

Informatique embarquée 4/16

J.-M Friedt

FEMTO-ST/département temps-fréquence

`jmfriedt@femto-st.fr`

transparents à `jmfriedt.free.fr`

7 février 2017

Exploitation d'une bibliothèque

- ① dangers d'une bibliothèque
- ② utilité d'une bibliothèque

Les ressources additionnelles ne se justifient que sur microcontrôleur haut de gamme (> 16 bits)

Diverses implémentations libres des bibliothèques standard du C :

- glibc (<http://www.gnu.org/software/libc/>, utilisée par le noyau Linux) ...
- ... et eglibc (*embedded* glibc, <http://www.eglibc.org/home>) ont fusionné,
- uClibc (<http://www.uclibc.org/>, utilisée par uClinux),
- ...

Exploitation d'une bibliothèque – newlib

Newlib : implémentation de `libc` répondant à toutes les exigences d'un OS \Rightarrow complet, mais complexe et lourd

L'utilisation de `printf` et du calcul flottant n'est pas à prendre à la légère :

Taille du fichier <code>.hex intel</code> ¹ :	avec <code>stdio</code>	80336
	sans <code>stdio</code>	4048

sans <code>stdio</code> , avec une division flottante	12950
---	-------

sans <code>stdio</code> , avec <code>atan</code> sur flottant	17090
---	-------

avec <code>stdio</code> , avec une division flottante	80397
---	-------

avec <code>stdio</code> , avec <code>atan</code> sur flottant	80397
---	-------

1. J.-M Friedt, É. Carry, *Développement sur processeur à base de cœur ARM7 sous GNU/Linux*, GNU/Linux Magazine France **117** (Juin 2009), pp.40-59

Le problème ...

Dans un programme en C, l'appel à `malloc()` se traduit par une erreur à l'édition de liens

```
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-sbrkr.o): In function '_sbrkr':  
newlib/libc/reent/sbrkr.c:58: undefined reference to '_sbrk'
```

Idem pour `printf()`

```
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-sbrkr.o): In function '_sbrkr':  
newlib/libc/reent/sbrkr.c:58: undefined reference to '_sbrk'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-writer.o): In function '_write_r':  
newlib/libc/reent/writer.c:58: undefined reference to '_write'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-closer.o): In function '_close_r':  
newlib/libc/reent/closer.c:53: undefined reference to '_close'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-fstatr.o): In function '_fstat_r':  
newlib/libc/reent/fstatr.c:62: undefined reference to '_fstat'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-isatty.o): In function '_isatty_r':  
newlib/libc/reent/isatty.c:58: undefined reference to '_isatty'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-lseekr.o): In function '_lseek_r':  
newlib/libc/reent/lseekr.c:58: undefined reference to '_lseek'  
arm-none-eabi/lib/thumb/cortex-m3/libc.a(lib_a-readr.o): In function '_read_r':  
newlib/libc/reent/readr.c:58: undefined reference to '_read'
```

Exploitation d'une bibliothèque – newlib

<https://sourceware.org/newlib/libc.html#Stubs> : 17 stubs, colle entre newlib et nos programmes².

- **_exit** : quitter un programme sans fermer les fichiers ouverts
- environ :
- execve :
- fork :
- fstat :
- getpid :
- isatty :
- kill :
- link :
- lseek :
- open :
- **read** : lire depuis un fichier
- sbrk : *utilisé par malloc*
- stat :
- times :
- unlink :
- wait :
- **write** : écrire dans un fichier, incluant stdout

Exploitation d'une bibliothèque – newlib

Exemple d'implémentation (envoyer stdio sur port série)

```
_ssize_t _read_r(struct _reent *, int file, void *ptr, size_t len)
{char c; int i; unsigned char *p;
 p = (unsigned char*)ptr;
 for (i = 0; i < len; i++)
   {while ( global_index == 0) {} // !uart0_kbhit() );
    c = global_tab[0];global_index=0; // (char) uart0_getc();
    if (c == 0x0D) {*p='\0';break;}
    *p++ = c; jmf_putchar(c,NULL,0,0);
  }
 return len - i;
}

_ssize_t _write_r ( struct _reent *, int file, const void *ptr, size_t len)
{int i; const unsigned char *p;
 p = (const unsigned char*) ptr;
 for (i = 0; i < len; i++)
   {if (*p == '\n') jmf_putchar('\r',NULL,0,0);
    jmf_putchar(*p++,NULL,0,0);
  }
 return len;
}

void jmf_putchar (int a,char *chaine, int* chaine_index,int USARTx)
{ if (chaine != NULL) {chaine[*chaine_index]=a;*chaine_index=*chaine_index+1;}
  else {if (usart_port==1)
    {USART_SendData (USART1, a);
     while(USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET) { ; }
    }
   else
    {USART_SendData (USART2, a);
     while(USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET) { ; }
    }
  }
}
```

Démonstration

```
#ifdef use_stdio
    for (tempe=1;tempe<10;tempe++) {mf=mf*lf;}
    printf( "\n+mf=%d\n" ,(int)mf);

    t=(char*) malloc(200); // malloc requiert _sbrk
    printf( "^%x\n^%x\n" ,(int)t[0] , t[199]);
    memset(t,0x55,200);
    printf( "^%x\n^%x\n" ,(int)t[0] , t[199]);
#endif

    while (1)
    {
#ifdef use_stdio
        put_chars(welcome);
#else
        printf( "%s" , welcome);
#endif
        if (i<10) put_char(USART1,i+'0');
            else put_char(USART1,i+'A'-10);
        i++;if (i==16) i=0;
    }
    ...
```

Démonstration

Compromis entre temps de développement, fonctionnalités et occupation de ressources

```
3240 main.bin  
31824 main_stdio.bin
```

Ici, affichage par `printf()`, allocation de mémoire par `malloc()`, calculs flottants

Un émulateur pour tester ses programmes en l'absence du matériel :

https://github.com/beckus/qemu_stm32

`qemu-system-arm -M stm32-p103 -serial stdio -serial stdio -kernel main.bin`
suppose néanmoins une implémentation “correcte” des périphériques.

Problème de la cohérence de la chaîne de compilation

- Les bibliothèques doivent suivre le même modèle de binaire que ceux générés par le compilateur (EABI)
- ... et être compilées avec les mêmes options (e.g. hard float v.s soft float).
- Recompiler ses outils pour garantir la cohérence :
<https://github.com/jmfriedt/summon-arm-toolchain> qui se charge de configurer tous les outils de la même façon

```
TARGET=arm-none-eabi  
PREFIX=${HOME}/sat  
./configure --target=${TARGET} --prefix=${PREFIX} --enable-multilib ...
```

Méthodes de déverminage de code embarqué

- 1 la partie algorithmique du code est portable et se compile sur PC
⇒ bien séparer les parties algorithmiques et accès au matériel
+ donne accès aux outils de profilage et analyse du PC
- 2 l'émulateur (plus souple que le matériel, et permet de sonder la machine à états qu'est le processeur)
- 3 sonde JTAG et gdb (si disponible) : exécution sur le "vrai" matériel avec points d'arrêt et avance pas à pas.

Compilation séparée pour qualifier un logiciel

- tester des conditions de fonctionnement imprévues ou rarement accessibles expérimentalement
- exploiter des outils de déverminage (valgrind) ou de profilage
- test unitaire

Programme principal, bibliothèque

```
int main()
{initialisation();
 while (1) {interroge();traitements();}
}
```

Programme pour microcontrôleur

```
unsigned short interroge (unsigned int freq)
{ GPIO_SetBits(GPIOB,GPIO.Pin_1);
  v = readADC1 (ADC_Channel_8);
  GPIO_ResetBits (GPIOB,GPIO.Pin_1);
  ...
  return (v);
}
```

Compilation séparée pour qualifier un logiciel

- tester des conditions de fonctionnement imprévues ou rarement accessibles expérimentalement
- exploiter des outils de déverminage (valgrind) ou de profilage
- test unitaire

Programme principale, bibliothèque

```
int main()
{initialisation();
 while (1) {interroge();traitements();}
}
```

Programme pour PC

```
unsigned short interroge (unsigned int freq)
reponse =exp(-(((float) freq-f01)/df)*(((float) freq-f01)/df))*3100.;
reponse+=exp(-(((float) freq-f02)/df)*(((float) freq-f02)/df))*3100.;
reponse+=(float)((rand()-RAND_MAX/2)/bruit); // ajout du bruit;
if (reponse <0.) reponse=0.;
if (reponse >4095.) reponse=4095.;
usleep(60);
return((unsigned short)reponse);
```

Cas des interruptions

Version PC

```

void handle_alarm(int xxx)
{ // printf("RTC : %d %d %d\n",seconde, →
  ↪minute, heure);
  tim0+=10;
  seconde+=10;
  if (seconde > 600){minute++;seconde=0;}
  if (minute > 60) {heure++;minute=0;}
  alarm(1);
}

void handle_io(int xxx)
{ // io_happened=1;
  global_tab[global_index] = getchar ();
  if (global_index < (NB_CHARS - 1))
    global_index++;
}

int main() {
  signal(SIGALRM, handle_alarm);

  io_happened=0;
  struct termios buf;
  tcgetattr(0, &buf);
  buf.c_lflag &= ~(ECHO | ICANON);
  buf.c_cc [VMIN]=1;
  buf.c_cc [VTIME]=0;
  tcsetattr(0, TCSAFLUSH, &buf);
  signal(SIGIO, handle_io);
  int savedflags=fcntl(0, F_GETFL, 0);
  fcntl(0, F_SETFL, savedflags | O_ASYNC | →
    ↪O_NONBLOCK );
  fcntl(0, F_SETOWN, getpid());
  alarm(1); // si on veut simuler le timeout
}

```

Version microcontrôleur

```

void tim3_isr(void) // VERSION LIBOPENCM3
{
  if (TIM_GetITStatus(TIM3, TIM_IT_Update)!⇒
    ↪RESET)
  {TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
  tim0++;
  seconde++; // seconde en 1/10 →
    ↪seconde
  if (seconde > 600)
    {seconde = 0; minute++;}
  if (minute > 60)
    {minute = 0; heure++;}
  if (heure > 24)
    {heure = 0;}
}
}

void usart1_isr(void)
{
  if( USART_GetITStatus(USART1, USART_IT_RXNE→
    ↪))
  {USART_ClearITPendingBit(USART1, →
    ↪USART_IT_RXNE);
  USART_ClearFlag(USART1, USART_IT_RXNE);
  {
    global_tab[global_index]=⇒
    ↪USART_ReceiveData(USART1);
    if (global_index <(NB_CHARS-1)) →
    ↪global_index++;
  }
}
}

```

Profiter des outils d'analyse de code

valgrind pour étudier les fuites de mémoire

```
int main()  
{int i[3],k;  
  for (k=0;k<10;k++) {i[k]=k; printf("%d ", i[k]);}  
}
```

qui s'exécute pour donner Segmentation fault ...

... est analysé par valgrind -q ./mon_programme et donne

```
==7410== Access not within mapped region at address 0x2  
==7410== at 0x8048448: main (demo_valgrind.c:5)
```

(noter la différence de comportement en ajoutant fflush(stdout) ou \n!)

Profiter des outils d'analyse de code

gprof pour étudier le temps d'exécution de chaque fonction

```
void fonction1s() {volatile int k;for (k=0;k<0x1000000;k++) {};}
void fonction2s() {volatile int k;for (k=0;k<0x2000000;k++) {};}
void fonction3s() {volatile int k;for (k=0;k<0x3000000;k++) {};}

int main()
{fonction1s();
 fonction2s();
 fonction3s();
}
```

qui se compile avec l'option `-pg` pour générer à l'exécution `gmon.out` ...

... qui s'analyse par `gprof -bp ./mon_programme` et donne

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
55.56	0.10	0.10	1	100.00	100.00	fonction3s
33.33	0.16	0.06	1	60.00	60.00	fonction2s
11.11	0.18	0.02	1	20.00	20.00	fonction1s

Mise en pratique

Nous quittons l'AVR qui a choisi de développer sa propre libc, pour passer au STM32

- processeur 32 bits capable d'embarquer des bibliothèques (puis environnements exécutifs)
- `summon-arm-toolchain` fournit un script pour générer un ensemble "cohérent" d'outils – explication de sa compilation à http://jmfriedt.free.fr/summon_arm.pdf
- le processeur se décline en une multitude de classes : *linker* script approprié.
- Émulateur pour STM32 : `qemu` pour ARM complété de l'émulation de périphériques – https://github.com/beckus/qemu_stm32.

Les divers outils de compilation : `Makefile`, `configure`, `cmake`

Les divers outils d'archivage : `svn`, `git`, `hg` (mercurial)