NAME

guasi_create, guasi_free, guasi_submit, guasi_fetch, guasi_req_cancel, guasi_req_info, guasi_req_free

guasi__open, guasi__read, guasi__write, guasi__pread, guasi__pwrite, guasi__sendfile, guasi__opendir, guasi__readdir, guasi__stat, guasi__fstat, guasi__recv, guasi__recvfrom, guasi__send, guasi__sendto, guasi__accept, guasi__connect, guasi__fsync

SYNOPSIS

Core API

```
#include <guasi.h>

guasi_t guasi_create(int min_threads, int max_threads, int max_priority);
void guasi_free(guasi_t hctx);
guasi_req_t guasi_submit(guasi_t hctx, void *priv, void *asid, int prio,
                         guasi_syscall_t proc, guasi_cleanup_t free, int nparams, ...);
int guasi_fetch(guasi_t hctx, guasi_req_t *reqs, int minreqs,
                int maxreqs, long timeo);
int guasi_req_cancel(guasi_req_t hreq);
int guasi_req_info(guasi_req_t hreq, struct guasi_reqinfo *rinf);
void guasi_req_free(guasi_req_t hreq);
```

POSIX wrappers:

```
#include <guasi_syscalls.h>

guasi_req_t guasi__open(guasi_t hctx, void *priv, void *asid, int prio,
                        char const *name, long flags, long mode);
guasi_req_t guasi__read(guasi_t hctx, void *priv, void *asid, int prio,
                        int fd, void *buf, size_t size);
guasi_req_t guasi__write(guasi_t hctx, void *priv, void *asid, int prio,
                         int fd, void const *buf, size_t size);
guasi_req_t guasi__pread(guasi_t hctx, void *priv, void *asid, int prio,
                         int fd, void *buf, size_t size, goff_t off);
guasi_req_t guasi__pwrite(guasi_t hctx, void *priv, void *asid, int prio,
                          int fd, void const *buf, size_t size, goff_t off);
guasi_req_t guasi__sendfile(guasi_t hctx, void *priv, void *asid, int prio,
                            int ofd, int ifd, goff_t *off, size_t cnt);
guasi_req_t guasi__opendir(guasi_t hctx, void *priv, void *asid, int prio,
                           char const *name);
guasi_req_t guasi__readdir(guasi_t hctx, void *priv, void *asid, int prio,
                           DIR *dir);
guasi_req_t guasi__stat(guasi_t hctx, void *priv, void *asid, int prio,
                        char const *name, struct stat *sbuf);
guasi_req_t guasi__fstat(guasi_t hctx, void *priv, void *asid, int prio,
                         int fd, struct stat *sbuf);
guasi_req_t guasi__recv(guasi_t hctx, void *priv, void *asid, int prio,
                        int fd, void *buf, size_t size, int flags,
                        long timeo);
guasi_req_t guasi__recvfrom(guasi_t hctx, void *priv, void *asid, int prio,
                            int fd, void *buf, size_t size, int flags,
                            struct sockaddr * from, socklen_t * fromlen, long timeo);
guasi_req_t guasi__send(guasi_t hctx, void *priv, void *asid, int prio,
                        int fd, void const *buf, size_t size, int flags,
                        long timeo);
```

guasi_req_t guasi__sendto(guasi_t *hctx***, void *** *priv***, void *** *asid***, int** *prio***,**
int *fd***, void const *** *buf***, size_t** *size***, int** *flags***,**
struct sockaddr const * *to***, socklen_t** *tolen***, long** *timeo***);**
guasi_req_t guasi__accept(guasi_t *hctx***, void *** *priv***, void *** *asid***, int** *prio***,**
int *fd***, struct sockaddr *** *addr***, socklen_t *** *addrlen***,**
long *timeo***);**
guasi_req_t guasi__connect(guasi_t *hctx***, void *** *priv***, void *** *asid***, int** *prio***,**
int *fd***, struct sockaddr const *** *addr***, socklen_t** *addrlen***,**
long *timeo***);**
guasi_req_t guasi__fsync(guasi_t *hctx***, void *** *priv***, void *** *asid***, int** *prio***,**
int *fd***);**

DESCRIPTION

The **guasi** library implements a thread based generic asyncronous execution engine, to be used to give otherwise syncronous calls an asyncronous behaviour. It can be used to wrap any syncronous call, so that it can be scheduled for execution, and whose result can be fetched at later time (hence not blocking the submitter thread). The **guasi** library can be used as complement to standard event retrieval interfaces like **poll**(2), **select**(2) or **epoll**(4). The **guasi** library is generic, by meaning that any (not only system calls) otherwise syncronous function can be made asyncronous. Functions submitted with **guasi_submit**(3) will/might execute in parallel, so proper care must be taken to handle this behaviour. Care must be taken to select the proper stack size, according to what the asyncronous functions will require. The default **PTHREAD_STACK_MIN** value is more than enough when executing simple system call wrappers, but it may not be enough if the function submitted with **guasi_submit**(3) uses a lot of local (stack) allocations.

Structures and Types

The following structures are defined:

guasi_param_t

The **guasi_param_t** type is the basic type used to return asyncronous operation results, and to pass parameters to asyncronous callbacks. Its size guarantee that a pointer can be safely passed through it.

guasi_syscall_t

The **guasi_syscall_t** type is a function pointer type, that is used to pass asyncronous functions to **guasi_submit**. An asyncronous function looks like:

```
guasi_param_t async_proc(void *priv, guasi_param_t const *params) {
        guasi_param_t result;

        ...
        return result;
}
```

guasi_cleanup_t

The **guasi_cleanup_t** type is a function pointer type, that is used to pass to **guasi_submit** to give the caller to perform a cleanup of the parameters passed to **guasi_submit** itself. A cleanup function looks like:

```
void cleanup_proc(void *priv, guasi_param_t const *params) {
        ...
```

```
}
```

struct guasi_reqinfo

```
struct guasi_reqinfo {
        void *priv;
        void *asid;
        guasi_param_t result;
        long error;
        int status;
};
```

The **struct guasi_reqinfo** describes the status of a request submitted to a **guasi** handle. The **guasi_req_info** function can be used to retrieve such information from a request handle (**guasi_req_t**). The *priv* member is the same *priv* parameter that is passed to the **guasi_submit** function when the request was submitted. The *asid* is the request identifier, passed to **guasi_submit**. The *result* member is the result of the asyncronous operation, that is in turn, the value returned by the *proc* parameter of the **guasi_submit** function. The *error* member if the value or the C library **errno** after the *proc* asyncronous function execution. The *status* is the current status of the request. Possible values for the *status* member are:

GUASI_STATUS_PENDING
The request has still to be dequeued.

GUASI_STATUS_ACTIVE
The request is executing.

GUASI_STATUS_COMPLETE
The request has completed.

GUASI_STATUS_CANCELED
The request has been canceled.

The *result* and *error* members are undefined until the request *status* reaches the **GUASI_STATUS_COMPLETE** value.

Functions (Core API)

The following functions are defined:

guasi_t guasi_create(int *min_threads***, int** *max_threads***, int** *max_priority***);**

The **guasi_create** function creates a **guasi** handle to be used as a gateway for all the following **guasi** operations. The *min_threads* parameter specifies the minimum number of threads to be used in the **guasi** thread pool, and the *max_threads* parameter specifies the maximum number of threads (zero means unlimited). Since **guasi** requests can have different priorities, the *max_priority* parameter specifies the number of priorities that the new **guasi** handle must support. The function returns the new **guasi** handle if succeeded, or **NULL** if failed.

void guasi_free(guasi_t *hctx***);**

The **guasi_free** function frees all the resources associated with the **guasi** handle passed in *hctx*.

guasi_req_t guasi_submit(guasi_t *hctx***, void *** *priv***, void *** *asid***, int** *prio***,**
guasi_syscall_t *proc***, guasi_cleanup_t** *free***, int** *nparams***, ...);**

The **guasi_submit** function submits a new request to the **guasi** handle passed in *hctx*. The *priv* parameter is an opaque value that is passed as is to the *proc* asyncronous function. The *free* function, if not **NULL**, will be called to give the caler the chance to cleanup the parameters. The *asid* is a cookie that identifies the request, and is returned by the **guasi_req_info** function, inside the **struct guasi_reqinfo** strcture. The priority of the request is passed in the *prio* parameter, that should range from 0 to (*max_priority* - 1). The *nparams* parameter specifies the number of arguments that follows and that will be passed to the *proc* asyncronous function. The function returns a request handle if succeeded, or **NULL** if failed.

int guasi_fetch(guasi_t *hctx***, guasi_req_t ***reqs***, int** *minreqs***,**
int *maxreqs***, long** *timeo***);**

The **guasi_fetch** function retrieves completed requests from the **guasi** handle passed in *hctx*. Not less than *minreqs*, and up to *maxreqs* requests are retrieved, and stored inside the *reqs* pointer. The *timeo* represent a maximum time (in milliseconds) to wait for some request to complete. If *timeo* is negative, the wait for completed requests will not have a time limit. The function returns the number of completed requests, or a negative number in case of error.

int guasi_req_cancel(guasi_req_t *hreq***);**

The **guasi_req_cancel** function cancels a pending request specified in the *hreq* parameter. The function returns the current status of the request, or a negative number in case of error. A canceled request will be available at the next **guasi_fetch**() call, with **guasi_req_info**() returning the proper status. If the reported status is **GUASI_STATUS_CANCELED**, it means that the request has been canceled before having the chance to be executed. If the reported status is **GUASI_STATUS_COMPLETE**, the caller should check the *error* member of the **struct guasi_reqinfo** structure to see if the request has fully completed before we canceled, or if if has been interrupted while executing (in which case *error* should hold the **EINTR** value - at least for system call wrappers).

int guasi_req_info(guasi_req_t *hreq***, struct guasi_reqinfo ***rinf***);**

The **guasi_req_info** functions retrieves information about the request passed in the *hreq* parameter. The request information will be stored inside the *rinf* **struct guasi_reqinfo** pointer. The function returns 0 is case of success, or a negative number in case of error.

void guasi_req_free(guasi_req_t *hreq***);**

The **guasi_req_free** function frees all the resources associated with the request handle passed in *hreq*. A request can be freed only after it has been returned by the **guasi_fetch**() function.

Functions (POSIX wrappers)

All the POSIX wrappers functions implemented by **guasi**, map to the corresponding POSIX function in terms of parameters and result type. Some networking functions can have an extra *timeo* parameter, that can be used to specify a timeout (in milliseconds) for the operation.

PERFORMANCE

One software that allows you to compare **guasi** performance with other kinds of asyncronous I/O providers, is **FIO**:

http://freshmeat.net/projects/fio/

Performance results of **guasi** against **libaio** using **FIO** are reported in this page:

http://www.xmailserver.org/guasi-libaio-fio.html

LICENSE

AUTHOR

Developed by Davide Libenzi <**davidel@xmailserver.org**>

AVAILABILITY

The latest version of the **guasi** library can be found at:

http://www.xmailserver.org/guasi-lib.html

BUGS

There are no known bugs. Bug reports and comments to Davide Libenzi <**davidel@xmailserver.org**>