Comment écrire du code portable pour Amiga
(AmigaOS 3.x / MorphOS / AmigaOS4 / AROS)

Par Mathias PARNAUDEAU - mathias.p@wanadoo.fr
12 février 2006

Afin d'aider les développeurs à prendre en compte les évolutions de ces dernières années, ce document se propose de donner des informations et conseils pour développer du code standard qui pourra se compiler sur les différents systèmes de la famille Amiga, dont on distingue : AmigaOS 3.x, MorphOS, AmigaOS 4, AROS.

L'API de base est la même et plusieurs bibliothèques se sont également imposées : AHI pour la gestion du son, CGFX (supporté aussi par Picasso96), SDL, TTEngine, ... Mais pour le développeur, ce n'est malheureusement pas toujours le cas : on note par exemple d'importantes différences entre ReAction et MUI, Poseidon et Sirion, ...
D'autre part, chaque OS propose de nouvelles fonctions ou plus largement de nouveaux mécanismes, comme AmigaInput sous OS4 pour la gestion des périphériques d'entrée ou Reggae sous MorphOS pour le multimedia.

Avant d'aborder les différences d'API, d'includes et de bibliothèques, voici quelques conseils plus larges :
- réaliser son projet de façon modulaire : le code peut être correct, s'il est confus, il sera difficile de le faire évoluer et de faire intervenir des développeurs extérieurs
- encapsuler les différences majeures comme l'accès au port USB ou autres périphériques : ne pas faire d'appel direct à des fonctionnalités spécifiques
- respecter les conventions du C et éviter les spécificités de chaque compilateur (le meilleur moyen est de maintenir son programme avec plusieurs compilateurs)
- utiliser les avantages de chacun des OS et de leurs outils (splint, Grim Reaper, GDB, mpatrol, ...), la portabilité ne représente pas que des contraintes !
- utiliser plusieurs compilateurs pour mettre en évidence des erreurs et warnings à la compilation, chacun étant plus ou moins permissif

La suite du document décrit de manière plus précise les différences entre les SDK des systèmes et comment gérer au mieux un environnement unique avec des pratiques communes pour écrire et maintenir du code portable en fournissant le moins d'effort possible.

On considère ici que le lecteur connaît déjà les bases de la programmation et possède au moins une petite expérience du développement sur Amiga avec le langage C.

Enfin, une liste de ressources est disponible en annexe.

1. Includes

Les includes contiennent ce que le système met à disposition du développeur : structures, constantes, prototypes des fonctions, ... Certaines includes sont propres à chaque compilateurs ou à chaque système : pragma, inline, ppcinline, ... Ces dernières ne doivent pas être appelées directement depuis un programme : à la place on inclut uniquement les fichiers proto qui eux-mêmes se chargeront d'appeler les includes spécifiques adaptées.

1.1. OS4, un cas à part

Sous OS4, le type de base des bibliothèques (dans les fichiers proto) sont désormais quasiment toutes des "struct Library" au lieu du nom de la bibliothèque suffixée par Base (comme IntuitionBase) ou autres cas particuliers (SysBase pour Exec, GfxBase pour Graphics, ...). Suivant la révision de l'OS, les types ont parfois changé et cela a amené une complexité conséquente. Pour une question de compatibilité et si l'on veut conserver le type d'origine (Library ou XxxBase), il faut compiler avec le define __USE_BASETYPE__.

D'autre part, les inlines sont rangées dans le répertoire "inline4" et grâce à des macros, elles font le lien entre le prototype des fonctions et leurs interfaces, une nouvelle notion apportée par OS4 et détaillée dans une partie qui suit.

1.2. Génération des includes

L'outil central et historique se nomme fd2pragma et il est capable de générer chaque type d'includes pour une bibliothèque donnée à partir de son fichier de description FD.
Pour MorphOS, il existe un script en Perl, cvinclude.pl pour générer ces fichiers et un autre (UpdateMosIncForVbcc.pl) pour patcher les includes afin de mieux supporter VBCC (voir sur http://developer.morphosppc.com/index.php, partie " Files ").
Avec OS4, fd2pragma est uniquement utilisé pour générer un fichier XML, sorte de super fichier FD qui permet au nouvel outil idltool de créer les includes mais aussi un squelette de bibliothèque par exemple.

1.3. Organisation des includes avec plusieurs compilateurs

Quand on développe sous plusieurs OS et/ou avec plusieurs compilateurs, la gestion des répertoires d'includes peut devenir un véritable casse-tête. En effet, certaines sont propres au compilateur et n'ont même rien à voir avec l'API système de l'AmigaOS.

D'autres sont communes à tous les OS et compilateurs comme SDL, OpenGL, classes MUI, ... Il est souhaitable de les réunir dans un même répertoire à indiquer au compilateur, au niveau de son paramétrage ou lors de la compilation avec l'option -I (pour GCC et VBCC). Sous OS4, l'organisation du SDK prend déjà en compte un tel répertoire, dont le chemin choisi est " SDK:Local/Common/Includes ".

Attention, ces includes communes sont souvent liées à des bibliothèques statiques qui elles, sont propres à l'OS mais leur format peut aussi dépendre du compilateur. VBCC gère cela en spécifiant des cibles différentes dans lesquelles on retrouve un répertoire include et un autre lib.

2. Spécificités des OS

Rendre son code portable n'empêche pas d'avoir à prendre en compte quelques particularités. Nous indiquons ici comment faire.

2.1. Defines spécifiques

Les particularités de chaque OS ou compilateurs seront abordées dans la suite mais comme certaines parties du code seront spécifiques à l'un ou l'autre voyons quels " define " existent pour du code conditionnel traité par le compilateur.

OS4 : __amigaos4__
MorphOS : __MORPHOS__
AROS : __AROS__

VBCC : __VBCC__
GCC : __GNUC__
SASC : __SASC

Attention : Pour choisir quel define appliquer, il est nécessaire de prendre un peu de recul pour traiter ou exclure un cas particulier. Par exemple, si on porte une application audio d'AmigaOS 3.x vers MorphOS et qu'on veut exploiter le hardware dans un cas et AHI dans l'autre, pour ce dernier on n'utilisera pas "#ifdef __MORPHOS__" mais quelque chose du genre "#ifdef USE_AHI". Ainsi, bien que développé dans l'optique du portage sur MorphOS, le code sera déjà prêt à être recompilé sous OS4.

2.2. Interfaces OS4

Il s'agit d'un nouveau concept qui permet de spécifier explicitement la bibliothèque d'une fonction utilisée. Bien que son mécanisme permette apparemment une utilisation évoluée avec plusieurs cibles possibles, l'intérêt actuel réside principalement dans la lisibilité du code, les fonctions étant appelées avec une syntaxe rappelant le C++ : IExec->OpenLibrary(...).
Pour du code portable, il ne faut pas que les interfaces viennent interférer, c'est pourquoi, à moins de vouloir bénéficier de particularités d'OS4, on se passera de ce mécanisme.
Pour en bénéficier implicitement en indiquant qu'on utilisera la notation classique (sans que les appels de fonctions ne soient préfixés par l'interface), il faut compiler avec l'option -D__USE_INLINE__. Sous OS4, ça ne dispense pas de déclarer les interfaces qui restent utilisées bien qu'on n'utilise pas leur notation. A noter qu'avec l'option -lauto, les bibliothèques de base (exec, dos, intuition, ...) sont ouvertes et leur interface est obtenue également.

Pour gérer une interface, par exemple concernant MUI, on effectue manuellement la déclaration au début du source :

```
Struct MUIMasterIFace *IMUIMaster;
```

Ensuite l'interface est obtenue ainsi lors de l'initialisation du programme :

```
#ifdef __amigaos4__
IMUIMaster = (struct MUIMasterIFace *)GetInterface(MUIMasterBase,
"main", 1, NULL);
#endif
```

... et elle est retirée en fin de programme :

```
#ifdef __amigaos4__
if (IMUIMaster) {
   DropInterface((struct Interface *)IMUIMaster);
}
#endif
```

2.3. Fonctions amiga.lib, DoSuperNew

Au fur et à mesure du temps, l'amiga.lib s'est enrichie en fonctions, dépendant à la fois du compilateur et du système. Bien que pratique, elle ne représente pas une base commune et peut être amenée à changer à nouveau.

D'ailleurs, sous OS4 des fonctions de l'amiga.lib ont été déplacées dans les bibliothèques auxquelles elles pouvaient logiquement se rattacher. Par exemple, DoMethod a été migrée dans Intuition sous le nom IDoMethod. C'est pourquoi la recompilation sous cet OS nécessite d'utiliser le define suivant :
-DDoMethod=IDoMethod

La fonction DoSuperNew est un cas un peu spécial. Cette fonction existe dans l'amiga.lib de MorphOS mais est par exemple absente sous OS4 ou sous AmigaOS 3.x (SAS/C, VBCC, GCC). Elle sert à instancier un nouvel objet d'une classe (custom class) notamment dans le constructeur d'un classe MUI.

Il existe plusieurs façon de définir cette fonction mais aucune n'offre la garantie de fonctionner partout ! La manière la plus simple trouvée (utilisant les includes SDI, voir plus bas) pour répondre aux différents cas semble prendre la forme :

```
#ifndef __MORPHOS__
#ifdef __amigaos4__
Object * STDARGS VARARGS68K DoSuperNew(struct IClass *cl, APTR obj, ...)
{
   Object *rc;
   VA_LIST args;

   VA_START(args, obj);
   rc = (Object *)DoSuperMethod(cl, obj, OM_NEW, VA_ARG(args, ULONG),
NULL);
   VA_END(args);
```

```
    return rc;
}
#else
APTR STDARGS DoSuperNew(struct IClass *cl, APTR obj, ULONG tag1, ...)
{
    return ((APTR)DoSuperMethod(cl, obj, OM_NEW, &tag1, NULL));
}
#endif
#endif
```

Remarque : Sous MorphOS, il semble qu'en liant avec aboxstubs ou en utilisant -DUSE_INLINE_STDARG, cela utilise la déclaration de DoSuperNew avec la macro suivante :

```
#define DoSuperNew(MyClass, MyObject, ...) \
    ({ ULONG _tags[] = { __VA_ARGS__ }; \
    struct opSet MyopSet; \
    MyopSet.MethodID = OM_NEW; \
    MyopSet.ops_AttrList = (struct TagItem*) _tags; \
    MyopSet.ops_GInfo = NULL; \
    DoSuperMethodA((MyClass), (MyObject), (APTR) &MyopSet); \
    })
```

Ainsi le mot-clé VARARGS68K n'est pas utilisé, lui qui n'est pas forcément reconnu par toutes les versions de GCC.

2.4. Création de tâches

Avec le passage au PowerPC, certains éléments plus ou moins liés à l'architecture matérielle ont dû évoluer. C'est le cas des tâches qui étaient en plus reconnues pour faire preuve de lacunes dans leur gestion peu moderne. Sous OS4, aucune modification n'est à apporter contrairement à MorphOS où l'on doit ajouter le tag suivant :

```
#ifdef __MORPHOS__
    NP_CodeType,    MACHINE_PPC,
#endif
```

MorphOS propose d'autres facilités telles que la création et libération automatique du port de message ou le passage de données utilisateur grâce au tag NP_UserData (fonctionnalité disponible aussi dans AROS).

2.5. Mémoire

L'évolution de nos systèmes met en avant un des points demandant à être réformé en profondeur.

Les allocations mémoire sont caractérisées d'après les flags de préfixe MEMF_. Sous OS4, il devient interdit d'utiliser MEMF_PUBLIC à remplacer par le type MEMF_SHARED qui est spécifique à OS4 mais n'est de toute façon à utiliser que dans certains cas très précis. On ne doit employer que MEMF_ANY ou MEMF_PRIVATE pour les allocations classiques répondant aux besoins internes au programme.

MorphOS propose lui aussi de nouveaux flags particuliers comme MEMF_SWAP (sans doute équivalent au MEMF_VIRTUAL d'OS4) pour la mémoire virtuelle faisant appel au disque dur.

3. Spécificités des compilateurs

3.1. Macros SDI_headers

A l'origine, chaque compilateur a proposé des mots-clés spécifiques (comme reg, __asm, __saveds, ...), chacun employant une syntaxe particulière. C'est pourquoi, dans un souci d'homogénéisation, Dirk Stoecker avait commencé à créer des macros au sein d'un fichier "SDI_compiler.h". Cela a évolué, avec l'aide de Jens Langner, pour donner un ensemble d'includes réunies sous le nom générique de SDI_headers :

- SDI_compiler : registres, passage de paramètres, ...
- SDI_hook : macros pour hooks et dispatchers
- SDI_lib :
- SDI_stdarg : varargs, etc.

Ainsi, toutes les particularités des compilateurs et systèmes sont prises en compte.

3.2. Cas des hooks

Un hook ("crochet" en français) est un mécanisme qui permet d'associer une fonction à un événement, ne sachant pas quand celui va se déclencher : pour un player de CD, hook sur l'éjection du CD par exemple ou très utilisé dans MUI, par exemple lors de l'ajout d'un élément dans une liste, ...

Le mécanisme mis en jeu est fortement lié aux registres 680x0. L'include SDI_hook apporte une solution multi-plateforme et extrêmement simple :

```
HOOKPROTO(nom_de_la_fonction)
{
   code de la fonction
}

MakeHook(nom_du_hook, nom_de_la_fonction);
```

Le hook est ensuite appelé au moyen de la fonction CallHook (avec "&nom_du_hook") ou équivalent MUI (méthode MUIM_CallHook). MakeStaticHook s'utilise à la place de MakeHook dans le cas où le hook n'a besoin d'être visible que par les fonctions du module (fichier C).

HookEntry (présent dans amiga.lib) était une manière de faire commune à AmigaOS et MorphOS mais a été retirée de OS4.

Dans la même lignée que les hooks et concernant les classes BOOPSI et MUI, il y a le dispatcher dont l'écriture est elle aussi simplifiée :

```
DISPATCHERPROTO(nom_du_dispatcher)
{
   code du dispatcher : switch avec les méthodes de la classe
}
```

En plus de cette macro, une autre nommée ENTRY intervient lors de l'affectation du dispatcher, au moment de la création de la classe :

cl_disko = MUI_CreateCustomClass(NULL, MUIC_Group, NULL, sizeof(struct DisKoData), ENTRY(DisKoDispatcher));

3.3. Fonctions à nombre d'arguments variable

Là encore, le groupe de fichiers SDI_headers propose une solution commune qui encapsule les mots-clés permettant la création de fonctions, telles printf, pouvant recevoir un nombre indéfini de paramètres. On se reportera au cas de DoSuperNew cité plus haut.

Annexe 1 : Ressources

Références communes

VBCC
Compilateur C gratuit disponible sur http://sun.hasenbraten.de/vbcc/

MUI
Site ZeroHero pour classes MUI : http://www.zerohero.se/mui/
SDK MUI : mui38dev.lha disponible sur Aminet (http://www.aminet.net)

CD Developer
Includes, documentations (Rom Kernel Manual, ...)

Rom Kernel Reference Manual
Livres de référence, complets mais plus destinés aux programmeurs avertis

Programmation Amiga en français
Guru-meditation : http://www.guru-meditation.net
Guide Programmation AmigaOS : http://amigadev.free.fr/ProgrammationAmigaOS.pdf

CubicIDE
Environnement de développement : http://www.dietmar-eilert.net/cubic/index.htm

Et bien sûr d'autres outils et exemples sur http://www.aminet.net

MorphOS

SDK MorphOS
Disponible sur le site de support aux développeurs : http://developper.pegasosppc.com

AmigaOS 4

SDK OS4
Téléchargeable avec les mises à jour de l'OS : http://www.hyperion-entertainment.biz

BitByBit
Environnement et outils de développement : http://bitbybitsoftwaregroup.com/index.php

OS4 depot
Autres outils et exemples sur http://os4depot.net

AROS

Site officiel AROS
Informations sur AROS, manuel de développement, ... : http://www.aros.org
Site de support aux utilisateurs et développeurs

Site support
Actualités, forums, images ISO, ... sur http://www.aros-exec.org

Autres liens

http://www.monkeyhouse.eclipse.co.uk/amiga/dev.htm
http://www.utilitybase.com

Ne pas oublier de lire les autodocs qui sont les ressources à exploiter en premier !