uLan/Universal Light Event Poll Library (ULEVPOLL)

Pavel Pisa

pisa@cmp.felk.cvut.cz

uLan/Universal Light Event Poll Library (ULEVPOLL)
by Pavel Pisa

The uLevPoll library provides infrastructure to to process system level events in application with well defined and portable triggers register and modification operations.

Table of Contents

1. Event Processing Library Concepts..1

1.1. The Goals..1

1.2. Cascading of Event Processing Implementations..1

1.3. History..2

2. Functions Description..4

2.1. Basic Level Public API..4

struct ul_evptrig_t..4

struct ul_evpbase_t..5

ul_evptrig_preinit_detached..5

ul_evptrig_is_detached..6

ul_evptrig_init..7

ul_evptrig_done..7

ul_evptrig_set_fd..8

ul_evptrig_set_time..9

ul_evptrig_set_timeout..9

ul_evptrig_set_callback..10

ul_evptrig_arm..11

ul_evptrig_disarm..11

ul_evptrig_arm_once..12

ul_evptrig_set_param..12

ul_evptrig_get_param..13

ul_evptrig_get_base..14

ul_evpoll_new..14

ul_evpoll_destroy..15

ul_evpoll_update..15

ul_evpoll_dispatch..16

ul_evpoll_get_current_time..17

ul_evpoll_loop..17

ul_evpoll_quilt_loop..18

ul_evpoll_cascade..18

Chapter 1. Event Processing Library Concepts

1.1. The Goals

The processing of system level events is basic need of most of applications. When multiple events should be processed in single thread some mechanism to allows select which events should be wait for and how they should be processed is required.

Many of projects are assembled from multiple libraries/components. These libraries should be portable and need to be able to integrate into different environments and applications. It is possible to write many of them without need to block on events such way, that they are only fed by data by main application and provide data back. But there are many situations, when even libraries depend processing system level events and need to register into application wide events processing mechanism.

But there is critical problem for libraries that they have to follow application environment selected event processing mechanism or introduce own one and force application use it. Problem is to combine libraries written for different environments or to select different application environment the library has been designed for.

The goal of uLevPoll is to provide common interface which allows to hide environment differences and allows to write libraries which can be used in diferent environments without need of rewrite or recompile.

To achieve goal next list of requirements has been defined

- use such FD monitoring mechanism, which would be well portable
- even binary version of compiled libraries has to be independent of application selected main loop mechanism used by applications which use components/libraries ⟶ libraries have to adapt for main loop used by applications
- libraries should allow to be used with minimal set of external dependencies to allow their use in small embedded applications
- but components should integrate well even with graphical or large applications, so defined interface should not prevent use of Gtk or Qt for main loop in applications
- the used mechanism should allow to switch to high throughput solution (as libevent is for example) when required in future.

The uLevPoll library API/ABI is defined on above basis. It exposes minimal amount of information directly to user - only "handler" like event trigger structure with minimal set of fields and pointer to one field of event base structure with information about used operations set.

1.2. Cascading of Event Processing Implementations

The other interesting feature is to be able to switch or cascade event monitoring at runtime when some third party library enforcing different main loop implementation is dynamically loaded. This goal has been achieved by uLevPoll as well. Next complex scenarios works now

- start application with Linux epoll or Sys V poll base, when GLIB based library is required, create new uLevPoll based on GLIB, cascade original set with epoll above it (or transform Sys V poll triggers to GLIB based ones in new main loop) and continue to run with GLIB main loop.
- start with GLIB base, wrap it as uLevPoll base and when GLIB scalability fails, create new uLevPoll based on Linux epoll which can be cascaded into GLIB main loop and use this new better scalling base for most of the evenets registration.

The events can be inserted by GLIB based applications as glib event sources, by uLUt based applications as ul_evpoll events into uLevPoll wrapper or over original sysvpoll or lnxepoll event bases and all runs concurrently without noticing real bottom base in use. The Linus epoll cascaded over GLIB base can corrects GLIB ill behavior for C10K problem for these events, which are registered over uLevPoll API as ul_evptrig_t into epoll based ul_evpbase_t.

1.3. History

We have need for system events/file handles monitoring for uLan project and other PiKRON company projects in 2007. The selected solution should provide functionality of other older Sys V poll based code included in OCERA project CAN/CANopen VCA component as well.

The libevent looked as good candidate for our projects at that time, even that it would add yet another prerequisite for our projects. It was considered acceptable. But the goal of our libraries/components was to allow their combination in environment based on other main-loop implementation (Qt, GTK, Python). But libevent enforces its own main loop and prevents to use libraries based on it to integrate into other environment main loop mechanism. This was considered as fundamental problem and the goals (Section 1.1) for required solution has been defined.

The minimal API conforming these requirements and allowing separation of libraries code from used system event processing method has been defined.

Then the simple Sys V poll based implementation has been provided to allow stand-alone use of the API without enforcing external dependencies. The use of libevent-1 has been considered as next target. But after deeper look at libevent-1 code distributed with Debian stable, the analysis shown, that it is unusable for multi-threaded environment - event_base_new() has not been provided by that version and sequence for creation and attaching of events to non default base has been considered strange as well.

For above reasons, the own epoll based mechanism was implemented for uLevPoll. During its testing some misbehavior in epoll Linux kernel implementation has been found. Davide Libenzi has kindly provided help and result is enhancement in Linux epoll implementation and introduction of keyed wake-ups which lead to significant speedup even for plain blocking read and write socket operations.

Next experiment was to try, if designed ABI really allows components to be compatible with Gtk/Glib and Qt. Fortunately, usual distributions Qt builds use Glib main loop so only support for that was added to uLevPoll. It allows to hide Glib event sources based API under uLevPoll API for our libraries which can then transparently use Glib main loop without notice of that. Yet for different threads better performing epoll base main loop can be used. Even for main thread event loop it is possible to cascade over uLevPoll Glib abstraction another uLevPoll base with different mechanism (epoll for example) which is great win because Glib main loop has horrible scalability.

Then the time to finally try move to libevent come. But version 1 has been disappointing. But new development version 2 shows in much better light. It really allows multi-threaded support and when more available by distributions, it would allow to use it as high performance mechanism for uLevPoll. uLevPoll wrapper code has been adapted for libevent 2 now.

Chapter 2. Functions Description

2.1. Basic Level Public API

struct ul_evptrig_t

Name

`struct ul_evptrig_t` — event trigger public structure

Synopsis

```
struct ul_evptrig_t {
  struct ul_evptrig_data_t * impl_data;
  ul_evpbase_t * base;
  ul_evpoll_cb_t cb;
};
```

Members

impl_data

pointer to implementation data set by base in `ul_evptrig_init`

base

pointer to the base which event is member of

cb

pointer to user callback function

Description

The event trigger is basic element of whole library. One or more instances of &ul_evptrig_t structure are typically contained by some user data structure holding state for given communication object. Structure is initiated and assigned to selected event poll base by `ul_evptrig_init`. From this point it s associated to base until `ul_evptrig_done` is called. If poll base is destroyed / `ul_evpoll_destroy` called before `ul_evptrig_done`, `UL_EVP_DONE` event is delivered to the assigned callback function. It should call `ul_evptrig_done` in such case.

The trigger is setup to accept selected events, functions `ul_evptrig_set_fd`, `ul_evptrig_set_callback`, `ul_evptrig_set_time`/`ul_evptrig_set_timeout` and then it is marked active by `ul_evptrig_arm` or `ul_evptrig_arm_once` call. When event occurs, the callback *`cb`* is activated with set of active events. The argument of *`evptrig`* is pointer to corresponding &ul_evptrig_t structure. Use of `UL_CONTAINEROF` is expected to obtain pointer communication object data structure containing activated event trigger. Monitoring of given event can be (temporarily) disabled by `ul_evptrig_disarm`.

struct ul_evpbase_t

Name

`struct ul_evpbase_t` — common part of event poll base structure

Synopsis

```
struct ul_evpbase_t {
  const ul_evpoll_ops_t * ops;
};
```

Members

ops

pointer set of operations provided by this base

Description

There is typically one such base for each thread which needs to process events. The poll base is created by `ul_evpoll_new` call. `ul_evpoll_destroy` informs all attached triggers (`UL_EVP_DONE`), about base cease, ensures, that implementation specific data are released even for triggers, which do not call `ul_evptrig_done` / handle `UL_EVP_DONE`, closes and deallocates base.

The call `ul_evpoll_dispatch` starts single iteration waiting for events. If there is no need to implement own loop in application the `ul_evpoll_loop` can be called to handle all events for given thread. The loop is terminated when call `ul_evpoll_quilt_loop` is used during iteration.

ul_evptrig_preinit_detached

Name

`ul_evptrig_preinit_detached` — mark trigger structure as not initialized yet

Synopsis

```
void ul_evptrig_preinit_detached (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

Description

This call allows user application to initialize communication object data structure and later check, if given trigger is already initialized or not. Only valid operation for uninitialized trigger is `ul_evptrig_is_detached`

ul_evptrig_is_detached

Name

`ul_evptrig_is_detached` — test if trigger is not initialized/attached to base

Synopsis

```
int ul_evptrig_is_detached (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

ul_evptrig_init

Name

`ul_evptrig_init` — initialization of event trigger structure

Synopsis

```
int ul_evptrig_init (ul_evpbase_t * base, ul_evptrig_t * evptrig);
```

Arguments

base

event poll base

evptrig

event trigger

Description

If the *base* parameter is `NULL`, default base is found/created and event trigger is attached to that default base. Base allocates required event rigger implementation data and fills *impl_data* pointer.

ul_evptrig_done

Name

`ul_evptrig_done` — detach and done event trigger

Synopsis

```
void ul_evptrig_done (ul_evptrig_t * evptrig);
```

Arguments

`evptrig`

event trigger

Description

Operation can be called only to previously initialized trigger

ul_evptrig_set_fd

Name

`ul_evptrig_set_fd` — set file descriptor monitored for specified events

Synopsis

```
int ul_evptrig_set_fd (ul_evptrig_t * evptrig, ul_evfd_t fd, int what);
```

Arguments

evptrig

event trigger

fd

file descriptor

what

which events to monitor - set of `UL_EVP_READ`, `UL_EVP_WRITE` `UL_EVP_STATE`

ul_evptrig_set_time

Name

`ul_evptrig_set_time` — set absolute time to trigger event

Synopsis

```
int ul_evptrig_set_time (ul_evptrig_t * evptrig, ul_htim_time_t * time);
```

Arguments

evptrig

event trigger

time

pointer to absolute time specification to trigger event

Description

The call back is activated with `UL_EVP_TIMEOUT` set for armed event when time elapses. The time can be changed even for armed event trigger freely and is set to never if *`time`* is `NULL`

ul_evptrig_set_timeout

Name

`ul_evptrig_set_timeout` — inactivity timeout for trigger event

Synopsis

```
int ul_evptrig_set_timeout (ul_evptrig_t * evptrig, ul_htim_diff_t *
timeout);
```

Arguments

evptrig

event trigger

timeout

pointer relative time inactivity interval triggering event

Description

The call back is activated with `UL_EVP_TIMEOUT` if there is no activity on given trigger for given time interval. The timeout value and start time can be re-trigger by call to `ul_evptrig_set_timeout` even for armed event. The disarm and arm sequence re-triggers timeout interval start as well. *timeout* equal to `NULL` disables timeout monitoring.

ul_evptrig_set_callback

Name

`ul_evptrig_set_callback` — set user calback function for given event trigger

Synopsis

```
int ul_evptrig_set_callback (ul_evptrig_t * evptrig, ul_evpoll_cb_t cb);
```

Arguments

evptrig

event trigger

cb

callback function

ul_evptrig_arm

Name

`ul_evptrig_arm` — activates trigger to monitor for selected events

Synopsis

```
int ul_evptrig_arm (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

ul_evptrig_disarm

Name

`ul_evptrig_disarm` — stop monitoring of events by this trigger

Synopsis

```
int ul_evptrig_disarm (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

ul_evptrig_arm_once

Name

`ul_evptrig_arm_once` — activates trigger to wait for first of events only

Synopsis

```
int ul_evptrig_arm_once (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

ul_evptrig_set_param

Name

`ul_evptrig_set_param` — set extended/system specific parameter

Synopsis

```
int ul_evptrig_set_param (ul_evptrig_t * evptrig, int parnum, const void *
parval, int parsize);
```

Arguments

evptrig

event trigger

parnum

parameter number `UL_EVPTRIG_PARAM_xxx`

parval

pointer to value to be set

parsize

the size of the parameter

ul_evptrig_get_param

Name

`ul_evptrig_get_param` — get extended/system specific parameter

Synopsis

```
int ul_evptrig_get_param (ul_evptrig_t * evptrig, int parnum, void * parval,
int parmaxsize);
```

Arguments

evptrig

event trigger

parnum

parameter number UL_EVPTRIG_PARAM_xxx

parval

pointer to buffer to store value

parmaxsize

-- undescribed --

ul_evptrig_get_base

Name

ul_evptrig_get_base — get pointer to poll base trigger is member of

Synopsis

```
ul_evpbase_t * ul_evptrig_get_base (ul_evptrig_t * evptrig);
```

Arguments

evptrig

event trigger

ul_evpoll_new

Name

ul_evpoll_new — create new event base

Synopsis

```
ul_evpbase_t * ul_evpoll_new (const ul_evpoll_ops_t * ops, int flags);
```

Arguments

ops

pointer to preferred mechanism/operations set

flags

set of option flags

ul_evpoll_destroy

Name

`ul_evpoll_destroy` — destroy base and inform all attached triggers

Synopsis

```
void ul_evpoll_destroy (ul_evpbase_t * base);
```

Arguments

base

event poll base

ul_evpoll_update

Name

`ul_evpoll_update` — mostly reserve for some mechanisms requiring update calls

Synopsis

```
int ul_evpoll_update (ul_evpbase_t * base);
```

Arguments

base

event poll base

ul_evpoll_dispatch

Name

`ul_evpoll_dispatch` — start single iteration of the wait and process events cycle

Synopsis

```
int ul_evpoll_dispatch (ul_evpbase_t * base, ul_htim_diff_t * timeout);
```

Arguments

base

event poll base

timeout

pointer relative time maximal wait interval or `NULL` - forever

ul_evpoll_get_current_time

Name

`ul_evpoll_get_current_time` — get current time in given base epoch and units

Synopsis

```
ul_htim_time_t ul_evpoll_get_current_time (ul_evpbase_t * base);
```

Arguments

base

event poll base

ul_evpoll_loop

Name

`ul_evpoll_loop` — run event loop as long as required

Synopsis

```
int ul_evpoll_loop (ul_evpbase_t * base, int flags);
```

Arguments

base

event poll base

flags

none defined yet, provide 0

ul_evpoll_quilt_loop

Name

`ul_evpoll_quilt_loop` — mark event loop to terminate before next iteration

Synopsis

```
int ul_evpoll_quilt_loop (ul_evpbase_t * base);
```

Arguments

base

event poll base

ul_evpoll_cascade

Name

`ul_evpoll_cascade` — cascade base or event triggers attached to it onto another base

Synopsis

```
int ul_evpoll_cascade (ul_evpbase_t * base, ul_evpbase_t * new_base, int
prioshift, int bc_flags);
```

Arguments

`base`

event poll base which should be part of event processing of *`new_base`*

`new_base`

upper level base which will include *`base`* if operation succeed

`prioshift`

possible priority shift - not implemented yet

`bc_flags`

combination of `UL_EVP_CASFL_INHERIT_DESTROY`, `UL_EVP_CASFL_PROPAGATE_DESTROY`