MANUEL D'UTILISATION DE LA MINILIB C

Version 1.01

S. Di Mercurio

Table des matières

1. Présentation

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 référer à la norme ISO/IEC C'99

• STDIO
  • **`printf`**
  • **`fprintf`**
  • **`sprintf`**
  • **`sscanf`**
  • **`fputc`**
  • **`fgetc`**
  • **`fputs`**
  • **`fgets`**
  • **`getchar`**
  • **`putchar`**
	• STDLIB
	  • **`atoi`**
	  • **`atol`**
	  • **`strtol`**
	  • **`div`**
	  • **`ldiv`**
	  • **`abs`**
	  • **`exit`**
	  • **`assert`**
	  • **`rand`**
	  • **`malloc`**
	  • **`calloc`**
	  • **`free`**
		• CTYPE
		  • **`isalnum`**
		  • **`isalpha`**
		  • **`isascii`**
		  • **`isblank`**
		  • **`iscntrl`**
		  • **`isdigit`**
		  • **`islower`**
		  • **`isprint`**
		  • **`ispunct`**
		  • **`isspace`**
		  • **`isupper`**
		  • **`isxdigit`**
		  • **`toascii`**
		  • **`tolower`**
		  • **`toupper`**

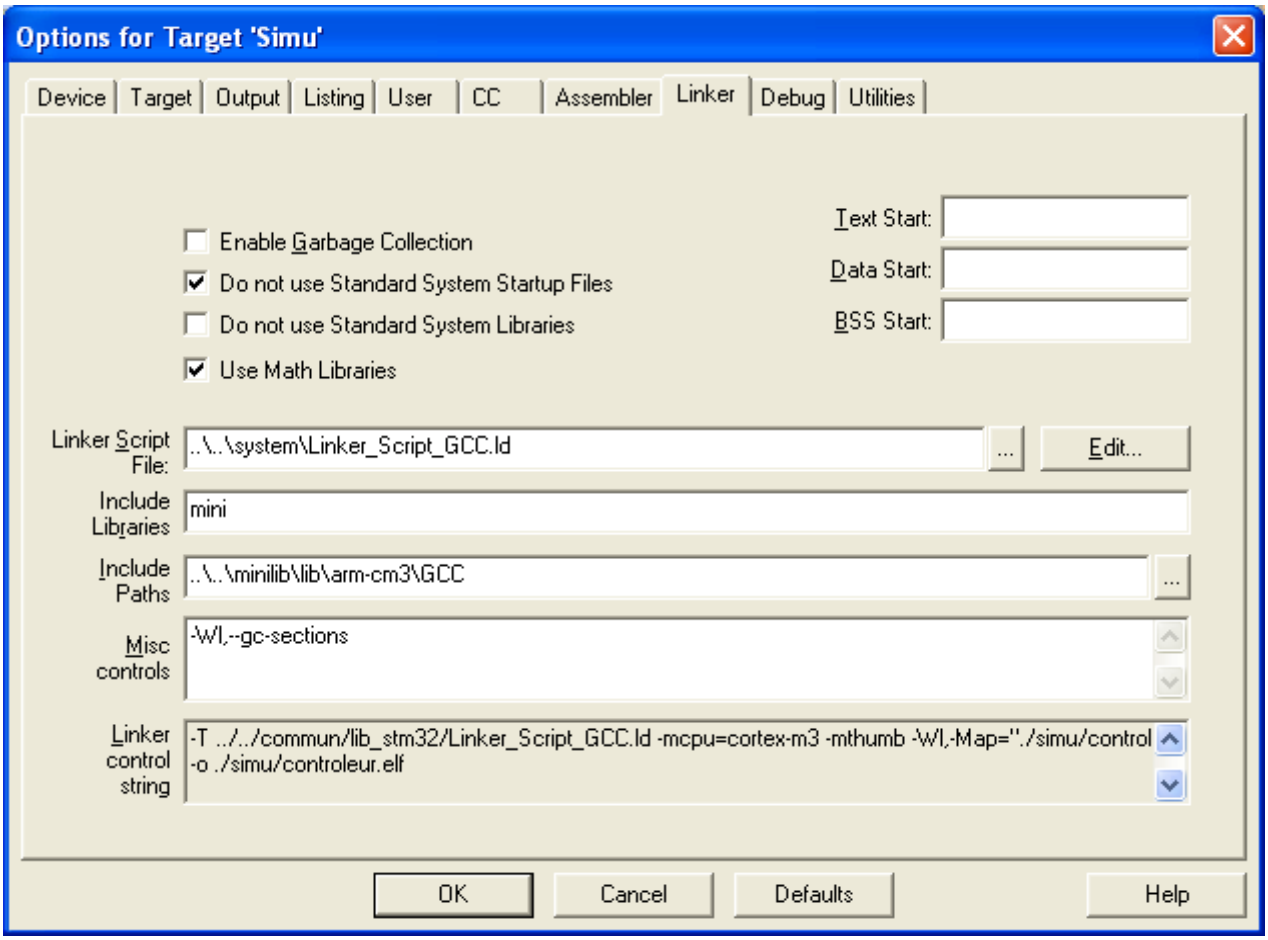
• STRING
  • **`bcmp`**
  • **`bcopy`**
  • **`bzero`**
  • **`index`**
  • **`memccpy`**
  • **`memchr`**
  • **`memcmp`**
  • **`memcpy`**
  • **`memmove`**
	• STRING (suite)
	  • **`mempcpy`**
	  • **`memsetrindex`**
	  • **`strcat`**
	  • **`strchr`**
	  • **`strcmp`**
	  • **`strcoll`**
	  • **`strcpy`**
	  • **`strcspn`**
	  • **`strlcat`**
		• STRING (fin)
		  • **`strlcpy`**
		  • **`strlen`**
		  • **`strlwr`**
		  • **`strncat`**
		  • **`strncmp`**
		  • **`strncpy`**
		  • **`strnlen`**
		  • **`strrchr`**
		  • **`strsep`**
		  • **`strspn`**
		  • **`strstr`**
		  • **`strupr`**

3. Utilisation de la bibliothèque

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. Dans l'onglet du linker, rajoutez le chemin vers libmini.a (Include Path) et le nom de la bibliothèque « mini » (Include libraries)

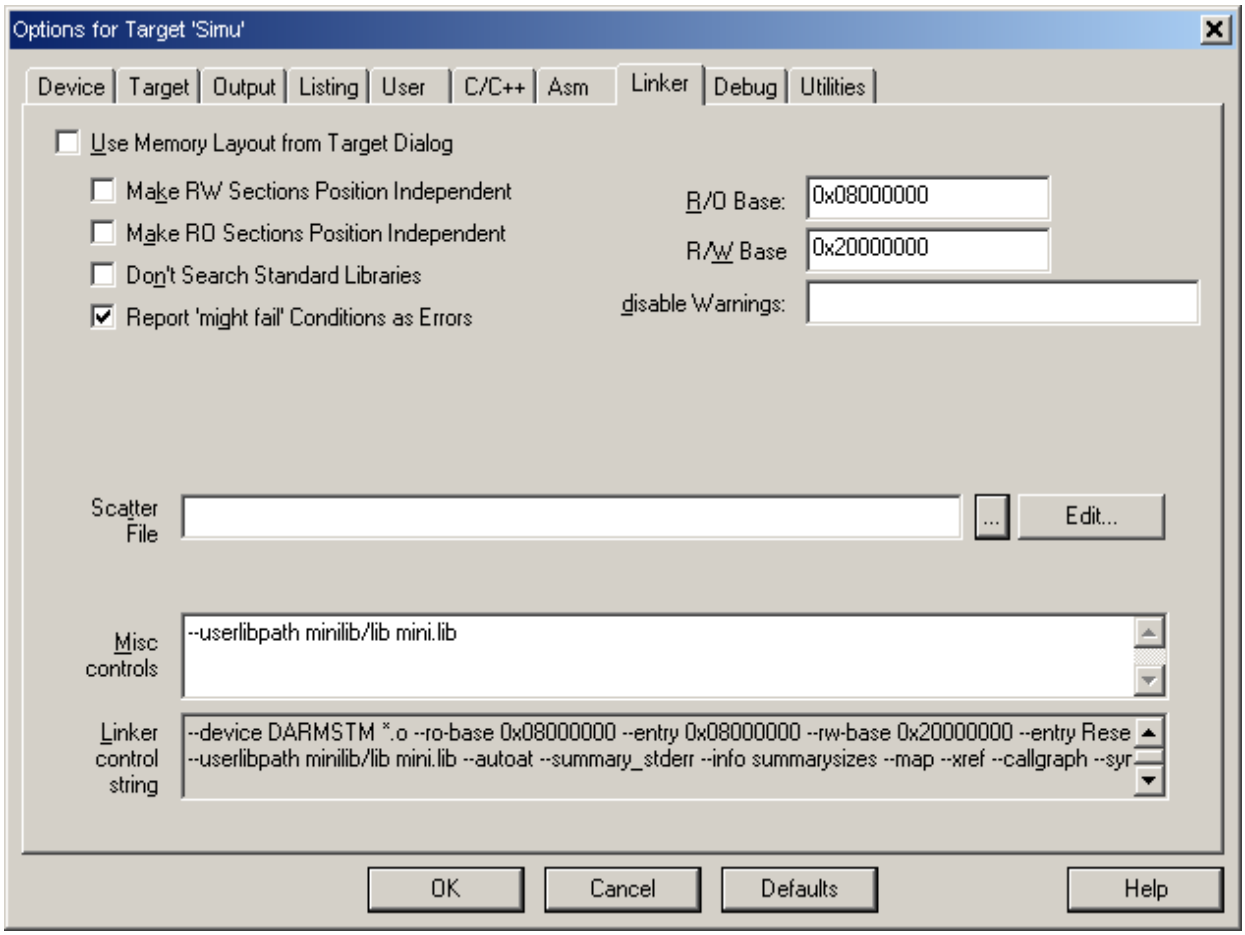


4. Bien vérifier dans la fenêtre précédente que les fichiers startup standards ne sont pas utilisés.

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: il suffit, normalement de rajouter le fichier mini.lib au projet (vérifier qu'il est rajouté comme bibliothèque)



4. Bien vérifier dans la fenêtre précédente que les bibliothèques standards ne sont pas utilisés (ce qui n'est pas la cas justement sur cette capture d'écran) (voir aussi l'onglet Target et décocher, si nécessaire l'option Microlib).

4. A propos du fichier « syscalls.c »

Le fichier « template » syscalls.c qui est fourni possède une clef de compilation (_MINILIB_) qui, si elle est définie dans le projet et que l'on utilise la bibliothèque stm_usartx..c, permet d'utiliser directement les fonctions printf et autres fputc pour envoyer du texte vers la liaison série.
Seulement, les limites d'un tel système sont qu'on ne peut utiliser qu'une liaison série comme sortie d'un printf.
Dés que l'on veut plusieurs moyen de communication (LCD, plusieurs usart, i2c, CAN, SPI, USB, …), la fonction printf n'est plus suffisante (il faut pouvoir indiquer vers où envoyer les caractères) et l'on utilisera du coup fprintf. De plus, les fonctions _write et _read contenues dans syscalls.c doivent être modifiées pour utiliser le paramètre file de ces fonctions pour diriger les caractères vers les différents canaux de com.

5. 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.

6. 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.
- Désactivée: le code est plus efficace (transfert 32 bits au lieu de 8 bits), mais au prix d'un code plus complexe.

7. 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 toute les fonctions lisant dans un flux (**fgetc**, **fgets**, …). Renvoi un tableau d'octet pointé par ***ptr**, d'une longueur **len**, remplit avec les données du flux **file**.
- **_write**: Utilisé par toutes les fonctions écrivant dans un flux (**fputc**, **printf**, **fprintf**, …). Transfert 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**

8. 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-a-d qu'il ne contiennent pas forcement 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 privilégie, il est plus futé de se servir de ces variables, et d'utiliser donc les formes **printf** et **scanf**, aux formes **fprintf** et **fscanf**.

8.1. Un point important au sujet de ces flux

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

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

GCC définit les flux standards comme des pointeurs sur variables définit 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.

8.2. Utilisation concrète dans une application

Voici un exemple minimal d'utilisation des flux:

- Fichier main.c

```
#define «stdio.h»
#define "missing_defs.h"

FILE vers_USART;
FILE vers_USB;
FILE vers_CAN;

void main (void)
{
      vers_USART._file=1;
      vers_USB_file=2;
      vers_CAN._file=3;

      fprintf(&vers_USART, "Du texte vers l'USART");
      fprintf(&vers_USB, "Du texte vers l'USB");
      fprintf(&vers_CAN, "Du texte vers le CAN");

      for (;;);
}
```

- Fichier syscalls.c

```
int _write(int file, char *ptr, int len)
{
int index;

      for (index = 0; index < len; index++)
      {
            if (file == 1) // 1 = USART
            {
                  write_usart(ptr[index]);
            }
            else if (file == 2) //2 = USB
            {
                  write_on_USB(ptr[index]);
            }
            else // 3 = CAN
            {
                  send_message_on_can(ptr[index]);
            }
      }

      return len;
}
```