

MANUEL D'UTILISATION DE LA MINILIB C

Version 1.00

S. Di Mercurio

Table des matières

1.Presentation.....	3
1.1.Limite de ce document.....	3
2.Fonctions intégrées à la minilib.....	4
3.Utilisation de la bibliotheque.....	5
3.1.Sous KEIL.....	5
a)Utilisation avec GCC.....	5
b)Utilisation avec ARMCC.....	5
4.Recompiler la bibliotheque.....	6
5.Options utilisables lors de la compilation de la bibliotheque.....	6
6.Description des fonctions contenues dans syscalls.c.....	7
7.Un mot au sujet des fonctions utilisant le type FILE.....	7
7.1.Un point important au sujet de ces flux.....	7
7.2.Utilisation concrete dans une application.....	8

1. Presentation

La minilib C est une bibliothèque C, basée sur le code de la newlib de redhat, mais fortement réduite en fonctionnalité et taillée dans la masse pour être la plus petite possible. En ce sens, elle n'embarque pas toutes les fonctions proposées par la bibliothèque C classique (libc): voir la chapitre « Fonctions intégrées à la minilib » pour connaître la liste des fonctions incluses.

La minilib nécessite, pour certaines fonctions (tel `printf`) un « support système », en gros, un moyen, dépendant de la plateforme et de l'application, pour réaliser sa tâche. Dans le cas du `printf`, c'est un endroit où écrire le texte (une liaison série le plus souvent). Pour être le plus souple et adaptable possible, ce support système est volontairement non codé et aboutit dans des fonctions contenues dans le fichier `syscalls.c`, non inclus dans la bibliothèque et que l'utilisateur doit intégrer dans son projet, puis l'adapter à son besoin. Pour plus d'info, se référer aux chapitres « Utilisation de la bibliothèque » et « Description des fonctions contenues dans `syscalls.c` ».

1.1. Limite de ce document

Ce document se limite à la version 1.00 de la minilib.

2. Fonctions intégrées à la minilib

Les fonctionnalités suivent le découpage existant au sein de la libc et que l'on retrouve dans les fichiers en-tête tel que `stdio.h` ou `stdlib.h`.

Pour plus d'information concernant les fonctions listées ici, se referer à la norme ISO/IEC C'99

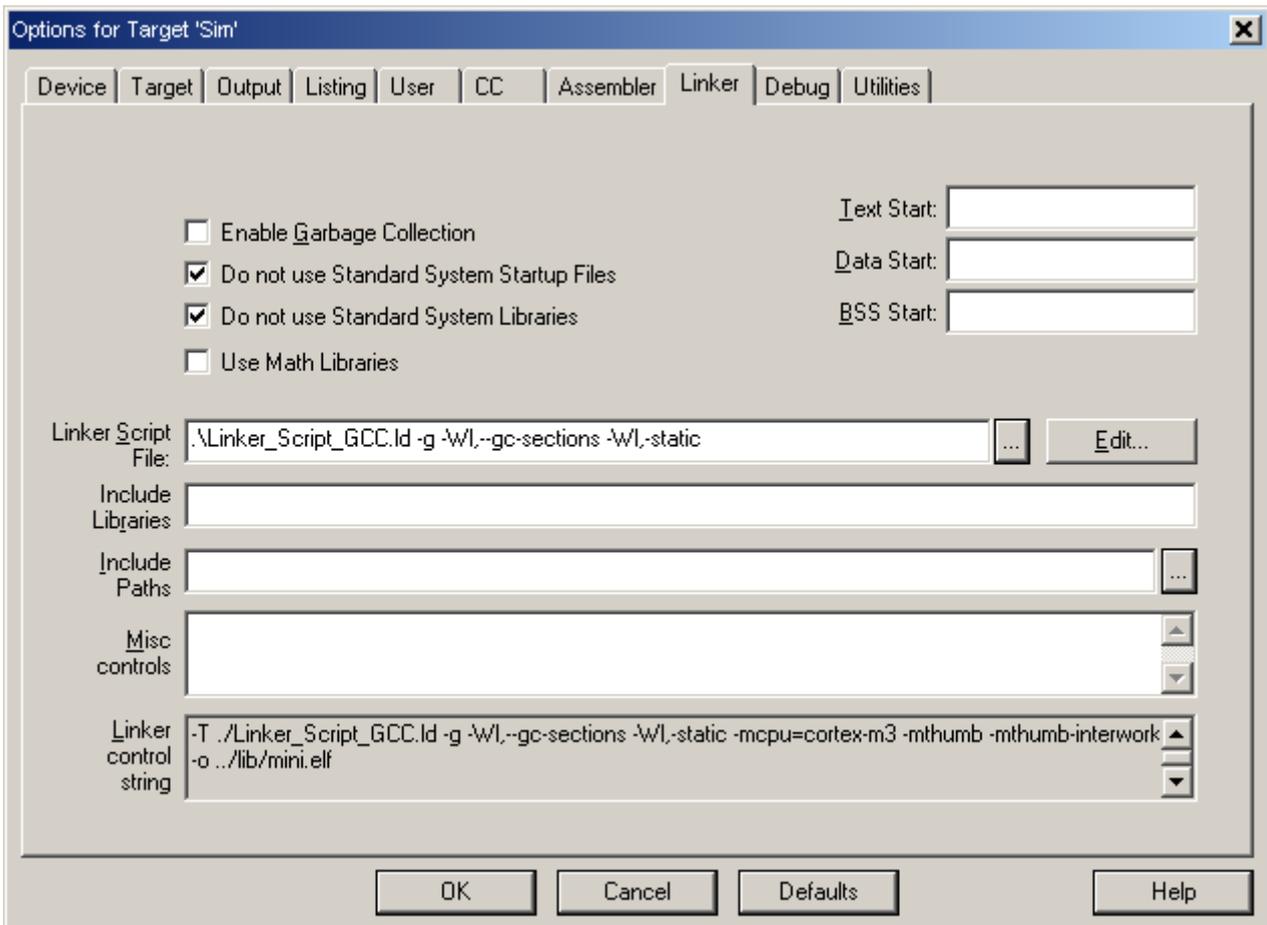
<ul style="list-style-type: none">• STDIO<ul style="list-style-type: none">• <code>printf</code>• <code>fprintf</code>• <code>sprintf</code>• <code>scanf</code>• <code>fputc</code>• <code>fgetc</code>• <code>fputs</code>• <code>fgets</code>• <code>getchar</code>• <code>putchar</code>	<ul style="list-style-type: none">• STDLIB<ul style="list-style-type: none">• <code>atoi</code>• <code>atol</code>• <code>strtol</code>• <code>div</code>• <code>ldiv</code>• <code>abs</code>• <code>exit</code>• <code>assert</code>• <code>rand</code>• <code>malloc</code>• <code>calloc</code>• <code>free</code>	<ul style="list-style-type: none">• CTYPE<ul style="list-style-type: none">• <code>isalnum</code>• <code>isalpha</code>• <code>isascii</code>• <code>isblank</code>• <code>iscntrl</code>• <code>isdigit</code>• <code>islower</code>• <code>isprint</code>• <code>ispunct</code>• <code>isspace</code>• <code>isupper</code>• <code>isxdigit</code>• <code>toascii</code>• <code>tolower</code>• <code>toupper</code>
<ul style="list-style-type: none">• STRING<ul style="list-style-type: none">• <code>bcmp</code>• <code>bcopy</code>• <code>bzero</code>• <code>index</code>• <code>memccpy</code>• <code>memchr</code>• <code>memcmp</code>• <code>memcpy</code>• <code>memmove</code>	<ul style="list-style-type: none">• STRING (suite)<ul style="list-style-type: none">• <code>memcpy</code>• <code>memsetrindex</code>• <code>strcat</code>• <code>strchr</code>• <code>strcmp</code>• <code>strcoll</code>• <code>strcpy</code>• <code>strcspn</code>• <code>strlcat</code>	<ul style="list-style-type: none">• STRING (fin)<ul style="list-style-type: none">• <code>strncpy</code>• <code>strlen</code>• <code>strlwr</code>• <code>strncat</code>• <code>strncmp</code>• <code>strncpy</code>• <code>strnlen</code>• <code>strrchr</code>• <code>strsep</code>• <code>strspn</code>• <code>strstr</code>• <code>strupr</code>

3. Utilisation de la bibliotheque

3.1. Sous KEIL

a) Utilisation avec GCC

1. Rajoutez le fichier `syscalls.c` à votre projet (le template se trouve sous `minilib/syscalls_template/syscalls.c`)
2. Adaptez les fonctions du fichier à votre besoin (voir le chapitre « Description des fonctions contenues dans `syscalls.c` » à propos des fonctions)
3. Rajoutez le chemin vers `libmini.a` dans l'onglet du linker

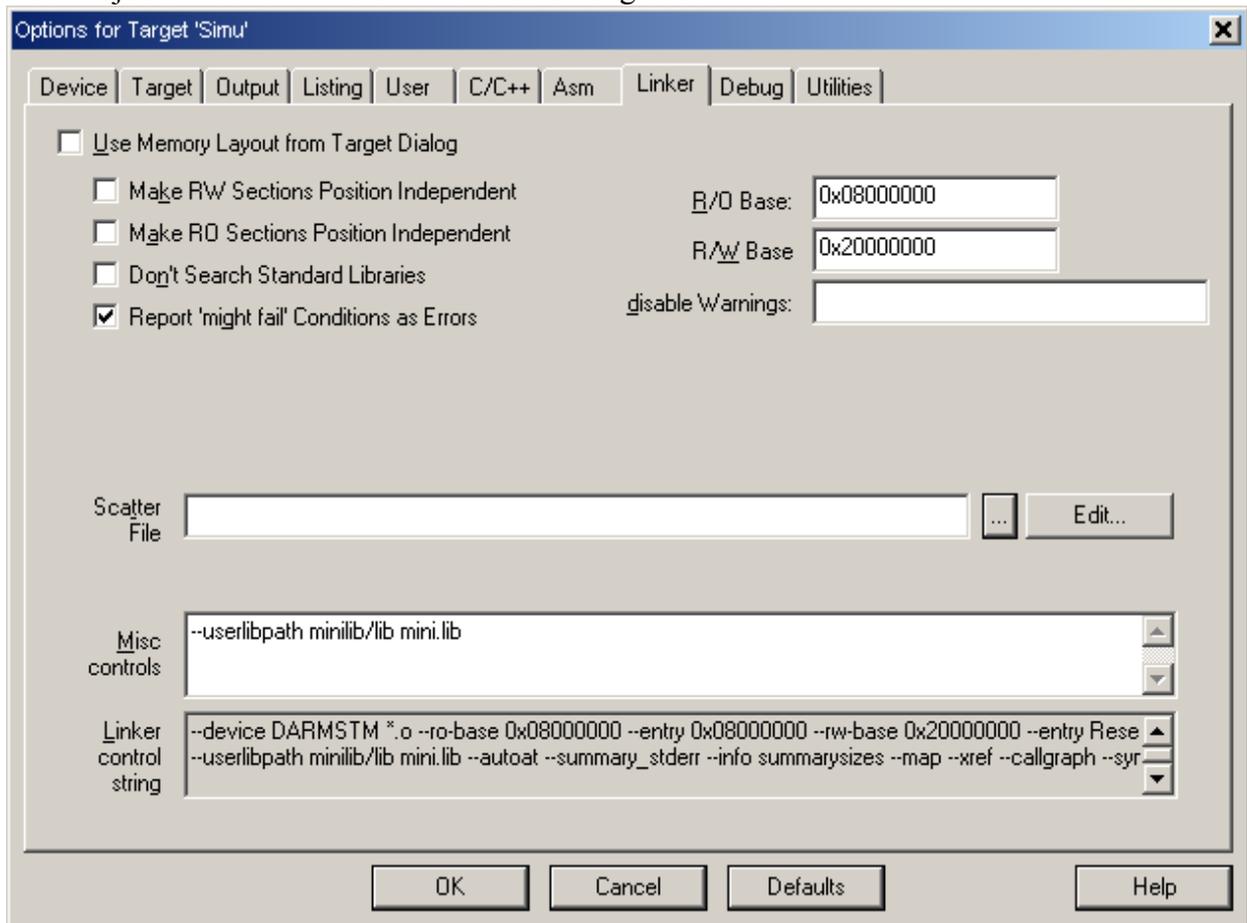


4. Bien verifier dans la fenetre precedente que les bibliotheques standards et les fichiers startup standards ne sont pas utilisés.

b) Utilisation avec ARMCC

1. Rajoutez le fichier `syscalls.c` à votre projet (le template se trouve sous `minilib/syscalls_template/syscalls.c`)
2. Adaptez les fonctions du fichier à votre besoin (voir le chapitre « Description des fonctions contenues dans `syscalls.c` » à propos des fonctions)

3. Rajoutez le chemin vers mini.lib dans l'onglet du linker



4. Bien vérifier dans la fenêtre précédente que les bibliothèques standards ne sont pas utilisées (voir aussi l'onglet Target et décocher, si nécessaire l'option Microlib).

4. Recompiler la bibliothèque

Normalement, il n'est pas nécessaire de recompiler la bibliothèque. Néanmoins, si le besoin se faisait sentir (bug découvert, évolution souhaité, ...), le répertoire **minilib/keil** contient deux projets: un pour ARMCC, l'autre pour GCC.

Il suffit d'ouvrir le projet ad-hoc et lancer la recompilation du projet (vérifier que la cible est bien la bibliothèque)

Sous GCC, il est très important de vérifier que les options **-ffunctions-section** et **-fdata-sections** sont ajoutées à la ligne de commande du compilateur (onglet CC): sans cela, au moment de l'édition de lien, GCC ne pourra pas supprimer les fonctions non utilisées → code gros, sans nécessité.

5. Options utilisables lors de la compilation de la bibliothèque

Les clefs de compilation suivantes (**define**) sont utilisables avec la bibliothèque:

PREFER_SIZE_OVER_SPEED:

Utilisée essentiellement par les fonctions du répertoire **string**

- Positionnée: le code est plus compact, mais plus lent.
- Desactivée: le code est plus efficace (transfert 32 bits au lieu de 8 bits), mais au prix d'un code plus complexe.

6. Description des fonctions contenues dans `syscalls.c`

- `_exit`: Utilisé essentiellement par la fonction `exit()`. Par défaut contient une boucle infinie.
- `_kill`: Non utilisé pour l'instant
- `_open`: Non utilisé pour l'instant
- `_read`: Utilisé par toutes les fonctions lisant dans un flux (`fgetc`, `fgets`, ...). Renvoie un tableau d'octet pointé par `*ptr`, d'une longueur `len`, rempli avec les données du flux `file`.
- `_write`: Utilisé par toutes les fonctions écrivant dans un flux (`fputc`, `printf`, `fprintf`, ...). Transfère un tableau pointé par `*ptr`, d'une longueur `len`, dans le flux `file`.
- `_malloc`: Sert à implémenter le mécanisme d'allocation mémoire
- `_free`: Compagnon de `_malloc`

7. Un mot au sujet des fonctions utilisant le type `FILE`

Des fonctions comme `fprintf`, `fputs`, `fgets`, etc utilisent un paramètre de type `FILE`. La définition par la minilib de la structure `FILE` est la suivante:

```
struct FILE
{
    int _file; // only field required for our lib C */
};
```

La structure ne contient qu'un champ, `_file`, qui contient un numéro de canal. Ce numéro sera, au final, passé aux fonctions `_read` et `_write`, selon que l'on lit dans un flux, ou qu'on y écrit. Ce numéro de canal peut dès lors servir à choisir, dans ces fonctions, parmi plusieurs canaux de lecture et ou d'écriture.

Bien entendu, si il n'existe qu'un seul canal de lecture et/ou d'écriture, il est inutile de remplir le champ `_file` et une variable, même non initialisée suffit (mais pas de pointeur `NULL` !)

De plus, `stdio.h` définit trois flux standards:

```
stdin: flux standard d'entrée
stdout: flux standard de sortie
stderr: flux standard d'erreur.
```

Ces trois flux sont de type `FILE` mais sont non initialisés de base (c-à-d qu'il ne contiennent pas forcément un numéro de canal valide). Ces flux sont notamment utilisés par `printf` (`stdout`) et `scanf` (`stdin`). Si un seul flux de donnée existe pour l'application, ou si il existe un flux de donnée privilégié, il est plus fûté de se servir de ces variables, et d'utiliser donc les formes `printf` et `scanf`, aux formes `fprintf` et `fscanf`.

7.1. Un point important au sujet de ces flux

ARMCC definit les flux standards comme des variables definit comme suit:

```
FILE __stdin;  
FILE __stdout;  
FILE __stderr;
```

GCC definit les flux standards comme des pointeurs sur variables definit comme suit:

```
FILE *stdin;  
FILE *stdout;  
FILE *stderr;
```

Notons au passage que les noms de ces flux ne sont pas les même selon que l'on utilise une chaine GCC ou ARMCC.

7.2. Utilisation concrete dans une application

Voici un exemple minimal, commenté, d'utilisation des flux:

- Fichier main.c

```
#define <<stdio.h>>  
#define "missing_defs.h"  
  
FILE mon_flux;  
  
void main (void)  
{  
    mon_flux._file=<numero_du_canal>;  
  
    fprintf(&mon_flux, "Hello world");  
  
    for (;;) ;  
}
```

- Fichier syscalls.c

```
int _write(int file, char *ptr, int len)  
{  
    int index;  
  
    for (index = 0; index < len; index++)  
    {  
        if (file == <numero_du_canal>)  
        {  
            fonction_ecrire_dans_ce_flux(ptr[index]);  
        }  
    }  
  
    return len;  
}
```