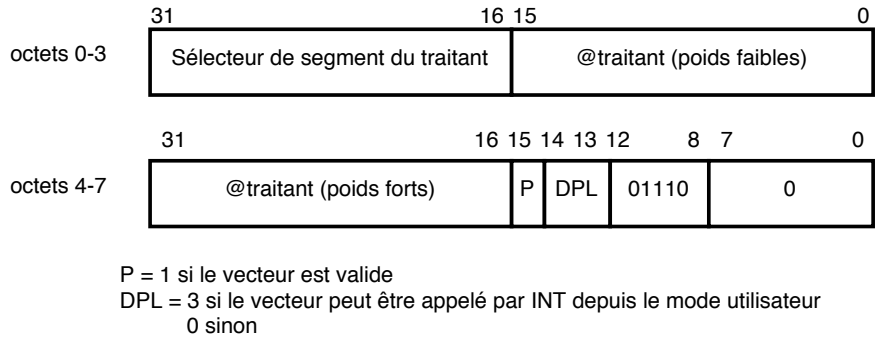

FIG. 3 – Format d'une entrée de l'IDT

La sauvegarde du contexte d'exécution se fait dans la pile correspondant au niveau de privilège du traitant. Le mécanisme d'interruption y sauvegarde :

- le pointeur de pile (SS,ESP) du programme interrompu dans le cas ou l'interruption s'accompagne d'un changement de niveau de privilège,

et dans tous les cas,

- le registre EFLAGS,
- le compteur ordinal (CS,EIP) du programme interrompu.

Pour trouver l'adresse de la pile concernée, le matériel utilise une table globale appelée la TSS (« Task State Segment ») qui, dans notre cas, est placée à l'adresse$0x20000$. Le sommet de la pile superviseur y est défini par les entrées de déplacement 4 (sommet de pile dans le segment) et 8 (sélecteur du segment de pile).

Dans notre projet où le traitant s'exécute systématiquement en mode superviseur, il faut à chaque changement de contexte donnant la main à un nouveau processus placer dans la TSS l'adresse du sommet de la pile système de ce processus.

Ceci revient à placer en $0x20004$ l'adresse du sommet de la pile système du processus ; en $0x20008$ se trouve le sélecteur du segment pile du superviseur qui pour tous les processus contient la valeur $0x18$.

Quelles que soient les sauvegardes effectuées, le retour au programme interrompu est fait par l'instruction `iret`.

Remarque 1. Pour certaines exceptions du processeur, un code d'erreur est également empilé.

Remarque 2. Un effet de bord de l'instruction `iret` peut être le passage de l'exécution du mode superviseur au mode utilisateur. On l'utilisera en particulier pour le démarrage de programmes en mode esclave.

4.2 Interruption programmée

Nous utilisons pour passer en mode superviseur l'instruction

```
int $49
```

qui déclenche l'interruption de numéro 49.

Cette instruction sera exécutée pour chaque appel d'un processus utilisateur à une fonction implantée dans le noyau et devant s'exécuter en mode superviseur, c'est-à-dire pour chaque

fonction définie dans l'API du système à réaliser.

Il nous faut donc écrire un traitant de l'interruption 49 dont le rôle est d'identifier la primitive système à exécuter, de récupérer les paramètres de cette procédure, puis de l'appeler. Nous suggérons de transmettre les différents paramètres par l'intermédiaire des registres ; en particulier, `eax` contiendra le numéro de la procédure à exécuter.

Le traitant doit bien sûr assurer les sauvegardes des registres de l'appelant, en particulier des registres de segments et mettre en place les registres de segments nécessaires en mode superviseur. À la fin du traitant, on doit restaurer l'environnement du processus appelant et y revenir en mode utilisateur par `iret`.

4.3 Interruption externe

Il existe 15 sources d'interruptions externes (IRQ pour interrupt request). Ces sources sont dirigées à l'initialisation sur les entrées 32 - 33 et 35 à 47 de l'IDT. Le traitement d'une interruption externe est très proche du traitement d'une interruption à cause interne. Il y a cependant quelques spécificités à prendre en compte : le masquage, l'autorisation et l'acquittement.

4.3.1 Masquage des interruptions externes

Le traitement d'une interruption externe peut être retardé pour éviter les conflits d'accès aux tables centrales du système : une bonne règle est de faire une chose à la fois et de la terminer avant de passer à autre chose.

Nous vous conseillons de faire exécuter tout votre noyau en mode interruptions masquées, sauf dans le cas d'un système oisif dont le fonctionnement reprendra à la prochaine interruption : il faut donc, dans ce cas uniquement, permettre à une interruption de passer en démasquant les interruptions (instruction `sti`).

Quand vous créez un processus utilisateur, il faut prendre garde à ce que ce processus s'exécute interruptions démasquées : le bit `I` du registre `EFLAGS` doit rester à 1 (valeur initiale de `EFLAGS` : $0x202$).

4.3.2 Autorisation d'une interruption externe

Chaque interruption externe peut être individuellement autorisée ou interdite. Au départ, l'initialisation interdit toutes les interruptions externes.

Il va donc falloir autoriser les IRQ pour chaque composant à utiliser, horloge et clavier dans notre cas. La gestion des 8 premières IRQ se fait par le port d'entrée sortie $0x21$ (cf. section 6). L'écriture d'un octet sur ce port autorise les IRQ dont les bits correspondants sont à 0 ; la lecture sur ce port retourne l'état courant des autorisations, chaque 1 correspondant à une IRQ interdite.

Exemple : pour autoriser l'horloge sans rien changer aux autres autorisations, on peut exécuter

```
outb(inb(0x21) & 0xfe, 0x21)
```

Ces instructions commencent par lire l'état courant des autorisations, forcent à 0 le bit correspondant à l'horloge, puis réécrivent l'octet modifié sur le port $0x21$.

4.3.3 Acquittement d'une interruption

Il faut impérativement indiquer au gestionnaire matériel des interruptions (PIC pour Programmable Interrupt Controller) qu'une interruption vient d'être prise en compte. Cette opération s'appelle l'acquittement d'une interruption. En l'absence d'acquittement, aucune nouvelle interruption ne sera envoyée.

Concrètement, pour les 8 premières IRQ, l'acquittement est fait sur le port $0x20$ sur lequel il faut envoyer la commande $0x20$ par l'instruction

```
outb(0x20,0x20)
```

5 L'horloge

Nous utilisons le contrôleur i8253 qui permet d'envoyer périodiquement une interruption à l'unité centrale. Cette interruption est utilisée pour gérer une heure courante, réveiller les processus endormis et pour changer de processus élu.

La fréquence de base du contrôleur est de $0x1234DD$ Hz. Il est possible de réduire la cadence des interruptions en divisant cette fréquence de base par la fréquence souhaitée FREQ.

La programmation de l'horloge demande d'abord d'envoyer le code $0x34$ sur le port d'entrée sortie $0x43$. Il faut ensuite envoyer sur le port d'entrée-sortie $0x40$ l'octet de poids faible de réglage de la fréquence d'interruption, puis l'octet de poids fort de cette valeur. Soit en définitive :

```
reglage = 0x1234DD / FREQ ;
outb(0x34,0x43) ;
outb(reglage % 256, 0x40 ) ;
outb (reglage / 256, 0x40) ;
```

L'IRQ associée à l'horloge doit être autorisée et les interruptions démasquées pour que le processeur soit effectivement interrompu. Dans notre système, l'IRQ associée à l'horloge est l'IRQ 0 qui est redirigée sur l'entrée 32 de l'IDT.

Remarque. La valeur de réglage de la fréquence est limitée à deux octets. Certaines fréquences trop basses ne peuvent donc pas être demandées.

6 Entrées sorties

6.1 Généralités

Les entrées-sorties se font par l'intermédiaire de canaux bidirectionnels de communication appelés des ports d'entrée-sortie ou, par souci de concision, des ports. Chaque port est désigné par une adresse dans un espace d'entrée-sortie séparé de l'espace d'adresse de la mémoire. Les adresses d'entrée-sortie sont comprises entre $0x0$ et $0xffff$ (espace de $64Ko$).

On accède à un port d'entrée sortie en lecture par l'instruction `in` et en écriture par l'instruction `out`. On choisit de n'autoriser ces instructions qu'en mode superviseur en réglant le niveau de privilège des entrées-sorties (IOPL) à 0 (ce sont deux bits du registre EFLAGS).

6.2 Programmation de l'écran

6.2.1 Affichage de caractères

Nous nous limitons dans le projet à un affichage en mode texte VGA dans lequel l'écran correspond à 25 lignes de 80 caractères chacune.

À cette matrice de caractères correspond une mémoire vidéo, c'est-à-dire une zone de mémoire qui est balayée régulièrement par l'écran pour afficher effectivement les caractères inscrits dans la mémoire vidéo.

Remarque. Plus précisément, la mémoire vidéo se trouve dans la carte vidéo et mise en correspondance avec des adresses de mémoire centrale. En pratique, tout se passe comme si la mémoire vidéo se trouvait en mémoire centrale.

La mémoire vidéo est une matrice 80 x 25 de mots de 16 bits dans chacun desquels les huit bits de poids faibles correspondent au code d'un caractère à afficher, les huit bits de poids forts à la couleur du fond, à la couleur du texte, avec un bit de clignotement. La mémoire vidéo commence à l'adresse $0xB8000$ et est organisée ligne par ligne.

Remarque. L'affichage sur l'écran ne demande aucune instruction d'entrée sortie, mais de simples écritures dans la mémoire vidéo.

6.2.2 Gestion du curseur

La programmation détaillée de l'écran fait appel cette fois ci à des instructions d'entrée-sortie ; des registres du contrôleur peuvent être lus ou écrits via des ports d'entrée sortie.

Plus précisément, le port $0x3d4$ permet de sélectionner un registre de l'écran et le port $0x3d5$ est utilisé pour lire ou écrire dans le registre sélectionné. En ce qui concerne le curseur, deux registres $0x0e$ et $0x0f$ gèrent sa position. Le programme ci-dessous permet de placer le curseur à la position (ligne `l`, colonne `c`).

```
column = c;
line = l;
c = c + l * width;
outb(0x0F, 0x3d4);      // sélection du registre
outb((char)c, 0x3d5);   // écriture
outb(0x0E, 0x3d4);
outb((char)(c >> 8), 0x3d5);
```

6.3 Entrée de caractères au clavier

6.3.1 Principe de fonctionnement

Nous décrivons ici le minimum à connaître sur le fonctionnement du clavier et sa programmation. Une documentation plus complète est accessible via la page web du projet.

Le clavier du PC fonctionne de façon très proche de ce que nous avons étudié en cours : à chaque enfoncement ou relâchement d'une touche, une interruption est envoyée à l'unité centrale. En réponse à cette interruption, le traitant peut lire, sur un port d'entrée-sortie associé au

clavier, un code (appelé scancode) précisant la touche concernée et si la touche a été enfoncée ou relâchée. On dit que la programmation du clavier est pilotée par interruption (« interrupt-driven »).

Remarque. Il y a en général deux interruptions pour la frappe d'une touche, l'une à l'appui sur la touche, l'autre au relâchement de la touche.

La transformation d'un scancode en code de caractères est une opération complexe et pénible : elle dépend de l'état d'autres touches du clavier (Shift, Control, Alt, etc.), ainsi que d'événements passés (la touche « Caps Lock » a-t-elle été pressée un nombre pair ou impair de fois ?). L'arrivée d'un nouveau scancode peut correspondre à un nouveau caractère utile, mais de temps en temps à 2 et souvent à 0. Nous vous fournissons des outils vous permettant d'ignorer cette complexité. Nous les présentons au paragraphe suivant.

6.3.2 La lecture de caractères dans le projet

Le contrôleur du clavier i8042

Le contrôleur du clavier possède quatre registres de lectures ou d'écritures, chaque registre ayant une taille d'un octet. Le seul dont nous avons besoin est accessible sur le port d'entrée-sortie $0x60$. En pratique, à chaque interruption clavier, on récupère le scancode correspondant par une instruction du genre

```
inb $0x60,%al
```

L'interruption clavier

La structure d'ensemble de la programmation de l'interruption clavier est identique à ce qui a été réalisé pour l'interruption horloge. Cette interruption doit être autorisée à l'initialisation du système. Le traitant de l'interruption doit l'acquitter pour qu'on puisse recevoir une nouvelle interruption venant du clavier. Dans notre système, le clavier correspond à l'IRQ 1 qui est redirigée sur l'entrée 33 de l'IDT.

Le décodage des scancodes

Pour vous éviter le casse-tête du décodage des scancodes, des aides vous sont fournies dans le répertoire kbd_linux et les fichiers keyboard_glue.c et keyboard_glue.h. Vous n'avez rien à modifier à ces fichiers.

Le fichier kbd.h contient la déclaration d'une fonction `void do_scancode(int scancode)` définie dans keyboard_glue.c que le traitant d'interruption doit appeler en lui fournissant le scancode obtenu. Lorsque cette fonction détecte que un ou plusieurs caractères utiles sont devenus disponibles, elle fait appel à la fonction `void keyboard_data(char *str)` en lui fournissant dans la chaîne `str` le ou les nouveaux caractères disponibles. Vous avez à implanter cette fonction de façon à ce qu'elle recopie les nouveaux caractères disponibles dans le tampon associé au clavier. Lorsque le tampon du clavier est plein, ces caractères sont ignorés. Un « bip » peut éventuellement signaler cet événement à l'utilisateur.

Écho des caractères frappés

Il est commode au fur et à mesure de la frappe de caractères de les afficher sur l'écran : on dit qu'on fait l'écho des caractères entrés. En plus de la mise dans le tampon du clavier, votre pilote de clavier doit assurer l'écho des caractères entrés, sauf dans le cas où l'écho a été désactivé par un appel adéquat à la fonction `cons_echo`.

6.3.3 Pilote de console

Il reste à intégrer les pilotes du clavier et de l'écran avec le reste du système. En particulier, un processus qui exécute une opération de lecture doit se bloquer jusqu'à ce que des caractères soient effectivement disponibles.

6.3.4 Extension

En option, vous pouvez programmer les diodes du clavier pour que leur affichage soit cohérent avec l'état de l'automate de conversion. Lorsque des modifications doivent être effectuées, `do_scancode` appelle la fonction `kbd_leds` en fournissant en paramètre un octet correspondant à la nouvelle valeur des diodes :

Bits 3-7 : 0

Bit 2 : Capslock led (1= on, 0= off)

Bit 1 : Numlock led (1= on, 0= off)

Bit 0 : Scroll lock led (1=on, 0=off)

Les diodes peuvent être éclairées ou éteintes en envoyant sur le port $0x60$ d'abord la commande $0xed$, puis, après un petit délai, l'octet reçu par `kbd_leds`.