SOMMAIRE

EDITO

Spécial Little Zeus

D'après votre courier, la Tube-Intro de Little Zeus vous pas-sionne. Ce qui se conçoit très facilement, étant donné d'une part, qu'elle est superbe, et que d'autre part, la plupart des gens veut absolument créer leur propre intro ou démo, pour finalement aboutir à leur propre jeu. Ce qui, également, se conçoit aisément.

C'est pourquoi, en accord avec les Little Zeus Brothers (qui, soit dit en passant, ne sont absolument pas "incorporated" et encore moins "limited"), nous avons décidé de passer le listing source de la Tube-Intro dans son intégralité. Ce qui sera fait dès le numéro 18, vue la place qu'il occupera.

Ce même listing source sera également mis en téléchargement sur le 36-15 COMREV, à l'intention des fainéants du clavier.

En attendant, précipitez-vous sur le Virus Killer, la nouvelle et dernière application de Little Zeus.

Stéphane Schreiber

ExecBase

VOTRE CERVEAU NOUS INTERESSE

L'Amiga NewsTech de Commodore Revue, plus familièrement dénomée ANT, est en constante évolution. Nous essayons de faire en sorte de la rendre accessible à tous les programmeurs de France, de Navarre et d'ailleurs, quelque soit leur niveau : débutant, professionnel ou bidouilleur. Quelques rubriques nouvelles ont été créées, comme Transactor, Requester ou ViewPort, pour ne citer qu'elles. Leur but ? Communiquer. Tout simplement.

Tout ça pour dire qu'on a besoin de vous. La programmation ne peut que vous passionner, sinon, vous ne seriez pas en train de lire cette ANT. Tiens; regardez un peu autour de vous : le succès du domaine public, pensez-vous réellement qu'il ne soit dû qu'à la gratuité des programmes qu'on y trouve ? Ou bien le fait d'avoir à sa disposition, dans 80% des cas, le code source avec l'exécutable attire-t-il également pas mal de monde ?

En fait, et au risque de me répéter, l'Amiga NewsTech est désormais ouverte à tous. C'est votre journal à l'intérieur du journal. Ce qui signifie que n'importe qui peut y participer, la seule condition étant que cette personne ait quelque chose d'intéressant à dire. Intéressant pour les autres, s'entend.

N'importe qui peut venir expliquer dans Transactor des problèmes de programmation auxquels il a été confronté à un moment donné, et auxquels d'autres peuvent être confrontés.

N'importe qui peut venir parler dans ExecBase de sujets en rapport avec l'Amiga ou l'ANT qui lui tiennent à coeur.

N'importe qui enfin, et à plus forte raison encore, peut écrire à Requester pour essayer de trouver une réponse à ses difficultés.

De plus, toute participation à quelque rubrique que ce soit, à l'exception évidemment du courrier, sera rémunérée.

Pour reprendre la conclusion de l'édito du mois dernier : *"la balle est dans votre camp, à vous de jouer"*.

L'AMIGADOS : SHELL ET SCRIPTS

A lire ma prose au fil des numéros de Commodore Revus, vous avez sans nul doute remarqué que je suis un maniaque du clavier. Tout ce qu'un utilisateur normal réalise avec les menus du Workbench, moi, je le réalise à partir du CLI!

Je fais même beaucoup plus, puisque les commandes de l'AmigaDos permettent un contrôle beaucoup plus grand de la machine. Mais le principal attrait de l'AmigaDos réside dans sa programmation. Toute procédure complexe, c'est-à-dire utilisant plus de deux commandes, mérite d'être enregistrée dans un fichier Script. Le Shell du système 1.3 a apporté beaucoup dans ce domaine, puisqu'il incorpore deux outils majeurs : les alias et les attributs de fichiers.

Les alias sont des synonymes aux commandes trop longues pour être tapées aisément. En voici quelques exemples, tirés du fichier de configuration du Shell, le S:SHELL-Startup :

```
Prompt "%N.%S "
```

Change l'interrogation du système. %N est remplacé par le numéro du Shell en cours, %S par le répertoire courant.

```
alias xcopy copy [] clone
```

Les crochets [] sont remplacé par les arguments de la commande. Ainsi par exemple, xcopy df0: dh0: all est équivalent à copy df0: dh0: all clone.

```
alias endshell endcli
```

Cet alias est le plus simple que l'on puisse trouver.

```
alias pro execute s:spat protect[]
```

Crée une commande nommée "pro", qui n'est autre que la commande protect avec possibilité de spécifier une sélection de fichiers grâce à l'utilisation du script s:spat.

```
alias sdate execute s:spat setdate []
```

De manière identique, "sdate" permet de modifier les dates de création de plusieurs fichiers à la fois.

```
alias ren execute s:dpat rename []
```

Ici encore, les alias permettent de corriger les commandes existantes, en leur ajoutant la gestion des noms de fichiers.

```
alias clear echo "*E[0,0H*E[J"
```

Ici, ce sont les paramètres complexes d'une commande qui sont inclus dans le synonyme. On utilise les caractères de contrôle du standard ANSI pour modifier le comportement des fenêtres CON:

```
alias reverse echo "*E[0;0H*E[41,30m*E[J
```

Il faut essayer clear, reverse et normal avant de jouer avec ces caractères de contrôle...

```
alis normal echo "*E[0,0H*E[40;31m*E[J"
```

Jusque là, nous avions les alias standards de Commodore. Rien ne nous empêche bien entendu d'en ajouter quelques-uns bien de chez nous.

```
alias del delete
alias cys cd sys:
alias md makedir
alias ls list [] quick
alias x execute
alias sable protect [] +s
alias m execute s:spat more #?[]
alias prog newshell from s:shell-startup-prog
alias fmt sys:system/format "NIL: drive [] name "vide"
```

X est une réminescence du système 1.2 où il fallait taper la commande "execute" à chaque fois que l'on souhaitait utiliser un Scrip. Comme on n'avait pas d'alias, j'avais dupliqué la commande c:Execute sous le nom de c:X.

Xable est un alias indispensable à ceux qui souhaitent manipuler beaucoup de Scripts; il permet par exemple de rendre EXECUTE-able tous les fichiers d'un répertoire.

Le programme d'affichage de texte More est si pratique que je l'ai transféré dans mon tiroir C: des commandes systèmes. Cependant, il ne comprend pas les spécifications évoluées de fichiers; avec cet alias, je peux désormais faire "m s:#?" pour visionner agréablement tous les fichiers du répertoire S:.

"Prog" utilise un petit truc qui me permet de créer des Shells avec un jeu d'alias spécifiques. Le paramètre FROM de la commande NewShell lui permet d'exécuter les commandes contenues dans un fichier, en lieu et place de celles contenues dans le S:SHELL Startup standard.

"Fmt" et peut-être le plus beau de mes alias actuels. Il intègre un raffinement extrêmement pratique : la redirection grâce au signe inférieur (. En effet, la commande format, qui attend normalement l'appui sur la touche return pour fonctionner, prend ici la suite de ses commandes depuis le gestionnaire de périphérique NIL:. Le résultat est que cette commande n'attend plus l'utilisateur. Attention, c'est également le plus sûr moyen d'écraser le contenu d'une disquette si l'on n'est pas très prudent. Notez aussi que si vous souhaitez utiliser les caractères de redirection des entrées-sorties dans un fichier Script, il est impératif de mettre en tête du fichier les trois instructions suivantes :

```
key dummy
bra [
ket ]
```

qui créèent un argument bidon (dummy) et remplacent les délimiteurs d'arguments, habituellement (et), par < et >.

Le fichier S:SHELL-Startup peut se révéler d'une très grande utilité, mais il ne faut pas cependant abuser en y insérant trop de commandes AmigaDos normales. En effet, si la commande alias, interne au SHELL, est exécutée instantanément, il n'en est pas de même pour les commandes que l'AmigaDos doit charger depuis le disque système. Comme les commandes du fichier SHELL-Startup sont exécutées à chaque lancement du SHELL, les utilisateurs

normaux préfèrent ne pas attendre. Lorsque la passion de la programmation me prend, j'utilise la commande prog (voir son alias plus haut) : le fichier S:SHELL-Startup-prog contient toutes les instructions du fichier S:SHELL-Startup normal, plus quelques commandes spécifiques à mon environnement, que voici :

```
cd sources :      ; direction mon répertoire préféré
avail             ; une bonne habitude à prendre pour éviter
status            ; quelques crashes très désagréables :
stack 10000       ; un peu de marge pour mes utilitaires
changetaskpri 5 : dans le cas d'une utilisation intensive
: en multi-tâche, j'alloue toujours une forte priorité aux
: processus CLI-SHELL à partir desquels je travaille. Par
: contre les CLI-SHELL d'exécution des programmes en cours de
: tests sont remis à un niveau normal de priorité (par un
: CHANGETASKPRI 0). Ainsi, lorsque le programme testé boucle
: sans fin, il ne gêne absolument pas le déroulement de mon
: travail.
```

A titre d'exercice, je vous suggère d'essayer de créer un fichier de configuration S:SHELL-Startup-R récursif, dans le genre :

```
newshell newcon:10/10/200/100/Test from S:SHELL-Startup-R.
```

On utilise ici le gestionnaire de fenêtre-console NewCON:, plus évolué que la version précédente CON:. De plus, en spécifiant les coordonnées, on ouvre des fenêtres plus petites, ce qui permet de faire fonctionner des tests sur des Amigas dotés de seulement 512 Ko de mémoire. Avant que de le lancer, vous pouvez utiliser la commande

```
resident c:newcli
```

pour accélérer le déroulement des opérations. Tapez ensuite la commande

```
execute S:SHELL-Startup-R
```

ou bien

```
protect S:SHELL-Startup-R +s
SHELL-Startup R
```

Ce test met en évidence deux choses : les fichiers de configuration des SHELL (Shell-Startup) ne sont que des fichiers scripts, et il existe une contrainte dans l'AmigaDos qui limite le nombre de processus CLI-SHELL actifs (pour ceux qui ne suivent pas cette série d'articles sur leur ordinateur, la limite est 20).

Toujours dans le domaine des fichiers Scripts, Carolyn Scheppner (responsable du support développeurs aux Etats-Unis) a démontré que l'AmigaDos pouvait abandonner son côté rébarbatif lorsqu'on l'utilisait pour monter une blague. Tapez le texte des deux fichiers suivants avec votre éditeur préféré (Ed ou MEmacs par exemple). Placez-vous ensuite, par la commande cd, dans le répertoire où vous avez mis les programmes, et exécutez le fichier Gremlette. Passez finalement le clavier de l'Amiga à votre victime (surtout si elle prétend connaître l'AmigaDos).

Voici le contenu du fichier S:Gremlette, qui permet de lancer aisément le programme principal S:Gremlin.

```
; Fichier Gremlette
if exists Gremlin
   Execute Gremlin
else
   if exists s Gremlin
      Execute s:Gremlin
   end if
endif
```

Voici maintenant le contenu du fichier S:Gremlin, dont les commentaires devraient vous permettre de ne pas être trop surpris lors de l'exécution.

```
; Fichier Gremlin
.key dummy
.bra [
.ket ]
;
Failat 30
;
Lab prom
Assign )NIL: sg[$$]: exists
If NOT Warn
   Assign sg[$$]:
Else
   Assign sg[$$]: Sys:
Endif
; astuce de programmation établissant une bascule :
; si le répertoire sg(numéro du cli) avait un sens, il est
; est détruit ; s'il n'existe pas, il est créé.
Echo "[$$]è.RAM DISK:) " NOLINE
; affiche une chaine de caractères qui ressemble étrangement à
; un prompt
Skip "NIL: ?
; la commande skip, qui permet de sauter à une adresse dans le
; fichier script, attend de l'utilisateur cette adresse, mais
; sans lui afficher de renseignements grâce à la redirection
; vers NIL:
Lab Alias
   Echo "On m'appelle le Shell du GURU"
   Skip prom BACK

Lab Assign
   Echo "Le soleil brille
   Echo "Les arbres sont verdoyants"
   Echo "Les oiseaux chantent"
   Echo  Avez-vous écouté pousser votre barbe aujourd'hui ?"
   Skip prom BACK

Lab Avail
   Echo "Je suis prêt à tout"
   Skip prom BACK

Lab Cd
   Echo "Vous êtes perdu*N(moi aussi)*N(ou sont les secours)"
   Skip prom BACK

Lab Date
   Echo "Vendredi 1-Apr-88 00:00:00"
   Skip prom BACK

Lab Dir
   Echo "Juste quelques fichiers sans importance"
   Skip prom BACK

Lab Endcli
Lab Endshell
   Echo "Non merci, je pense rester encore un peu"
   Skip prom BACK

Lab Failat
   Echo "Je n'échoue jamais!!!"
   Skip prom BACK

   Echo "C'est VOTRE faute!"
   Skip prom BACK

Lab Help
   Echo "Il n'y a plus d'abonné au numéro que vous avez demandé"
   Skip prom BACK

Lab Info
   Echo "Rien d'intéressant"
   Skip prom BACK

Lab Ls
Lab List
   Echo "  Lait*N OEufs*N Papier Toilette"
   Skip prom BACK

Lab Newcli
Lab Newshell
   If exists Gremlette
      NewShell from Gremlette
   Else
      If exists s gremlette
         NewShell from s:gremlette
      Endif
   Else
      Echo "Non. On reste là."
   Endif
   Skip prom BACK
c'est ici que le fichier s gremlette prend tout son sens
```

```
Lab Path
   Echo "Je sais pas, je ne suis pas du quartier."
   Skip prom BACK

Lab Quit
   Echo "J'y suis, j'y reste"
   Skip prom BACK

Lab Status
   Echo "Moi ? Ça va bien, merci."
   Skip prom BACK

Lab Why
   Echo "Parce que."
   Skip prom BACK

Endskip
Lab Default
   Assign [NIL: sgl$$]: exists
   If Warn
      Echo "Pas ce soir, chéri, j'ai la migraine."
   Else
      Echo "Arrête, ou j'efface quelque chose."
   Endif
   Skip prom BACK
; voici un endroit où le balancement sur le nom sgl:numéro de CLI\
; permet de fournir alternativement deux réponses différentes.
```

Lorsque vous aurez tapé et fait fonctionner ce programme, la fonction de chacune de ses commandes devrait vous apparaître clairement. Le but général aussi, d'ailleurs, car après tout, il ne s'agit que d'un interpréteur de commandes bidon écrit en Amigados, soit en rapport avec un nom valide de commande, soit avec des réponses par défaut.

B. de Mil

Application

LE VIRUS-KILLER DE LITTLE-ZEUS : PREMIERE PARTIE

Trève de graphisme, nous allons profiter des virus pour explorer les fins fonds du hardware de l'Amiga. Effectivement, en exclusivité pour Commodore Revue, voici un détecteur de virus qui se reproduit et qui est capable de détecter toute disquette infectée dès lors que l'utilisateur en insère une nouvelle dans le drive.

Hé oui, il ne se contente pas, comme la majorité des tueurs de virus, d'exterminer ces éléments gênants lorsque nous procédons à un nouveau reset. Il est bien plus puissant que celà : imaginez que vous travaillez sous Prowrite et que, pour une raison ou une autre, vous deviez insérer une disquette sur laquelle vous désirez lire un fichier. Malheureusement, un virus figure sur cette disquette et s'installe en mémoire, puis se reproduit sur votre disquette de travail. Vous n'apprécierez certainement pas d'être définitivement interrompu après avoir tapé trois ou quatre pages...

Le Virus-Killer made in Little Zeus que nous vous proposons à partir de ce mois-ci, reconnaît le virus dès lors que vous avez inséré la disquette contaminée dans l'un des lecteurs attachés à l'ordinateur, la purifie aussitôt, puis efface le virus logé dans la mémoire.

Nous nous sommes toujours efforcés de présenter chaque routine comme un exemple d'application de cours théorique. Le Virus-Killer que nous allons étudier s'inscrit dans cette optique; nous ne doutons pas que vous ayiez déjà à votre disposition quelque anti-virus (VirusX ou autre)... Néanmoins, le nôtre, outre le fait qu'il se démarque des autres par son originalité et sa puissance, va nous servir de prétexte pour examiner les "devices" de l'Amiga, en particulier le lecteur de disquettes.

BREVE THEORIE SUR LES ENTREES/SORTIES

Les entrées/sorties (ou input/output, souvent notées I/O) désignent tout ce qui concerne les échanges de données entre l'ordinateur et ses périphériques (imprimante, modem mais aussi et surtout pour le cas qui nous intéresse, lecteurs de disquettes).

Il existe deux moyens de les gérer : le premier, bien que très simple car faisant appel aux routines de la "dos. library", apparaît limité, cette dernière ne pouvant fonctionner en multi-tâche; le second consiste à faire appel aux "devices". Cette méthode permet d'effectuer toutes les opérations d'entrées/sorties tout en poursuivant le programme en cours. Vous l'aurez sans aucun doute deviné, c'est cette méthode qui va nous intéresser, car, ne l'oublions pas, le Virus-Killer doit pouvoir détecter (et éliminer!) les virus tout en laissant les différentes tâches en cours se poursuivre normalement.

LES DEVICES

Nous venons d'énoncer le mot "device" sans pour autant l'avoir défini. Qu'à celà ne tienne, voici une définition bien exhaustive : les devices sont des programmes implantés en ROM, habituellement appelés par les routines des différentes librairies (mais nous allons voir que l'on peut procéder autrement). Dans des conditions normales (ce que nous ne faisons jamais car nous préférons bidouiller afin de présenter des techniques plus intéressantes...), l'utilisation des devices implique quatre étapes (quatre appels de fonctions de la librairie Exec), que voici :

- *FindTask()* permet de pointer sur une tâche. Avec un paramètre de valeur 0, c'est la tâche en cours qui est pointée, en l'occurence, le programme à partir duquel on a appelé FindTask;
- *AddPort()* structure un port réponse au device dont on fournit préalablement l'adresse de base;
- *OpenDevice()* ouvre le device et initialise la structure I/O;
- *Dolo()* lance le Device.

C'est intentionnellement que ces explications sont très superficielles; effectivement, nous survolons ces étapes uniquement dans un dessein démonstratif, pour attirer votre attention sur le fait que "passer par la porte de devant" ferait tellement appel aux librairies que nous perdrions un temps fou, ainsi que toute liberté de programmation. Nous allons donc nous intéresser uniquement aux I/O et à Dolo.

Le tableau qui suit indique les différentes adresses des I/O. Attention, ces dernières sont données en offset négatif, c'est-à-dire que le nombre correspondant à chacune des adresses ôté à l'adresse de base donne l'adresse de la case mémoire qui nous intéresse. Ainsi, pour la dernière ligne du tableau I/O_Data est la case mémoire ayant pour adresse valeur de base mois 40.

Offset	Nom	Fonction
0	Succ	Prochaine entrée
4	Pred	Entrée précédente
8	Type	Type d'entrée
9	Pri	Priorité
10	Name	Nom
14	ReplyPort	ReplyPort
31	I/O_Error	Niveau de l'erreur
32	I/O_Actual	Nombre d'octets transférés
36	I/O_Length	Nombre d'octets à transférer
40	I/O_Data	Tampon des données

Nous avons sciemment omis d'inscrire dans le tableau certaines adresses, car peu utilisées.

Dolo, KESAKO ?

Dolo est une routine de la librairie Exec qui nécessite pour paramètre le nom du périphérique avec lequel elle doit travailler. En fait, le système d'exploitation fait perpétuellement appel à Dolo pour quelque opération d'entrée/sortie que ce soit. D'une manière plus explicite, si vous envoyez des données en direction de votre imprimante, le système d'application fait appel à Dolo en lui ayant préalablement spécifié la périphérie intéressée. Il en va de même pour le modem ou le lecteur de disquettes.

Nous allons avantageusement tirer profit de Dolo au moyen d'un subterfuge assez puissant : étant donné que toute disquette insérée dans le drive est lue par Dolo, il suffit de modifier cette routine de telle manière qu'au lieu de se charger uniquement d'assurer le transfert des données entre le drive et l'ordinateur, Dolo puisse aussi vérifier la disquette. Ainsi, à chaque fois que vous rentrerez une disquette dans l'un des deux drives, ces dernières seront automatiquement vérifiées. Cependant il reste un problème (que nous allons résoudre un peu plus loin) : comment être certain que Dolo a bel et bien été appelé pour faire l'un des drives et non pour l'imprimante ou un autre périphérique ?

Une fois certain qu'il est bien question de drive, il faut dans un premier temps repérer lequel (/DF0 à DF3), puis dans un second temps examiner la disquette, avant de proposer, le cas échéant, de la purifier et de copier le Virus-Killer sur la première piste).

Mais avant d'aller plus loin dans les détails, attardons-nous sur la structure d'une disquette et sur le fonctionnement de Dolo lorsqu'elle travaille exclusivement avec les drives.

LES LECTEURS DE DISQUETTES

Tout d'abord, un peu de théorie sur la structure d'une disquette : une disquette 3"1/2 compte habituellement 80 pistes, numérotiées de 0 à 79. Chaque piste correspond à un sillon : la piste 0 correspond au cercle concentrique périphérique et la piste 79 au cercle intérieur (fig. 1). La disquette compte deux côtés (ou faces) servant de support aux pistes (tracks ou cylindres, termes équivalents). Chaque cylindre connaît une sous-division en onze secteur (ou blocs). En général, et bien que ces deux termes désignent les mêmes éléments, on compte les secteurs de 0 à 10 pour chaque piste et les blocs de 0 à 1759 pour l'ensemble de la disquette. Enfin, chaque secteur contient 128 longs mots, soit 512 octets.

Celà ne signifie pas pour autant que nous avons à notre disposition $1760 \times 512 = 910.120$ octets disponibles sur la disquette : vous comprendrez facilement que les données se sont pas écrites aléatoirement mais suivant un plan. C'est d'ailleurs au sein de celui-ci que figurent toutes les données de bootage (on entend par "bootage" les toutes premières étapes de la lecture). Les blocs de bootage (en Anglais, "boot-blocs") sont les deux premiers de la disquette (numéros 0 et 1). Ils spécifient le type de la disquette, à savoir DOS ou non-DOS. De plus, ils indiquent au système d'exploitation le numéro du bloc racine (en général le 880^{e}) et contiennent parfois un programme "auto-boot".

Les programmeurs de virus, et nous aussi par la même occasion, ont profité de cette caractéristique des boot-blocks. Cependant, afin que l'utilisateur ne suspecte pas la présence du virus, celui-ci est conçu de telle manière qu'après avoir été chargé et exécuté, il poursuive la procédure normale de bootage (initialisation de l'AmigaDos, exécution de la Startup-Sequence, etc.). C'est d'ailleurs pour cette raison que l'implantation de virus sur des disquettes non "bootables" les rend inutilisables. A titre d'exemple, les logiciels de Psygnosis utilisent ce procédé du boot-block et toute écriture sur ce dernier déprave le jeu.

Par curiosité, sachez que l'Amiga lit alternativement les faces supérieures et inférieures des disquettes de la façon suivante : d'abord sont lus les blocs 0 à 10 qui se trouvent sur la piste 0 de la face 0, puis le bloc 11 qui lui, par contre, correspond au premier secteur de la première piste de la seconde face.

LES DISQUETTES ET LA ROUTINE

Après avoir repéré le drive mis en marche au moyen de Dolo, notre anti-virus doit s'assurer qu'il s'agit bel et bien d'un transfert de données entre les boot-blocks et la mémoire. Dans ce cas, tout laisse croire qu'un virus est en train de s'implanter en mémoire. Pour le confirmer, il suffit de comparer les chaînes de caractères typiques de chaque virus avec celles des boot-blocks se situant à l'emplacement sur lequel la présumée chaîne devrait se trouver (fig. 2). Dans la situation pour laquelle un virus existe, notre programme offre la possibilité de l'effacer et par la même occasion de se loger à sa place dans les boot-blocks.

UN SECRET BIEN INTERESSANT

Examinons maintenant de plus près le subterfuge qui nous permet de détourner la routine Dolo. Cette dernière prend place dans la librairie Exec à l'offset —456 (l'offset d'une librairie présente une certaine similitude avec celui d'un device : il correspond à la valeur à déduire à l'adresse de base de la librairie). Ainsi, pour appeler, d'une manière normale, Dolo, il faut écrire les deux lignes suivantes :

```
Move.l ExecBase,a6
jsr    —456(a6)
```

Une librairie se présente comme une suite de sauts (instruction jmp) aux adresses réelles de ses fonctions. Etant donné qu'elles se trouvent en Ram, il existe un procédé simple permettant de détourner le saut à une fonction donnée sur une routine de notre crû (ici, notre anti-virus) : il consiste à échanger l'adresse du jmp dans la librairie par celle de notre routine. Cependant, une difficulté apparaît : la routine de la librairie ne doit pas purement et simplement disparaître du système (sans Dolo, l'Amiga ne pourrait pour ainsi dire plus rien faire!); notre routine doit donc se charger de l'appeler avant de tester la disquette, éventuellement la purifier, puis retourner au programme en cours.

LE VECTEUR RESET

La toute première étape du Virus-Killer est sans doute sa mise en action. Ceci se fait dès l'instant où l'on effectue un reset (touches control/Amiga gauche/Amiga droit).

L'Amiga, comme la plupart des ordinateurs, stocke en mémoire l'adresse du programme à exécuter à la suite d'un reset. Ce vecteur est aussi appelé CoolCapture. Le principe est simple : il faut détourner CoolCapture sur l'adresse de notre routine. Ainsi, à chaque reset, notre programme est-il exécuté.

J'en vois certains qui perdent le fil de l'histoire : on a dit tout à l'heure que le Virus-Killer se mettait en marche via Dolo et voilà maintenant que nous déclarons que celà se fait après le reset! Ne vous alarmez pas, notre Virus-Killer est en fait "doublement" appelé. La raison en est simple : à chaque reset, l'Amiga remet en place tous ses pointeurs, dont évidemment Dolo. De fait, notre Virus-Killer, bien que toujours en mémoire, ne serait plus actif, s'il ne prenait le soin de détourner également CoolCapture sur la routine détournant Dolo... et CoolCapture pour survivre au prochain reset!

Dans un dessein didactique (mais aussi parce que nous avons rempli l'espace qui nous était imparti), nous vous laissons essayer de comprendre la partir du Virus-Killer publiée ce mois-ci sans autre explications. Effectivement, grâce à ce que nous avons étudié dans ce numéro et moyennant réflexion, vous devriez comprendre.

Little Zeus Brothers Incorporation

```
; ici figurera une autre partie
; de la routine

;
; -- Appelée après chaque Reset --
init:
reseta:
; initialise CoolCapture
  jsr initcc


  move.l $4,a6
  move.l -456(a6),d0
  move.l d0,rom1+2(pc)
  move.l d0,rom2+2(pc)


; détourne DoIo sur ltz
  move.l #ltz,-454(a6)
```

```
; indique par un effet de couleurs
; la présence du virus-killer
  move.w #$f000,d0
nop: move.w d0,$dff180
  dbra d0,nop

  move.w #1,time
  clr.w aut
  btst #6,$bfe001 ; test la souris
  bne mas

; remet l'ancienne DoIo
  move.l #$fc06dc,-454(a6)

; efface CoolCapture
  clr.l 46(a6)
  jmp mam ;
mas: rts
error: clr.w aut
  rts

ltz: tst.w aut
  bne endio
  cmp.l #$500,36(a1) ;nbre d'octets
  bcc endio ; si plus de $500
  tst.l 44(a1) ; piste 0 ?
  bne endio
  tst.l 40(a1) ; face 0 ?
  beq endio
  cmp.b #2,29(a1) ; lit 1 secteur ?
  bne endio

  move.b #'1',drive
  cmp.b #$78,27(a1) ;drive externe?
  bne mor
  move.b #'0',drive
mor: move.w #1,aut

  move.l a1,diskio
```

```
; sauve l'adresse du buffer
  move.l 40(a1),waic
  jsr $fc06dc ; on appelle DoIo
  move.l diskio,a1
  move.b 31(a1),nerror
  bne error
  movem.l a0-a6/d0-d7,-(a7)



; teste les virus 1 par 1
  lea virustab,a0
tvirus:
  cmp.l #virustabf,a0 ; virus ?
  beq sortie
  move.l (a0)+,d0
  move.l waic,a2
  add.l d0,a2
  move.l (a2),d0
  cmp.l (a0)+,d0
  bne tvirus
  tst.w time ; trouvé !
  beq sor
  clr.w time
  move.l waic,a0


; installe un boot-block standard
  lea install,a2
  move.w #12,d3
bcl3: move.l (a2)+,(a0)+
  dbra d3,bcl3
  move.w #242,d3
bcl4: clr.l (a0)+
  dbra d3,bcl4
sor:
;
; ici figurera une autre partie
; de la routine
```

LE DIR GTI Turbo ZX92

Bienvenue à notre second rendez-vous mensuel avec l'AmigaDos. La dernière fois, notre sonde spatiale COMREV-II s'est approchée de la planète List et nous a fait parvenir des informations surprenantes sur ses options; certaines d'entre elles étaient jusque-là inconnues...

Vous avez déjà certainement remarqué la lenteur de l'affichage des fichiers présents sur la disquette, lenteur qui reste encore le principal défaut de notre machine bien-aimée (en attendant le FFS sur disquette du Workbench 1.4). Mais Commodore Revue vous offre un remède : la création d'une instruction qui bâtira et qui sera capable de relire un agenda de tous les fichiers accessibles dans le répertoire courant.

Il s'agit en fait d'un Script que l'on entrera avec Ed ou MEmacs, et que l'on transformera en une instruction autonome grâce à la commande Protect. Voici le listage de Dirap (pour Dir rapide, pour ne pas écrire FastDir) :

```
; Ce programme permet d'afficher ou de créer une Liste
IF EXISTS Liste
   Type Liste
ELSE
   Echo "*NPas de liste crée"
   Ask "Voulez-vous en créer une (Ya ou Nein) ou (Yabon ou Ne)?"
   IF WARN
      List QUICK TO Liste
      Echo "*NVoila, c'est fait"
   ELSE
      Echo "D'accord, terminé "
   ENDIF
ENDIF
```

La recopie fastidieuse de cet interminable listing achevée, il vous reste à le sauvegarder sous le nom Dirap et à utiliser Protect :

Protect Dirap +s

Dirap est maintenant une commande que vous exécutez en entrant son nom tout simplement. Retenez bien cette syntaxe, elle est très utile.

Dirap cherche tout d'abord un fichier Liste qui contient tous les noms de fichiers dans le répertoire courant. Si cette liste n'existe pas, l'utilisateur sera consulté pour savoir s'il désire ou non la créer (Commande Ask). On utilise ici WARN avec la commande If, car Ask renvoie un code 5 si la réponse est Yes et 0 si c'est No. On peut aussi s'amuser à créer un programme qui se chargerait de la création de cette Liste, dans chaque répertoire, comme par exemple :

```
; Ce programme permet de créer dans un listage rapide (fast listing)
; dans le répertoire courant.
LIST QUICK TO Liste
```

Cette liste peut être consultée, mais on peut aussi effectuer des recherches très rapides grâce à l'instruction SEARCH.

SEARCH s'appliquera donc ici à une recherche sur un fichier. La syntaxe est la suivante :

SEARCH Liste "Texte cherché"

où "Texte cherché" sera en l'occurence, le nom du fichier voulu.

L'occasion se présentant, étudions les autres modes d'utilisation de SEARCH:

— Recherche d'un texte dans tous les fichiers d'un répertoire :

SEARCH SEARCH "Texte cherché"

— Recherche d'un texte dans tous les fichiers et sous-répertoires du répertoire courant :

SEARCH SEARCH "Texte cherché" ALL

— Recherche d'un fichier dans un répertoire :

SEARCH Tiroir FIL NomDuFichier

SEARCH peut être intégré à un Script, car dans la version 1.3, il renvoie un code 5 s'il a trouvé, sinon 0. Encore une fois, WARN convient très bien pour le traitement de ce code.

Il est possible d'interrompre la recherche dans un fichier en pressant CTRL/D, Search reprend alors avec le fichier suivant. Pour interrompre complètement la recherche, entrer CTRL/C.

C'est terminé pour aujourd'hui, il ne reste plus qu'à attendre la prochaine planète que croisera COMREV-II...

Un ami qui vous veut du bien.

Phantasia

L'ASSEMBLEUR 68000

Ca y est ? Vos petits doigts potelés ont tapé le long listing publié le mois dernier ? Très bien, on va donc pouvoir commencer les indispensables explications.

Toute la première partie du programme consiste en la définition de constantes que nous utiliserons par la suite, constantes qui simplifient à la fois la lisibilité et la maintenance du programme. La plupart des assembleurs d'aujourd'hui offrent la possibilité d'inclure, grâce à la directive "include", des fichiers au sein d'un source, fichiers qui contiennent les définitions propres au système sur lequel on travaille (pour les ceusses qui ne l'auraient pas remarqué, nous, c'est l'Amiga). Cette méthode possède certes de nombreux avantages (par exemple, en cas de nouvelle version du système, il suffit de réassembler le programme avec les nouveaux includes), mais j'ai toujours préféré m'en passer, pour des raisons de vitesse d'assemblage. J'ai horreur d'aller boire un café pendant que l'ordinateur travaille.

Arrive alors le début du programme, avec le label "Start". On sauvegarde tous les registres, à l'exception de d0 qui nous servira à signaler une éventuelle erreur durant l'initialisation.

La suite, vous la connaissez déjà : ouverture des différentes libraries utilisées, avec éventuellement sortie du programme en cas de défaillance (très improbable pour les librairies en Rom, mais on ne sait jamais), puis ouverture d'un nouvel écran Intuition, ouverture d'une fenêtre dans cet écran, et enfin mise en place d'un menu. Vous aurez remarqué que toutes ces "ouvertures" se font par l'appel de routines (par exemple, bsr OpenS pour l'écran), ce qui n'est peut-être pas très économique en place mémoire, mais rend le programme modulaire.

On appelle ensuite la routine principale, baptisée "main" en hommage à mes (désastreux) début en C, avant de tout refermer (menu, fenêtre, écran et libraries), de récupérer les registres sauvegardés, et de retourner au CLI.

Car en effet, ce programme n'est accessible qu'à partir du CLI (ou du Shell, bien sûr, c'est pareil). Lancé depuis le Workbench (ce qui supposerait que vous lui ayiez dessiné une icône), il plante. Nous verrons le mois prochain pourquoi et comment y remédier.

LA MAIN DANS LA MAIN

La routine principale "main" est en fait une boucle qui ne se terminera que lorsque l'utilisateur aura choisi l'option Quitter dans le menu. Elle comprend un test permanent dudit menu, grâce à la fonction GetMsg de la librarie Exec. Cette fonction renvoie le cas échéant dans d0.L un pointeur sur une structure Message dans laquelle on trouvera de plus amples informations sur la teneur du message, ou bien 0 s'il ne s'est rien passé. Je ne vais pas m'amuser à décrire en détails cette structure, seulement les éléments intéressants pour nous :

Adresse	Signification
Message + 20	Classe du message
Message + 24	Code du message
Message + 28	Adresse des informations

La classe du message, appelée dans le listing MsgClass, indique quel est le type du message reçu. Etant donné que notre fenêtre n'autorise qu'un seul type de message (voir les flags IDCMP dans la structure NewWindow), en l'occurence MENUPICK, c'est le seul que nous serons habilités à recevoir. Il nous suffit donc de comparer MsgClass à la valeur MENUPICK (définie en début de listing, ça fait plaisir de voir qu'il y en a qui suivent) pour savoir si une option du menu a été sélectionnée.

Le code du message (noté MsgCode, suivez sur le listing) nous aidera, lui, à déterminer quel point de menu ou de sous-menu a été sélectionné. Il s'agit d'un mot (16 bits, donc) codé comme suit :

MsgCode			
Groupes de bits	15 à 11	10 à 5	4 à 0
Signification	sous-menu	item	menu

Quelques décalages à droite (à l'aide de l'instruction lsr) suffisent donc à déterminer précisémment quel point à été choisi.

Quant à MsgAddress, il ne nous est d'aucune utilité pour le menu, alors que pour le resquester, si ; nous verrons donc plus loin comment l'utiliser.

AU MENU CE SOIR

Il nous faut maintenant réagir à la sélection du menu, ce qui est justement le but de la routine judicieusement intitulée DoMenu. Cette routine est divisée en plusieurs sous-routines DoMenu1, DoMenu1_1, DoMenu2, etc. Les plus perspicaces d'entre vous auront tout de suite compris que DoMenu1 s'occuppe du premier menu (celui intitulé Projet), que DoMenu1_1 s'occupe du premier point de ce premier menu (ici, Requester), etc.

Par contre, il peut sembler étrange qu'une routine DoMenu2 existe, alors que nous n'avons précisément pas défini de second menu. La raison est toute bête : cette manière de procéder, qui soit dit en passant n'économise pas non plus les octets, permet toutefois de modifier très aisément la structure même de la barre de menus, en ajoutant ou en enlevant des points, des sous-menus ou carrément des menu entiers. Ainsi, DoMenu2 se contente-t-elle de sauter à la boucle principale (bra MainLoop), mais si je désire ajouter un second menu, je n'aurai qu'à remplacer ce saut par une routine, identique dans l'esprit à DoMenu1.

Concrètement, comment DoMenu et ses consoeurs fonctionnent-elle ? En vérité, c'est très simple.

DoMenu se charge d'abord de déterminer quel point de menu a été sélectionné. Pour ce faire, elle calcule, à l'aide de MsgCode, le numéro de menu qu'elle stocke dans le registre d0, le numéro du point de menu, stocké dans d1 et le numéro d'un éventuel sous-menu, stocké dans d2. Dans notre cas, si l'utilisateur choisit la rubrique "Quitter", d0 contiendra la valeur 0 (indiquant la premier menu), d1, la valeur 1 (indiquant le second point de menu) et d2, la valeur $3f (indiquant qu'il ne s'agit pas d'un sous-menu). Une fois ces trois valeurs initialisées, on arrive à DoMenu1.

Celle-ci se charge de déterminer, simplement en testant le registre d0, si c'est bien le premier menu qui a été choisi, sinon, elle saute à DoMenu2. Si oui, on arrive à DoMenu1_1 qui, en testant le registre d1, est capable de déterminer s'il s'agissait du premier point du premier menu (Quitter), auquel cas elle sort de la boucle principale par un simple rts, sinon elle saute à DoMenu1_2.

De la même manière, DoMenu1_2 compare le registre d1 à pour s'assurer que c'est bien le second point du menu qui a été sélectionné, auquel cas elle appelle le sous-programme de gestion du Requester. Dans le cas contraire, elle saute à DoMenu1_3, et ainsi de suite.

Rusé, non ?

PLAT DE RESISTANCE : LE REQUESTER

Voici donc (enfin) la partie la plus intéressante de ce programme. Le requester d'exemple que j'ai construit comprend deux gadgets booléens ("OK" et "Ca Flashe"), un gadget de chaîne pour écrire ce qui vous passe par la tête, un gadget proportionnel horizontal, ainsi qu'un troisième gadget booléen, désactivé celui-là, contenant le texte d'illustration de la fonte chargée depuis le disque. Un requester supplémentaire est également mis en place par la fonction AutoRequest() d'Intuition lorsque l'utilisateur confirme son entrée de texte dans le string-gadget par la touche Return. Tout ça est finalement très simple, mais prend beaucoup de temps à construire.

La routine qui gère tout ça s'appelle, encore une fois assez judicieusement, DoRequest. C'est une bonne habitude à prendre que de définir un modèle pour les noms des routines, celà permet, plusieurs mois plus tard, de replonger dans un programme sans se demander pendant trois heures à quoi telle partie du listing peut bien servir... Chez moi par exemple (je prends cet exemple parce que c'est celui que je connais le mieux ; rien ne vous oblige à le suivre à la lettre), tout ce qui ouvre quelque chose s'appelle OpenMachin, ce qui ferme, CloseBidule et ce qui gère un élément quelconque, DoTruc. Les anti-franglais risquent de ne pas apprécier, mais on n'est pas là pour discuter de ce genre de choses.

Je disais donc : DoRequest.

La première chose à faire pour ouvrir un requester, est d'ouvrir une fenêtre dans laquelle il sera dessiné. C'est ainsi, on n'a pas le choix. J'ai donc défini une structure NewWindow particulière baptisée ReqWindowDefs, qui contient toutes les éléments nécessaires à notre requester. Cette fenêtre pourra être déplacée à volonté, mais pas agrandie (ni même rétrécie, ça va de soi), pour des questions d'esthétique. Elle possède de plus les attributs ACTIVATE pour être activée dès son ouverture et RMBTRAP qui a pour effet de court-circuiter les effets du bouton droit de la souris. En clair, tant que la

fenêtre de ce requester sera active, il sera impossible d'appeler le menu (qui appartient à la première fenêtre, je vous le rappelle). Concernant les flags IDCMP, cette fenêtre n'autorise les messages GADGETUP et GADGETDOWN, qui indiquent lorsqu'un gagdet est cliqué. Sa structure NewWindow contient par contre un pointeur essentiel sur le premier gadget du requester.

THE GADGETS

Ce premier gadget est défini dans le listing au label ReqGadgetDefs, zone de mémoire divisée en plusieurs structures Gadget, IntuiText, Border et Image, nécessaires à la mise en place du requester.

Le premier gadget, baptisé ReqGadget1, est le bouton OK qui met fin au requester. Il s'agit d'un gadget booléen, c'est-à-dire pouvant prendre un état ON ou OFF uniquement. En l'occurence, lorsqu'il sera OFF, il faudra fermer le requester et retourner à la boucle principale MainLoop.

Ce premier gadget pointe sur le second, baptisé ReqGadget2. Il s'agit encore d'un gadget booléen dans lequel figure le texte "ca flashe !". Lorsqu'il sera cliqué, la routine DoRequest se chargera de faire clignoter l'écran à l'aide de la fonction DisplayBeep() de la librairie Intuition.

ReqGadget2 pointe sur ReqGadget3, qui n'est autre qu'un string-gadget dans lequel l'utilisateur (jusqu'à preuve du contraire, vous) pourra écrire tout ce qui lui passe par la tête, avec toutefois une limite de 33 caractères de long. Il convient ici de noter qu'Intuition n'envoie de message concernant un string-gadget que lorsque l'entrée de texte a été confirmée par la touche Return. En recevant un tel message, DoRequest bâtira un second requester, du type SimpeRequester celui-là, qu'elle affichera à l'aide de la fonction AutoRequest() d'Intuition. Un SimpleRequester ne peut comporter qu'un texte et deux boutons (en général, OK et CANCEL). Le fameux "Please insert volume xxxx in any drive" est un excellent exemple de SimpleRequester.

ReqGadget3 pointe, on commence à en avoir l'habitude, sur ReqGadget4, le gadget proportionnel, qu'on appelle également parfois "ascenseur". Disposé horizontalement, il n'est d'aucune utilité propre dans notre requester, sinon de montrer comment on met en place un gadget proportionnel. Quant à le gérer, Intuition ne nous facilite pas la tâche : il convient en effet de tester en permanence, donc au sein d'un boucle, la position de l'ascenseur (en lisant ses coordonnées dans sa structure de définition) et, par soustraction de ses anciennes coordonnées que l'on aura pris soin de sauvegarder quelque part, de déterminer s'il a été bougé ou non. Bref, tout le travail est à faire, ce qui peut prendre une place énorme et explique que ce ne soit point inclus dans notre programme.

Finalement, ReqGadget4 pointe sur ReqGadget5, qui terminera la liste des gadgets du requester. Il s'agit d'un gadget de type booléen (encore), dans lequel figure un texte écrit avec la fonte Emerald 17 (figurant sur la disquette Workbench 1.3) qui a normalement été chargée lors de l'ouverture de l'écran (dans le cas contraire, aucun problème, le texte apparaîtra avec la fonte normale). Ce gadget comporte le flag GADGHNONE, qui spécifie à Intuition qu'elle ne doit rien en faire de particulier s'il est cliqué. De fait, nous n'avons même pas à nous en préoccuper.

CHANGEMENT D'ADRESSE

Mais maintenant, comment DoRequest va-t-elle déterminer quel gadget du requester a été cliqué ? La réponse figure dans MsgAddress, que nous avons évoqué plus haut.

Le principe est toujours le même : appel de la fonction GetMsg() d'Exec pour déterminer s'il est survenu un évènement quelconque à la fenêtre contenant le requester. Si rien ne se passe, on boucle sur le label DoRequestLoop. Dans le cas contraire, on commence par répondre au message, histoire qu'Exec sache qu'on l'a bien reçu et puisse le virer se sa liste des messages à nous envoyer. Ceci n'est absolument pas obligatoire, mais c'est beaucoup plus propre. De plus, on aura l'air malin devant le guru si une future version de l'Amiga exige absolument que l'on y réponde, à ce message. On utilise pour ce faire la fonction ReplyMsg() d'Exec, qui attend pour unique paramètre le pointeur sur le message reçu dans le registre a1.

C'est là qu'intervient MsgAddress. On y trouve l'adresse d'une structure décrivant plus précisément le message envoyé. Dans le cas qui nous intéresse, on y trouvera, à l'adresse MsgAddress+$26, le numéro d'identification du gadget cliqué. Ce numéro n'est autre que celui que nous avons nous-même défini dans les différentes structures Gadget (ReqGadget1 à ReqGadget5), voyez Commodore Revue numéro 15. En langage Intuition, ce numéro s'appelle GadgetID. Je me suis amusé, totalement arbitrairement, à donner des numéros d'identification compris entre 61 et 65, mais rien n'y oblige. Donnez les numéros que vous voulez, sans oublier le complémentaire.

Donc, une fois que DoRequest aura déterminé le GadgetID du gadget cliqué, elle n'aura plus qu'à appeler la routine correspondante, à savoir DoReqGad1), DoReqGad2 ou DoReqGad3 (le string-gadget), puisque ce sont les seuls dont nous nous occupons. C'est aussi simple que ça.

DEFINITIONS

La suite du listing, vous la connaissez : elle regroupe les différentes structures NewScreen, NewWindow, Gadget, etc. nécessaires à tous les éléments utilisés. On y trouve également des adresses utiles, comme IntBase ou ScrHandle, ainsi que les noms des différentes librairies utilisées (intuition.library, graphics.library et diskfont.library).

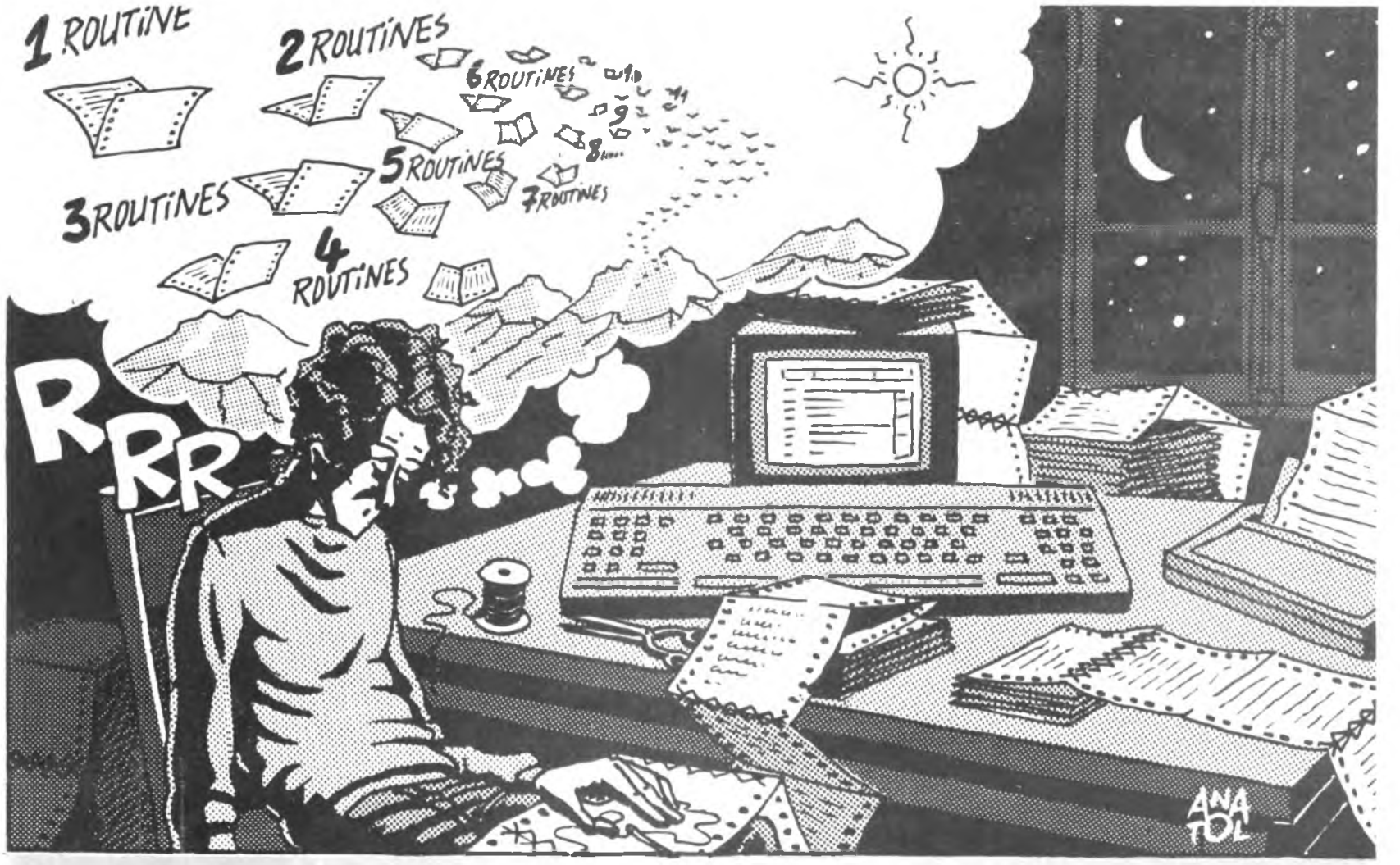
En tout, ce sont quelques 558 lignes de code source qui, une fois assemblées, produiront un programme exécutable de 1884 octets... Impressionnant, non ?

PUB

Depuis quelques temps, j'ai changé de logiciel assembleur, le Devpac Amiga version 2 de la société HiSoft m'apparaissant comme le meilleur environnement de développement en assembleur que l'on puisse trouver sur Amiga. Il possède, parmi beaucoup d'autres avantages, la possibilité de produire du code directement exécutable, sans passer par un phase de linkage intermédiaire. Ceux qui utilisent le K-Seka de la société Kuma Computers peuvent utiliser directement le programme tel qu'il est imprimé ; les autres, ceux qui préfèrent des assembleurs "pro" comme le Megamax ou celui inclus dans les compilateurs C Lattice et Aztec, devront remplacer le label "Start" par "main" (en minuscules) ainsi que le "Main" déjà existant par au autre de leur choix (pourquoi pas "Start" ?). Ils devront ensuite linker le code objet produit avec la librarie "AmigaLib.o" livrée avec ces assembleurs pour obtenir un fichier exécutable (aussi bien sous Workbench que sous CLI d'ailleurs) d'une taille en octets légèrement supérieure à celle indiquée plus haut.

Voilà, cette fois-ci, c'est tout, on se retrouve le mois prochain pour, comme promis, une brève mais suffisante explication de ce qu'il faut faire pour rendre un programme exécutable depuis le Workbench. Ciao.

Max



ViewPort

EAGLE'S RIDER

Pour votre plus grand plaisir, la fin de la routine de ciel étoilé façon StarWars par les gens de chez Microïds.

Une fois le tout tapé, il ne vous reste qu'à assembler et à exécuter. Utilisez le joystick pour contrôler le champ d'étoiles.

SECTORISE

C'est au tour de Loriciel de dévoiler l'un de ses secrets. Il ne s'agit pas d'une routine utilisée dans un jeu quelconque, mais d'un utilitaire permettant d'écrire n'importe quel fichier AmigaDOS (programme exécutable, image, musique, etc.) sur une suite de secteurs de la disquette.

Les avantages sont évidents : le chargement par lecture directe de secteurs est de toute manière beaucoup plus rapide que par l'appel de fonctions Dos évoluées, dont lourdes. Ce critère de rapidité de chargement est l'un des points auxquels Loriciel s'attache tout particulièremlent lors de la réalisation de ses jeux.

Le programme sectorise.c a été écrit par Jean-Pierre Vitulli, programmeur professionnel chez Loriciel. Toutes les explications quant à son utilisation sont incluses. Il vous donnera en outre un exemple d'utilisation du trackdisk.device en C.

Ce programme est écrit pour le Lattice version 4.0 ou supérieure. Pour compiler, utilisez le Script cc, et pour linker, le Script bl. Lors de la compilation, ignorez les messages d'avertissement ("pointers do not point to same object").

Fichier Script cc :

```
.key file
.bra {
.ket }
lc -csuw -d -v -r -b {file}
```

Fichier Script bl :

```
blink with with
```

Fichier de linkage with :

```
FROM LIB:c.o
sectorise.o
LIBRARY LIB:lc.lib,LIB:amiga.lib
NODEBUG
SMALLCODE
SMALLDATA
TO sector
VERBOSE
```

Programme sectorise.c :

```
/*************************************************************
Exemple d'utilisation du trackdisk.device en C
pour Commodore Revue

Ce programme lit un fichier (sur un autre drive ou en ram:)
et l'installe sur une suite de secteurs d'une disquette
formatée en DF0:
Il s'utilise à partir du CLI en lui donnant sur la ligne de
commande le path et nom du fichier et le numéro du 1er secteur à
utiliser ( 0-1759 )

Ex: sector DF1:toto 50 place le fichier toto sur la disquette
se trouvant dans DF1: à partir du secteur 50

Compilé avec lattice 4.00
jp Vitulli LORICIEL sept 1989
*************************************************************/
/*****************************************************/
/* includes standards */
/* pas tous utiles mieux vaut plus que pas assez */
#include "exec/types.h"
#include "exec/libraries.h"
#include "exec/memory.h"
#include "exec/devices.h"
#include "devices/trackdisk.h"
#include "libraries/dos.h"
#include "stdio.h"
/*****************************************************/
/* uniquement Lattice a partir de la version 4.00 */
/* permet l'appel des routines ROM avec passage des */
/* paramètres par les registres au lieu de la pile */
#include "proto/dos.h"
#include "proto/exec.h"
/*****************************************/
/* variables port pour trackdisk.device */
struct Port *diskport;
struct IOStdReq *diskreq;
/*****************************************/
/* variables globales */
int error; /* pour retour code erreur device */
char *buffer; /* buffer de travail */
int secteur_depart; /* No du secteur de depart extrait*/
/* de la ligne de commande */
struct FileHandle *fichier; /* handle fichier */
/*****************************************/
/* MAIN */
main(argc, argv)
int argc;
char *argv[];
{
register int origine;
/*****************************************/
/* allocation buffer de travail */
if((buffer=AllocMem(1000,MEMF_CHIP|MEMF_CLEAR))==NULL)
```

```
{   printf("impossible d'allouer le buffer\nabort\n\n");
    end_prog();
}
/*****************************************/

/* usage si mauvaise ligne de commande */
if(argc != 3)
{   printf("USAGE: %s <fichier> <secteur de départ>\n",argv[0]);
    printf("Ce programme place le fichier sur une disquette\n");
    printf("depuis <secteur de départ> (0-1759).\n");
    printf("La disquette destination DOIT être dans DF0:.\n");
    printf("Le bloc BitMap n'est pas mis à jour.\n");
    end_prog();
}

/*****************************************/
/* détermine No du secteur de départ */
secteur_depart = atoi(argv[2]);
if((secteur_depart <0) || (secteur_depart > 1759))
{   printf("secteur de départ %d invalide\n",secteur_depart);
    end_prog();
}

/*****************************************/
/* ouverture fichier source */
if(( fichier = Open(argv[1],MODE_OLDFILE)) == 0)
{   printf("impossible d'ouvrir le fichier %s",argv[1]);
    end_prog();
}

/*****************************************/
/* port communication avec trackdisk */
diskport = CreatePort(0,0);
diskreq = CreateStdIO(diskport);

/*****************************************/
/* ouverture trackdisk.device */
error = OpenDevice("trackdisk.device", 0, diskreq, 0);
if (error > 0)
{   puts("impossible ouvrir trackdisk.device");
    end_prog();
}

/*****************************************/
/* vérifie protection en écriture disk */
diskreq->io_Command = TD_PROTSTATUS;
DoIO(diskreq);
if (diskreq->io_Actual != 0)
{   puts("disque protégé en écriture ");
    end_prog();
}
/*****************************************/
/* vérification de sécurité */
printf("placer la disquette destination dans le drive df0:\n");
printf("puis taper return\n");
getchar(); /* attend un retour chariot */
origine = secteur_depart;
/***********************************************
    boucle lecture fichier/écriture disque
    si vous n'avez qu'un drive, il vaut
    mieux mettre le fichier en RAM:
    sinon il faut augmenter la taille du
    buffer et lire tout le fichier avant de
    l'écrire dans les secteurs.
    Cette version présente l'avantage de
    fonctionner quelque soit la taille du
    fichier à transférer
***********************************************/
while(Read(fichier,buffer,512)) /* lecture par 512 octets */
{   diskreq->io_Length = 512;
    diskreq->io_Data = buffer;
    diskreq->io_Command = CMD_WRITE;
    diskreq->io_Offset = (long) (secteur_depart++ * 512);
    DoIO(diskreq);
    if(secteur_depart > 1759) /* dépasse capacité disquette */
    {   printf("Dépassement du secteur 1759\n");
        printf("Plus de place sur la disquette\n");
        break;
    }
}
/*********************/
/* quelques infos utiles */
printf("Dernier secteur : %d\nTotal transfert :%d secteurs\n",
secteur_depart-1,
(secteur_depart - origine) );
printf("total : %d octets\n",(secteur_depart - origine) * 512);
/*****************************************/
/* flush buffer disque par securité */
diskreq->io_Command = CMD_UPDATE;
DoIO(diskreq);
error = diskreq->io_Error;
/*****************************************/
/* Arrêt moteur disquette */
diskreq->io_Length = 0;
diskreq->io_Command = ETD_MOTOR;
DoIO(diskreq);
/*****************************************/
/* On signale une erreur éventuelle */
if (error > 19)
{   printf("Erreur trackdisk, code %d.\n",error);
}
else
{   puts("trackdisk.device fermé sans problème.\n");
}
end_prog(); /* fin normale du programme */
}

/*****************************************/
/* dé-allocations et fermetures */
/*****************************************/
end_prog()
{
    if( fichier )
        Close(fichier); /* on ferme le fichier s'il a été ouvert */
    if( buffer )
        FreeMem(buffer,1000); /* et dé-alloue le buffer si alloué */
/* fermeture trackdisk */
    if( diskport )
        DeletePort(diskport);
    if( diskreq )
    {   CloseDevice(diskreq);
        DeleteStdIO(diskreq);
    }
    exit();
}

/*************************************************************
Evidement, les données placées sur disquette de cette façon
n'apparaissent pas au directory...
De plus, le bloc BitMap n'etant pas mis à jour, vous risquez
d'effacer ces données en inscrivant un fichier sur la
disquette.
De même, vous pouvez détruire une disquette Dos en écrivant
dans n'importe quel secteur ( ex: bloc 880 = directory root )
D'un autre côté rien ne vous empêche de stocker vos données
dans un format qui vous est propre.
Petit exercice:
comment relire ces données ?
Remplacez le CMD_WRITE par un CMD_READ
l'élement de structure io_Length doit être multiple de 512
puisque vous devez lire par secteurs entiers
io_Offset donne la position du premier secteur
exemple:
    diskreq->io_Length = 512 * 2; (taille a lire)
    diskreq->io_Data = buffer;
    diskreq->io_Command = CMD_READ;
    diskreq->io_Offset = (long) (0*512);  (secteur de départ)
    DoIO(diskreq);
On va lire 2 secteurs à partir du secteur 0, (donc le boot-bloc
en entier). Le tout est stocké dans le buffer "buffer"
Voilà, c'est tout simple ; bonne chance !!
*****************************************/
```

```
; sous-programme d'affichage d'un plan d'étoile
;---------------------------------------------

affplan:
  moveq         #9,d5 ; Nb d'étoiles à afficher = 10
affbou1:
  movem.w       (a4)+,d0-d3 ; Lit les coordonnées
  bsr           Plot
  dbf           d5,affbou1 ; Boucle
  rts ; fin du sous-programme

; Sous-programme de Calcul des coordonnées
;-----------------------------------------

coor:
  moveq         #9,d6 ; 10 étoiles
  move.l        Var2,a5 ; Coordonnées aléatoires


; debut de la boucle
;-------------------
coorb1:

 movem.w        (a4),d1-d4 ; coorx,coory,depx,depy
 move.w         d1,Tpx ; sauvegarde coorx
 move.w         d2,Tpy ; sauvegarde coory
 move.w         d3,Tpc1 ; sauvegarde depx
 move.w         d4,Tpc2 ; sauvegarde depy

; Traitement des coordonnées en X
;--------------------------------

  subi.w        #160,d1 ; soustrait 160 a x
  asr           #2,d1 ; divise coorx par 4
  asr           #5,d1 ; divise coorx par 32
  addq          #1,d1 ; +1 pour un resultat non nul
  add.w         Tpx,d1 ; ajoute le résultat a coorx
  subi.w        #160,Tpx ; soustrait 160 a coorx
  bpl           cont1 ; >=160 ->a droite de l'écran
  neg           d3 ; sinon inverse depx

cont1:
  add.w         d3,d1 ; ajoute depx à coorx
  moveq         #0,d5 ; efface le contenu de d5
  move.w        Droigau,d5 ; offset de déplacement
  add.w         d5,d1 ; ajoute l'offset à coorx

; Test de clipping
  bmi           annule1 ; si négatif sort à gauche
  cmpi          #271,d1 ; si > 271 -> sort à droite
  bgt           annule1 ; (sous le tableau de bord)


; Traitement des coordonnees en Y
;--------------------------------
traity1:
  subi.w        #100,d2 ; Procède de la même manière
  asr           #2,d2 ; pour haut et bas
  asr           #5,d2 ; que pour droite et gauche
  addq          #1,d2
  add.w         Tpy,d2
  subi.w        #100,Tpy
  bpl           conty1
```

```
  neg           d4
conty1
  add.w         d4,d2
  clr.l         d5
  move.w        Hautbas,d5
  add.w         d5,d2
  bmi           annule1
  cmpi          #199,d2
  ble           sauve1

; Création d'une nouvelle étoile
; en cas de sortie par clipping
;------------------------------
annule1
  movem.w       (a5),d1-d4 ; coorx,coory,depx,depy
  bsr           Rnd ; nombre aléatoire 0->$3f
  andi.w        #$3f,d0
  sub.w         d0,d1 ; soustrait le nb alea à coorx
  bsr           Rnd ; nb aléa 0-$1f
  andi.w        #$1f,d0
  sub.w         d0,d2 ; soustrait nb aléa à coory

; Incrémente l'offset et le pointeur
; sur la table de coord aléatoires
;---------------------------------
  addq.w        #8,Var1 ; incrémente l'offset de 8
  cmpi.w        #128,Var1 ; fin de table ?
  bne           chang1 ; non -> passe a la suite
  clr.w         Var1 ; offset sur la table
  lea           Etoile2,a5 ; initialise le pointeur
  jmp           sauve1 ; passe a la sauveagarde
chang1:
  addq.l        #8,a5 ; incremente le pointeur
; sauvegarde des nouvelles coordonnees de l'etoile
;-------------------------------------------------
sauve1:
  tst           d3 ; teste depx
  bpl           sauve11 ; depx <0 -> inverse depx
  neg           d3
sauve11:
  tst           d4 ; teste depy
  bpl           sauve12 ; depy <0 -> inverse depy
  neg           d4
sauve12:
  movem.w       d1-d4,(a4)
  lea           8(a4),a4 ; Incremente le pointeur
  dbf           d6,coorb1 ; Etoile suivante
  move.l        a5,Var2 ; sauvegarde le pointeur

  lea           Custom,a5
  move.l        ExecBase,a6

  rts ; fin du sous-programme

Rnd:
  move.w        Custom+VHPOSR,d0
  rts
; ******************************
; *        Routine Plot        *
; * Affiche un pixel à l'écran *
; * aux coordonnées d0=x, d1=y *
; ******************************
```

```
Plot:
  movem.l       d0-d7/a0-a6,-(sp)

; Calcule l'adresse réelle du pixel

  move.l        Ecr_Log,a1

  move.w        d0,d2 ; d2=d0=x
  and.w         #$fff0,d0
  lsr.w         #3,d0 ; d0=x/8
  and.w         #$000f,d2 ; d2=x mod 16

  move.w        d1,d3 ; d3=d1=y
  lsl.w         #5,d1 ; d1=d1*32
  lsl.w         #3,d3 ; d3=d3*8
  add.w         d3,d1 ; d1=y*40 (octets/ligne)

  add.w         d0,d1
  lea           0(a1,d1.w),a1 ; a1=adresse du pixel

  move.w        Cou,d0
  move.w        d0,d2
  lsl.w         #1,d0
  lea           PlotMasques,a0
  move.w        0(a0,d0.w),d0 ; Masque à appliquer
  move.w        d0,d1
  not.w         d1

  lea           PlaneSize(a1),a2
  lea           PlaneSize(a2),a3
  lea           PlaneSize(a3),a4
  lea           PlaneSize(a4),a5
   lea          PlotsTable,a0
   lsl.l        #2,d2
   add.l        d2,a0
   move.l       (a0),a0
   jsr          (a0)

  movem.l       (sp)+,d0-d7/a0-a6
  lea           Custom,a5
  move.l        ExecBase,a6
  rts

Plot00:
  and.w         d1,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot01:
  or.w          d0,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot02:
  and.w         d1,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot03:
  or.w          d0,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot04:
  and.w         d1,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot05:
  or.w          d0,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot06:
  and.w         d1,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot07:
  or.w          d0,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  and.w         d1,(a5)
  rts

Plot08:
  and.w         d1,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot09:
  or.w          d0,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot10:
  and.w         d1,(a1)
  or.w          d0,(a2)
```

```
  and.w         d1,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot11:
  or.w          d0,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot12:
  and.w         d1,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot13:
  or.w          d0,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot14:
  and.w         d1,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot15:
  or.w          d0,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  and.w         d1,(a5)
  rts

Plot16:
  and.w         d1,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot17:
  or.w          d0,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot18:
  and.w         d1,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot19:
  or.w          d0,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot20:
  and.w         d1,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot21:
  or.w          d0,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot22:
  and.w         d1,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot23:
  or.w          d0,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  and.w         d1,(a4)
  or.w          d0,(a5)
  rts

Plot24:
  and.w         d1,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts

Plot25:
  or.w          d0,(a1)
  and.w         d1,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
```

```
Plot26:
  and.w         d1,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
Plot27:
  or.w          d0,(a1)
  or.w          d0,(a2)
  and.w         d1,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
Plot28:
  and.w         d1,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
Plot29:
  or.w          d0,(a1)
  and.w         d1,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
Plot30:
  and.w         d1,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
Plot31:
  or.w          d0,(a1)
  or.w          d0,(a2)
  or.w          d0,(a3)
  or.w          d0,(a4)
  or.w          d0,(a5)
  rts
*************
*           *
*   DATA    *
*           *
*************
Ecr_Phys        dc.l        0
Ecr_Log         dc.l        0
CLadr           dc.l        0
Ecrbuf1         dc.l        0
Ecrbuf2         dc.l        0
Ecrbuf3         dc.l        0

Tpy             dc.w        0
Tpx             dc.w        0
Tpc1            dc.w        0
Tpc2            dc.w        0
Of1f            dc.w        0
Of2f            dc.w        20
Of3f            dc.w        40
Cou             dc.w        0
```

```
Joy_x           dc.w        0
Joy_y           dc.w        0

Hautbas         dc.w        0
Droigau         dc.w        0
Hautbast        dc.w        0
Droigaut        dc.w        0

Var1            dc.w        0
Var2            dc.l        0
Var3            dc.w        0

Etoile:
  dc.w          30,30,1,1,102,55,1,1
  dc.w          240,58,1,1,172,75,1,1
  dc.w          98,105,1,1,217,117,1,1
  dc.w          80,140,1,1,155,143,1,1
  dc.w          220,160,1,1,105,180,1,1

Etoile3:
  dc.w          120,30,1,1,160,25,1,1
  dc.w          20,60,1,1,80,80,1,1
  dc.w          217,80,1,1,22,90,1,1
  dc.w          155,105,1,1,298,120,1,1
  dc.w          50,160,1,1,155,157,1,1

Etoile2:
  dc.w          90,60,3,2,230,140,2,3
  dc.w          160,180,2,3,300-48,60,3,2
  dc.w          160,100,2,3,300-48,140,3,2
  dc.w          160,140,2,3,230,100,2,3
  dc.w          90,100,3,2,230,180,2,3
dc.w            90,140,3,2,230,690,2,3
  dc.w          160,60,2,3,300-48,180,3,2
  dc.w          90,180,3,2,300-48,100,3,2

Palette:
  dc.w          $000,$fff,$acf,$8ac,$777,$468,$a00
  dc.w          $aaa,$0f0,$0f0,$0f0,$0f0,$0f0,$0f0
  dc.w          $0f0,$fff,$0f0,$0f0,$0f0,$0f0,$0f0
  dc.w          $0f0,$0f0,$0f0,$0f0,$0f0,$0f0,$0f0
  dc.w          $0f0,$0f0,$0f0,$0f0

PlotMasques:
  dc.w          $8000,$4000,$2000,$1000
  dc.w          $0800,$0400,$0200,$0100
  dc.w          $0080,$0040,$0020,$0010
  dc.w          $0008,$0004,$0002,$0001

PlotsTable:
  dc.l          Plot00,Plot01,Plot02,Plot03
  dc.l          Plot04,Plot05,Plot06,Plot07
  dc.l          Plot08,Plot09,Plot10,Plot11
  dc.l          Plot12,Plot13,Plot14,Plot15
  dc.l          Plot16,Plot17,Plot18,Plot19
  dc.l          Plot20,Plot21,Plot22,Plot23
  dc.l          Plot24,Plot25,Plot26,Plot27
  dc.l          Plot28,Plot29,Plot30,Plot31

VideEcr:
  dc.l          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

GfxName:
  dc.b          "graphics.library",0
  even
```

Requester

Suite à la question sur le format des fichiers IFF ILBM, un lecteur nous a demandé celui des fichiers ANIM utilisés par DeluxePaint III ou Sculpt Animate 4D. Dans l'impossibilité de lui répondre, nous adressons un appel au peuple : si quelqu'un parmi vous connaît la réponse, qu'il nous l'envoie pour en faire profiter les autres.

Question : *Je recherche un assembleur pour mes débuts en programmation, lequel pourriez-vous me conseiller ?*

Regis Versans, Versailles

Réponse : Il existe plusieurs assembleurs sur Amiga, les plus connus étant les ancêtres K-Seka et Metacomco, Profimat Amiga et Devpac Amiga version 2. Les meilleurs sont à notre avis K-Seka et Devpac, qui permettent l'édition, l'assemblage et le débogage en un seul programme. Entre les deux mon coeur balance, mais pour un débutant, le Devpac est sans doute plus appoprié, K-Seka restant réservé à une certaine élite de "bidouilleurs". Devpac est édité par la société britannique HiSoft.

Question *: Je me suis procuré divers livres concernant la programmation sur Amiga, et tous parlent d'utiliser la fonction GetCC() d'Exec, qui sert à récupérer le registre SR, plutôt que l'instruction MOVE SR,Destination. Seulement voilà, j'ai désassemblé la fonction GetCC(), elle se résume en une seule instruction, qui est (je vous le donne en mille) : MOVE SR,D0. Alors franchement, quel est l'intérêt ?*

Lilian Margerie, Rueil-Malmaison

Réponse : Je parie que vous travaillez avec un 68000... Le problème est le suivant : l'instruction MOVE SR,<Destination> provoque une exception sur un 68020 ou 68030, et donc un Guru Meditation. La fonction GetCC() permet de récupérer le registre SR de manière compatible 68000/020/030 ; lors de l'initialisation d'Exec (au démarrage à chaud ou à froid), le système va tester le type de processeur présent et modifier GetCC() en fonction du résultat de ce test. Avec un 68000, la fonction reste en effet MOVE SR,D0. Devinette : quelle(s) instruction(s) est (sont) utilisée(s) avec un 68020/030 ?

Question *: Peut-on, et si oui comment, adresser directement les co-processeurs en Basic (Amiga ou GfA) ?*

Antoine Defons, Paris

Réponse : Bien sûr, et très facilement, même. Les deux programmes suivants, l'un pour l'AmigaBasic, l'autre pour le GfA, illustrent la manière de procéder, même s'ils ne font rien de particulièrement visible à l'écran.

Ok et Cancel

```
'
' Accès aux co-processeurs en AmigaBasic
'
LIBRARY "exec.bmap"
CALL Forbid                     ' Interdit le multi-tâches
Custom&=&HDFF000                ' Adresse de base des
                                co-processeurs
Dmacon&=Custom&+&H96            ' Registre de contrôle DMA
'
PRINT "Appuyez sur une touche pour désactiver le Copper"
PRINT "puis sur une autre pour le ré-activer."
'
A$=INPUT$(1)
POKEW Dmacon&,&H0080            ' Copper OFF
A$=INPUT$(1)
POKEW Dmacon&,&H8080            ' Copper ON
'
CALL Permit                     ' Autorise le multi-tâches
LIBRARY CLOSE                   ' Ferme la librarie
END

' Accès aux co-processeurs en GfABasic
'
~Forbid()
'
custom%=&HDFF000
ABSOLUTE dmacon&=custom%+&H96
'
PRINT "Appuyez sur une touche pour désactiver le Copper"
PRINT "puis sur une autre pour le ré-activer."
'
A$=INPUT$(1)
WORD{dmacon&}=&H0080
A$=INPUT$(1)
WORD{dmacon&}=&H8080
'
~Permit()
EDIT
```

COURRIER

62

ST heu!

Bonjour je ne sais si vous passerez ma lettre car elle ne concerne pas vraiment l'Amiga. Néanmoins, je crois de mon devoir d'avertir les prochains acheteurs de micro 16/32 bits. Voilà, j'ai fait l'acquisition d'un Atari STE, le remplaçant du ST. Si je l'ai acheté, c'est qu'il me paraissait aussi bien qu'un Amiga 500 (qualités graphiques) et que je possèdais déjà un Atari ST, donc une bibliothèque de logiciels bien fournie. Je mets en garde vos lecteurs qui ne possèdent pas encore d'Amiga contre la publicité trop avantageuse qui est faite autour de cette nouvelle machine dite compatible. Plus d'une quarantaine de jeux ne fonctionnent pas sur le STE (en tout cas sur le mien!). Autant vous dire que j'en suis malade de l'avoir acheté...

Paul Farrey, Tarbes

OPERATION UPGRADE

L'opération Upgrade consistant à muter un Amiga 2000A en Amiga 2000B est payante et limitée. L'offre restera effective jusqu'à fin décembre pour la somme de 3490 F.

Le seul et réel signe extérieur permettant la reconnaisance entre un Amiga 2000A et un Amiga 2000B est la sortie composite monochrome qui se situe à l'arrière de la machine juste à côté des sorties audio. Tout ce qui vous a été affirmé précédemment n'est que pure fantasie. Autant pour la rédaction qui n'avait pas vu les monstrueux bugs dans le courrier du mois dernier.

TABLEAU DES DIFFÉRENCES
ENTRE AMIGA 2000A ET AMIGA 2000B

	A2000A	A2000B
Mémoire vive	512 ko mère 512 ko slot	1 Mo mère (1 Mo vidéo)
Circuit d'animation	AGNUS	SUPER AGNUS
Circuit d'affichage	DENISE	DENISE
Connecteur vidéo monochrome	NON	OUI
Slot CPU (par les 512 ko)	OCCUPE	LIBRE
Slot vidéo	NORMAL	ETENDU
Jumper Configuration 1 Mo	ABSENT	J500/J101/J102
Jumper Configuration A2010	J36	J301
Possibilité d'utiliser Super AGNUS et Super DENISE	NON	OUI
Circuit US à changer SI CNT-01 VERS CNT-02 En A2090/A2090A	OUI	NON (n'existe plus)
Kickstart d'origine	1.2	1.3
Options périphériques : Cartes Flicker Fixer	NON	OUI
Coexistence 68000/60820-68030	NON	OUI