Mini-HOWTO programmation des ports d'E/S sous Linux

(c) 1995 Riku Saikkonen rjs@spider.compart.fi 26 Dec 1995

Ce HOWTO traite de l'utilisation des ports d'E/S ainsi que de la programmation de mini-temporisations (de quelques microsecondes à quelques millisecondes) en C sous Linux (mode utilisateur) sur processeur Intel x86. Ce document est issu du minuscule IO-Port mini-HOWTO du même auteur. Si vous avez des modifications à apporter ou des compléments à ajouter, n'hésitez pas à m'envoyer un message (rjs@spider.compart.fi)... Innombrables modifications depuis la précédente version (16 Nov 1995) dont l'ajout des spécifications du port parallèle. Adaptation française réalisée par Nicolas Lejeune (nl@freenix.fr).

Table des matières

1 Utilisation des ports d'E/S dans les programmes C

1.1 Méthode classique

Les routines permettant l'accès aux ports d'E/S sont définies dans **/usr/include/asm/io.h** (ou **linux/include/asm-i386/io.h** dans les sources du noyau). Ce sont des macros « inline », il suffit donc de #inclure **<asm/io.h>** ; Aucune autre bibliothèque (*library*, NDT) n'est requise.

Du fait d'une limitation de **gcc** (au moins jusqu'à la version 2.7.0 comprise), vous **devez** compiler tout code source utilisant ces routines avec les options d'optimisation (i.e. *gcc -O*). Une autre limitation de **gcc** empêche de compiler à la fois avec les options d'optimisation et de mise au point (*-g*). Cela signifie que si vous désirez utiliser **gdb** sur un programme manipulant les ports d'E/S, il est judicieux de mettre les routines utilisant les ports d'E/S dans un fichier source séparé, puis, lors de la mise au point, de compiler ce fichier source avec l'option d'optimisation, le reste avec l'option de mise au point.

Avant d'utiliser un port, il faut donner à votre programme la permission de le faire. Il suffit pour cela d'appeler la fonction **ioperm(2)** (déclarée dans **unistd.h** et définie dans le noyau) quelque part au début de votre application (avant tout accès à un port d'E/S). La syntaxe est **ioperm(from,num,turn_on)**, où **from** représente le premier numéro de port et **num** le nombre de ports consécutifs à rendre accessibles. Par exemple, **ioperm(0x300,5,1);** autoriserait l'accès aux ports 0x300 à 0x304 (5 ports au total). Le dernier argument est un booléen précisant si l'on désire donner (vrai (1)) ou retirer (faux (0)) l'accès au port. Pour autoriser plusieurs ports non consécutifs, on peut appeler **ioperm()** autant que nécessaire. Consultez la page de manuel de **ioperm(2)** pour avoir des précisions sur la syntaxe.

Votre programme ne peut appeler **ioperm()** que s'il possède les privilèges de root ; pour cela, vous devez soit le lancer comme utilisateur root, soit le rendre suid root. Il devrait être possible (Je n'ai pas essayé ; SVP, envoyez-moi un message si vous l'avez fait) d'abandonner les privilèges de root une fois l'accès aux ports obtenu par **ioperm()**. Il n'est pas nécessaire d'appeler **ioperm(...,0)** à la fin du programme pour abandonner explicitement les droits, cette procédure étant automatique.

Les privilèges accordés par **ioperm()** demeurent lors d'un **fork()**, **exec()** ou **setuid()** en un utilisateur autre que root.

ioperm() ne permet l'accès qu'aux ports 0x000 à 0x3ff ; pour les ports supérieurs, il faut utiliser **iopl(2)** (qui donne des droits sur tous les ports d'un coup) ; je ne l'ai jamais fait, regardez le manuel pour en savoir plus. Je suppose que l'argument **level** doit valoir 3 pour autoriser l'accès. SVP, envoyez-moi un message si vous avez des précisions à ce sujet.

Maintenant, l'utilisation proprement dite... Pour lire un octet sur un port, appelez **inb(port);** qui retourne l'octet correspondant. Pour écrire un octet, appelez **outb(value, port);** (attention à l'ordre des paramètres). Pour lire un mot sur les ports x et x+1 (mot formé par un octet de chaque port, comme l'instruction INW en assembleur), appelez **inw(x);**. Pour écrire un mot vers deux ports, **outw(value,x);**.

Les macros **inb_p()**, **outb_p()**, **inw_p()** et **outw_p()** fonctionnent de la même façon que celles précédemment évoquées, mais elles respectent, en plus, une courte attente (environ une microseconde) après l'accès au port; vous pouvez passer l'attente à quatre microsecondes en #définissant **REALLY_SLOW_IO** avant d'inclure **asm/io.h**. Ces macros créent cette temporisation en écrivant (à moins que vous ne #définissiez **SLOW_IO_BY_JUMPING**, moins précis certainement) dans le port 0x80, vous devez donc préalablement autoriser l'accès à ce port 0x80 avec **ioperm()** (les écriture vers le port 0x80 ne devraient pas affecter le fonctionnement du système par ailleurs). Pour des méthodes de temporisations plus souples, lisez plus loin.

Les pages de manuels associées à ces macros paraitront dans une version future des pages de manuels de Linux.

1.2 Problèmes

1.2.1 Je récolte des segmentation faults lorsque j'accède aux ports !

Soit votre programme n'a pas les privilèges de root, soit l'appel à **ioperm()** a échoué pour quelqu'autre raison. Vérifiez la valeur de retour de **ioperm()**.

1.2.2 Je ne trouve pas les définitions des fonctions in*(), out*(), gcc se plaint de références inconnues !

Vous n'avez pas compilé avec l'option d'optimisation (-*O*), et donc gcc n'a pas pu définir les macros dans **asm/io.h**. Ou alors vous n'avez pas #inclus <**asm/io.h**>.

1.3 Une autre méthode

Une autre méthode consiste à ouvrir **/dev/port** (un périphérique caractère, major number 1, minor number 4) en lecture et/ou écriture (en utilisant les fonctions habituelles d'accès aux fichiers, **open()** etc. - les fonctions **f*()** de stdio utilisent des tampons internes, évitez-les). Puis positionnez-vous (*seek*, NDT) au niveau de l'octet approprié dans le fichier (position 0 dans le fichier = port 0, position 1 = port 1, etc.), lisez-y ou écrivez-y ensuite un octet ou un mot. Je n'ai pas vraiment essayé et je ne suis pas absolument certain que cela marche ainsi ; envoyez-moi un message si vous avez des détails.

Bien évidemment, votre programme doit posséder les bons droits d'accès en lecture/écriture sur **/dev/port**. Cette méthode est probablement plus lente que la méthode traditionnelle évoquée auparavant.

1.4 Interruptions (IRQs) et DMA

Pour autant que je sache, il n'est pas possible d'utiliser les IRQs ou DMA directement dans un programme en mode utilisateur. Vous devez écrire un pilote dans le noyau voyez le Linux Kernel Hacker's Guide (khg-x.yy) pour les détails et les sources du noyau pour des exemples.

2 Réglages de haute précision

2.1 Temporisations

Tout d'abord, je dois préciser que, du fait de la nature multi-tâches préemptive de Linux, on ne peut pas garantir à un programme en mode utilisateur un contrôle exact du temps. Votre processus peut perdre l'usage du processeur à n'importe quel instant pour une période allant d'environ 20 millisecondes à quelques secondes (sur un système lourdement chargé). Néanmoins, pour la plupart des applications utilisant les ports d'E/S, cela ne pose pas de problèmes. Pour minimiser cet inconvénient, vous pouvez augmenter la priorité (avec **nice**) de votre programme.

Il y a eu des discussions sur des projets de noyaux Linux temps-réel prenant ce phénomène en compte dans *comp.os.linux.development.system*, mais j'ignore leur avancement ; renseignez-vous dans ce groupe de discussion. Si vous en savez davantage, envoyez-moi un message...

Maintenant, commençons par le plus facile. Pour des délais de plusieurs secondes, la meilleure fonction reste probablement **sleep(3)**. Pour des attentes de quelques dixièmes de secondes (20 ms semble un minimum), **usleep(3)** devrait convenir. Ces fonctions rendent le processeur aux autres processus, ce qui ne gâche pas de temps machine. Consultez les pages des manuels pour les détails.

Pour des temporisations inférieures à 20 millisecondes environ (suivant la vitesse de votre processeur et de votre machine, ainsi que la charge du système), il faut proscrire l'abandon du processeur car l'ordonnanceur de Linux ne rendrait le contrôle à votre processus qu'après 20 millisecondes minimum (en général). De ce fait, pour des temporisations courtes, **usleep(3)** attendra souvent sensiblement plus longtemps que ce que vous avez spécifié, au moins 20 ms.

Pour les délais courts (de quelques dizaines de microsecondes à quelques millisecondes), la méthode la plus simple consiste à utiliser **udelay()**, définie dans **/usr/include/asm/delay.h** (**linux/include/asm-i386/delay.h**). **udelay()** prend comme unique argument le nombre de microsecondes à attendre (unsigned long) et ne renvoie rien. L'attente dure quelques microsecondes de plus que le paramètre spécifié à cause du temps de calcul de la durée d'attente (voyez **delay.h** pour les détails).

Pour utiliser **udelay()** en dehors du noyau, la variable (unsigned long) **loops_per_sec** doit être être définie avec la bonne valeur. Autant que je sache, la seule façon de récupérer cette valeur depuis le noyau consiste à lire le nombre de BogoMips dans **/proc/cpuinfo** puis à le multiplier par 500000. On obtient ainsi une évaluation (imprécise) de **loops_per_sec**.

Pour les temporisations encore plus courtes, il existe plusieurs solutions. Ecrire n'importe quel octet sur le port 0x80 (voyez plus haut la manière de procéder) doit provoquer une attente d'exactement 1 micro-seconde, quelque soit le type et la vitesse de votre processeur. Cette écriture ne devrait pas avoir d'effets secondaires sur une machine standard (et certains pilotes de périphériques du noyau l'utilisent). C'est ainsi que **{in|out}{b|w}_p()** réalise normalement sa temporisation (voyez **asm/io.h**).

Si vous connaissez le type de processeur et la vitesse de l'horloge de la machine sur laquelle votre programme tournera, vous pouvez coder des délais plus courts « en dur » en exécutant certaines instructions d'assembleur (mais souvenez-vous que votre processus peut perdre le processeur à tout instant, et, par conséquent, que l'attente peut, de temps à autres, s'avérer beaucoup plus importante). Dans la table suivante, la durée d'un cycle d'horloge est déterminée par la vitesse interne du processeur ; par exemple, pour un processeur à 50MHz (486DX-50 ou 486DX2-50), un cycle prend 1/50000000 seconde.

```
Instruction    cycles sur i386     cycles sur i486
nop                   3                   1
xchg %ax,%ax          3                   3
or %ax,%ax            2                   1
mov %ax,%ax           2                   1
```

```
add %ax,0              2                    1

{source : Borland Turbo Assembler 3.0 Quick Reference}
```

(désolé, je n'ai pas de valeurs pour les Pentiums ce sont probablement les mêmes que pour i486)

(Je ne connais pas d'instruction qui n'utilise qu'un seul cycle sur i386)

Les instructions **nop** et **xchg** du tableau n'ont pas d'effets de bord. Les autres peuvent modifier le registre des indicateurs, mais cela ne devrait pas avoir de conséquences puisque **gcc** est sensé le détecter.

Pour vous servir de cette astuce, appelez **asm("intruction");** dans votre programme. Pour "instruction", utilisez la même syntaxe que dans la table précédente ; pour avoir plusieurs instructions dans un même **asm()**, faites **asm("instruction; instruction; instruction");**. Comme **asm()** est traduit en langage d'assemblage « inline » par gcc, il n'y a pas de perte de temps consécutive à un éventuel appel de fonction.

L'architecture des Intel x86 n'autorise pas de temporisations inférieures à un cycle d'horloge.

2.2 Chronométrages

Pour des chronométrages à la seconde près, le plus simple consiste probablement à utiliser **time(2)**. Pour des temps plus fins, **gettimeofday(2)** fournit une précision d'une microseconde (voyez toutefois, plus haut, les remarques concernant l'ordonnancement).

Si vous désirez que votre processus reçoive un signal après un certain laps de temps, utilisez **setitimer(2)**. Consultez les pages des manuels des différentes fonctions pour les détails.

3 Quelques ports utiles

Voici quelques informations concernant la programmation des ports les plus courants, pouvant servir, à des fins diverses, d'E/S TTL.

3.1 Le port parallèle

Le port parallèle (BASE = 0x3bc pour /dev/lp0, 0x378 pour /dev/lp1 et 0x278 pour /dev/lp2) : {source : *IBM PS/2 model 50/60 Technical Reference*, et quelques expériences}

En plus du mode standard, monodirectionnel en sortie, il existe, pour la plupart des ports parallèles, un mode « étendu » bidirectionnel. Ce mode possède un bit de sens qui peut être positionné en lecture ou écriture. Malheurement, j'ignore comment sélectionner ce mode étendu (il ne l'est pas par défaut)...

Le port BASE+0 (port de données) contrôle les signaux de données du port (D0 à D7 pour les bits 0 à 7, respectivement ; états : 0 = bas (0V), 1 = haut (5V)). Une écriture sur ce port recopie (*latches*, NDT) les données sur les broches. En mode d'écriture standard ou étendu, une lecture renvoie les dernières données écrites. En mode de lecture étendu, une lecture renvoie les données présentes sur les broches du périphérique connecté.

Le port BASE+1 (port d'état), en lecture seule, renvoie l'état des signaux d'entrée suivants :

Bits 0 et 1
: réservés.

Bit 2
: IRQ status (ne correspond à aucune broche, j'ignore comment il se comporte)

Bit 3
: -ERROR (0=haut)

Bit 4
: SLCT (1=haut)

Bit 5
: PE (1=haut)

Bit 6
: -ACK (0=haut)

Bit 7
: -BUSY (0=haut)

(Je ne suis pas certain des états hauts et bas.)

Le port BASE+2 (port de contrôle), en écriture seule (une lecture renvoie la dernière donnée écrite), contrôle les signaux d'états suivants :

Bit 0
: -STROBE (0=haut)

Bit 1
: AUTO_FD_XT (1=haut)

Bit 2
: -INIT (0=haut)

Bit 3
: SLCT_IN (1=haut)

Bit 4
: si positionné à 1, autorise l'IRQ associée au port parallèle (qui intervient lors de la transition de -ACK de bas à haut).

Bit 5
: commande le sens du mode étendu (0 = écriture, 1 = lecture), en écriture seule (une lecture ne renvoie rien d'utile sur ce bit).

Bits 6 et 7

réservés.

(Là non plus, je ne suis pas certain des états hauts et bas.)

Brochage (un connecteur 25 broches femelle sur le port) (*e*=entrée, *s*=sortie) :

1*es* -STROBE, **2***es* D0, **3***es* D1, **4***es* D2, **5***es* D3, **6***es* D4, **7***es* D5, **8***es* D6, **9***es* D7, **10***e* -ACK, **11***e* -BUSY, **12***e* PE, **13***e* SLCT, **14***s* AUTO_FD_XT, **15***e* -ERROR, **16***s* -INIT, **17***s* SLCT_IN, **18-25** Masse.

Les spécifications d'IBM précisent que les broches 1, 14, 16 et 17 (les sorties de contrôle) sont à collecteurs ouverts, connectées au 5V à travers des résistances de 4,7kiloohms (puits 20mA, source 0,55mA, niveau de sortie haut 5V moins la tension aux bornes de la résistance). Les autres broches ont un courant de puits de 24mA, de source de 15mA et leur niveau de sortie haut est supérieur à 2,4V. L'état bas dans les deux cas est inférieur à 0,5V. Il est probable que les ports parallèles des clones s'écartent de cette norme.

Enfin, un avertissement : attention à la mise à la masse. J'ai endommagé plusieurs ports parallèles en les connectant alors que la machine fonctionnait. Il est conseillé d'utiliser un port parallèle non intégré à la carte mère pour faire des choses pareilles.

3.2 Le port jeu

Le port jeu (ports 0x200-0x207) : je n'ai pas de spécifications là-dessus, mais je pense qu'il doit y avoir au moins quelques entrées TTL et un peu de puissance en sortie. Si quelqu'un possède plus d'informations, qu'il me le fasse savoir...

3.3 E/S analogiques

Si vous voulez des E/S analogiques, vous pouvez connecter des circuits convertisseurs analogiques-numériques (ADC) et/ou numériques-analogiques (DAC) sur ces ports (astuce : pour l'alimentation, utilisez un connecteur d'alimentation (de lecteur) inutilisé que vous sortirez du boitier, à moins que votre composant ne consomme très peu, auquel cas le port lui-même peut fournir la puissance). Sinon, achetez une carte AD/DA (la plupart sont contrôlées par les ports d'E/S). Ou, si vous pouvez vous contenter de 1 ou 2 voies, peu précises, et (probablement) mal réglées en zéro, une carte son à bas prix, supportée par le pilote sonore de Linux, devrait faire l'affaire (et se montrera plutôt rapide).

4 Ce qu'il reste à faire

- vérifier ce dont je n'étais pas sûr
- donner des exemples simples d'utilisation des fonctions décrites

Merci pour les nombreuses corrections et additions utiles que j'ai reçues.

Fin du mini-HOWTO programmation des ports d'E/S sous Linux