LOGBOOK

Luc Weber
Observatoire de Genève

24 octobre 2012

Table des matières

Chapitre 1

Introduction

LogBook est un utilitaire d'affichage de message. Il travaille en mode serveur et accepte un grand nombre de client (le maximum est définis par le système). Chaque client peut se connecter ou se déconnecter à tout moment.
La communication avec le *LogBook* se fait au travers de la librairie GOP (Geneva Observatory Protocole) qui est basée sur la communication par sockets TCP/IP.
LibLogBook est la librairie contenant les fonctions d'accès au *LogBook*. On y trouve les fonctions de connexion et d'envoi de message. Ces fonctions sont accessibles depuis le C ou le Fortran.
Cette librairie contient également des fonctions d'affichage (en C) permettant a l'utilisateur de normaliser ses fonctions d'écriture en un appel standard (même syntaxe que printf) et par la suite définir (au moyen de l'enregistrement d'une fonction personnelle) la destination du texte envoyé. Ainsi, on peux envoyer l'ensemble des messages au *LogBook* ou sur tout autre sortie et même simultanément sur plusieurs sorties.

L'utilisation d'un Log Book unique par machine est une des notion de base de l'emploi de *LogBook*. Ainsi la connexion est simplifiée. Le numéro de port du socket n'est pas à fournir car il est codé dans la librairie. Il est récupérable par la fonction `lbk_get_port_standard()`.
La structure est elle aussi recupérable avec la fonction : `lbk_get_connect_struct()` qui retourne le pointeur sur la structure interne.

Chaque process n'a en principe accès qu'a un seul *LogBook*. La structure de connexion `gop_connect` est unique et existe dans la librairie *LibLogBook*. Les appels au logbook en sont d'autant simplifié. En C, on peut toutefois copier une structure de connexion locale au process dans la structure de *LibLogBook* (voir `lbk_set_param_log_book()`).

Pour l'emploi de certaines fonctions, il est nécessaire d'inclure le fichier "`logbook.h`".

Chapitre 2

Utilisation du LogBook

LogBook se lance avec les options suivantes :

-f <**file_name**> donne le nom d'un fichier ou les messages sont stockés en plus de l'affichage dans la fenêtre.

-v <**gop_verbose_value**> donne le niveau de verbosité de *GOP* (0-9) pour initialisation du *LogBook*.

-M <**memory_size**> donne la taille mémoire de la fenêtre du *LogBook*.

Le niveau de verbosité de Gop est par la suite définis par chacun des clients.

Chapitre 3

Accès au LogBook

Toutes les fonctions suivantes sont accessibles depuis le C et le Fortran. D'une manière générale, on conserve le status de la connexion puis on écrit sur le *LogBook* uniquement si le status est correct.

3.1 Connexion

Le déroulement de la connexion sur le *LogBook* se fait en deux temps. D'abord la mise–à–jour de la structure puis la connexion proprement dite.

Cela donne par exemple en C :

```
lbk_set_default_param_log_book();
lbk_set_my_name(name);
lbk_set_name(log_book_host);
if ((log_book_status = lbk_client_log_book()) == -1) {
   fprintf(stderr, "Connexion impossible sur le log book\n");
}
```

en Fortran :

```
call lbk_set_default_param_log_book()
call lbk_set_my_name(my_name(1:len_my_name)//char(0))
call lbk_set_name(log_book_host(1:len_log_book_host)//char(0))
call lbk_client_log_book(log_book_status)
if(log_book_status .eq. -1)then
   write(*,*)"Connexion impossible sur le log book"
endif
```

Dans le cas où le client possède une structure de connexion, il doit la copier sur la structure interne de *LibLogBook*. Ce cas arrive si le client utilise la fonction *GOP* `gop_parse_opt_cmd_line()`. Cette fonction permet de remplir une structure de connexion avec les arguments de la ligne de commande. Dans ce cas la structure est déclarée dans le programme. C'est au client de la copier dans la structure interne de *LibLogBook*. Ce cas est possible uniquement en C et on a par exemple :

```
main(int argc, char **argv)
{
     int   use_logbook = GOP_FALSE;
     struct gop_connect *connect_lbk;
     ....
     /* allocate structure */
     connect_lbk = (struct gop_connect *) gop_alloc_connect_structure();
     /* set defaults */
     gop_set_name(connect_lbk, "localhost");
     gop_set_type(connect_lbk, GOP_SOCKET);
     gop_set_port(connect_lbk, lbk_get_port_standard());
     gop_set_mode(connect_lbk, 0);
     gop_set_my_name(connect_lbk, my_name);
     gop_set_maxpacket(connect_lbk, maxpacket);
     /* read command line options */
     while ((c = getopt(argc, argv, "L:")) != -1) {
          switch (c) {
          case 'L':
               use_logbook = GOP_TRUE;
               gop_parse_opt_cmd_line(connect_lbk);
               break;
          }
     }
     /* connexion */
     if (use_logbook) {
          /* connect structure copy */
          lbk_set_param_log_book(connect_lbk);
          if ((log_book_status = lbk_client_log_book()) == -1) {
              fprintf(stderr, "Connexion impossible sur le log book\n");
          } else {
               /* redirection des message sur le logbook */
               lbk_registration_for_info(lbk_print_on_logbook);
               gop_registration_for_printf(lbk_print_on_logbook);
          }
     }
}
```

3.2 Envoi de messages

L'envoi de message se fait par la commande `lbk_write_log_book()`. Ses arguments diffèrent selon son utilisation en C ou en Fortran.

Cela donne par exemple en C :

```
log_book_status = lbk_write_log_book(str);
```

en Fortran :

```
call lbk_write_log_book(str, log_book_status);
```

3.3 Autres fonctions

Déjà utilisées en partie dans les exemple ci–dessus, les fonctions suivantes permettent de modifier la structure de connexion interne a *LibLogBook*. Ce sont :

```
void lbk_set_name(char *name)     /* nom du host */
void lbk_set_my_name(char *name)  /* nom symbolique du client */
void lbk_set_verbose(int verbose) /* niveau de verbosité GOP */
```

3.4 Reconnexion

Si le *LogBook* est tué pour une quelconque raison ou si le client est lancé avant le *LogBook*, il devient impossible d'envoyer des message même si un logbook est relancé sans faire de reconnexion. Le problème est résolu par l'emploi du signal SIGUP. Le client enregistre un handler pour le signal SIGUP et lorsqu'il le reçoit, le client effectue une reconnexion, le code client (en C) a cette allure :

```
void handler_sigup(int sig)
{
     printf("spectro_srv: reçu signal SIGHUP -> reconnexion sur logbook\n");
     if (log_book_status == -1) {
          if ((log_book_status = lbk_client_log_book()) == -1) {
              fprintf(stderr, "Connexion impossible sur le log book\n");
          }
     }
}

main()
{
     int   use_logbook = GOP_FALSE;
     signal(SIGHUP, handler_sigup);
     ...
}
```

Le signal s'envoie avec la commande Unix `kill`. Exemple :

```
     unix_prompt> ps -ef my_process
     unix_prompt> kill -1 my_process_pid
```

Chapitre 4

Fonction d'affichage (LibLogBook)

Les fonctions d'affichage décrites ci–dessous s'utilise en C uniquement. Elles permettent le remplacement des fonctions C : `printf()` ou `fprintf()` par une fonction utilisateur qui permet, même en cours de processus, d'aiguiller les impressions vers la sortie adéquate.

La fonction utilisateur reçoit comme argument la chaîne formatée et doit l'envoyer vers une sortie de son choix, par exemple : le *LogBook*, un fichier, stdout, stderr, ... Cette fonction peut naturellement effectuer d'autre tâche ou envoyer les message simultanément sur plusieurs sortie.

Il existe une seule fonction utilisateur par fonction d'impression, celle–ci est peut être enregistrée à tout moment ou être remplacée en cours de travail. Si aucune fonction utilisateur n'est enregistrée, le fonctionnement par défaut de ces fonction est l'impression sur stdout.
Ces fonctions sont au nombre de cinq, ce sont :

```
lbk_printf  (char *format [, args ...])
lbk_info    (char *format [, args ...])
lbk_debug   (char *format [, args ...])
lbk_warning (char *format [, args ...])
lbk_error   (char *format [, args ...])
```

Chacune fournis un style d'impression différent. Exemple :

```
lbk_printf  ("Essai No %d\n", 1)
     donne:  "Essai No 1"
```

```
lbk_info    ("Essai No %d\n", 2)
     donne:  "Info:    Essai No 2"
```

```
lbk_debug   ("Essai No %d\n", 3)
     donne:  "Debug:   Essai No 3"
```

```
lbk_warning ("Essai No %d\n", 4)
     donne:  "Warning: Essai No 4"
```

```
lbk_error   ("Essai No %d\n", 5)
     donne:  "Error:   Essai No 5"
```

l'enregistrement des fonctions se fait avec avec les fonctions suivantes :

```
lbk_registration_for_printf  ((*fct) ())
lbk_registration_for_info    ((*fct) ())
lbk_registration_for_debug   ((*fct) ())
lbk_registration_for_warning ((*fct) ())
lbk_registration_for_error   ((*fct) ())
```

Dans l'exemple suivant, l'utilisateur défini une fonction d'impression qui envoie les message sur le *LogBook* si la connexion sur celui–ci est valide ou sur `stdout` dans le cas contraire. Dans cet exemple, le programme écrira ce texte dans le logbook si la connexion a réussi :

```
Status de connexion sur logbook = 0
```

ou dans la fenêtre courante, si la connexion sur *LogBook* n'a pas réussi :

```
Status de connexion sur logbook = -1
```

Le code est :

```
int log_book_status=-1;

print_on_logbook(str)
     char           *str;
{
     if (log_book_status != -1) {
          log_book_status = lbk_write_log_book(str);
     } else {
          fprintf(stdout, "%s", str);
     }
}

main()
{
     ...
     log_book_status = lbk_client_log_book()
     lbk_registration_for_printf(print_on_logbook);
     lbk_printf("Status de connexion sur logbook = %d\n", log_book_status);
     ...
}
```