MACH Environment Manager

Mary R. Thompson

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Version of:
4 August 1989

Abstract

The Environment Manager is a Mach server which facilities the sharing of named variables between tasks.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4864, monitored by the Space and Naval Warfare Systems Command under contract N00039-84-C-0467.

1. Introduction

The Environment Manager is a server which facilitates the sharing of named variables between tasks. An environment is a set of named variables, which can be read or changed via calls on an *Environment Port*. A single environment may be shared between parent and child tasks, or an environment can be copied, and a copy can be passed to a child task. It is also possible to get a *read-only* port to an environment which allows reading but not modification of the environment.

The Environment Manager stores two types of named objects: strings and ports. The names of these two types may not overlap, and each type must be set and retrieved by the appropriately typed primitives. Sets of variables (called an environment) are accessible through a specific server port. There may be one read/write and one read-only port to the same environment.

Environments are passed to child tasks in one of two ways. Either the parent's `environment_port` is passed to the child in which case the two processes share the same environment with both having equal write access to all variables; or the parent clones his environment and passes a copy of it to the child. In the latter case the two environments are then completely disjoint. It is also possible to create a new empty (or default) environment.

It is also possible for one task to use more than one environment. In this way a task could have access to a widely-shared "global" environment as well as its own local environment.

2. Types

The following types are defined as smallish fixed length strings in order to make the passing of them as in-line message data efficient.

The following types can be included in C programs from `<servers/emdefs.h>`.

```
#define env_name_size  (80)
#define env_val_size   (256)

typedef char env_name_t[env_name_size];   /* environment variable name */
typedef char env_str_val_t[env_val_size]; /* string env variable value */

typedef env_name_t     *env_name_list;  /* list of names */
typedef env_str_val_t  *env_str_list;   /* list of string values */
```

3. Primitives

The following primitives are provided:

env_string

```
#include <servers/env_mgr.h>

kern_return_t env_get_string(env_port,env_name,env_val)
    port_t              env_port;
    env_name_t          env_name;
    env_str_val_t       env_val;

kern_return_t env_set_string(env_port,env_name,env_val)
    port_t              env_port;
    env_name_t          env_name;
    env_str_val_t       env_val;

kern_return_t env_del_string(env_port,env_name)
    port_t              env_port;
    env_name_t          env_name;
```

Arguments

`env_port`	port identifying environment
`env_name`	name of string to be found | set | deleted
`env_val`	returned pointing to value of string

Description

`env_get_string` returns the value of the string variable with the name `env_name`.

`env_set_string` sets the string variable `env_name` to `env_val`;

`env_del_string` deletes the string variable `env_name`.

Returns

`KERN_SUCCESS`	operation succeeded
`ENV_UNKNOWN_PORT`	`env_port` does not reference a known environment
`ENV_VAR_NOT_FOUND`	name does not exist.
`ENV_WRONG_VAR_TYPE`	names exists, but is a port variable
`ENV_READ_ONLY`	`env_port` only allows read access to the environment. (`env_set_string`, `env_del_string`)

See Also

`env_port[3]`, `env_conn[3]`, `env_list[3]`

env_port

```
#include <servers/env_mgr.h>

kern_return_t env_get_port(env_port,env_name,env_val)
    port_t              env_port;
    env_name_t          env_name;
    port_t              *env_val;

kern_return_t env_set_port(env_port,env_name,env_val)
    port_t              env_port;
    env_name_t          env_name;
    port_t              env_val;

kern_return_t env_del_port(env_port,env_name)
    port_t              env_port;
    env_name_t          env_name;
```

Arguments

`env_port`	port identifying environment
`env_name`	name of port to be found **|** set **|** deleted
`env_val`	returned pointing to value of port

Description

`env_get_port` returns the value of the port variable with the name `env_name`.

`env_set_port` sets the value of the port variable, `env_name` to `env_val`.

`env_del_port` deletes the port `env_name`;

Returns

`KERN_SUCCESS`	variable found
`ENV_UNKNOWN_PORT`	`ENV_PORT` does not reference a known environment
`ENV_VAR_NOT_FOUND`	name does not exist.
`ENV_WRONG_VAR_TYPE`	names exists, but is a string variable
`ENV_READ_ONLY`	`env_port` only allows read access to the environment. (`env_set_port` and `env_del_port`)

See Also

`env_string[3], env_conn[3], env_list[3]`

env_list

```
#include <servers/env_mgr.h>

kern_return_t env_list_strings(env_port,env_names,name_cnt,
                               env_string_vals,string_cnt)
    port_t          env_port;
    env_name_list  *env_names
    int            *name_cnt;
    env_str_list   *env_string_vals;
    int            *string_cnt);
kern_return_t env_list_ports(env_port,env_names,name_cnt,
                               env_port_vals,port_cnt)
    port_t          env_port;
    env_name_list  *env_names
    int            *name_cnt;
    port_array_t   *env_port_vals;
    int            *port_cnt);
kern_return_t env_set_stlist(env_port,env_names,name_cntf,
                             env_string_vals,string_cnt)
    port_t          env_port;
    env_name_list   env_names
    int             name_cnt;
    env_str_list    env_string_vals;
    int             string_cnt);
kern_return_t env_set_ptlist(env_port,env_names,name_cnt,
                             env_port_vals,port_cnt)
    port_t          env_port;
    env_name_list   env_names;
    int             name_cnt;
    port_array_t    env_port_vals;
    int             port_cnt);
```

Arguments

`env_port`	port identifying environment
`env_names`	pointer to list of names of all string variables
`name_cnt`	number of names
`env_string_vals`	pointer to values of string variables (for `env_list_strings`) list of values of string variables (for `env_set_stlist`)
`string_cnt`	number of string values (equal to `name_cnt`)
`env_port_vals`	pointer to values of port variables (for `env_list_ports`) list of values of port variables (for `env_set_ptlist`)
`port_cnt`	number of port values (equal to `name_cnt`)

Description

`env_list_string` returns a complete list of all the string variables in the environment specified by `env_port`. The two arrays `env_names` and `env_string_vals` are returned in newly allocated virtual memory. This memory should be released by a call to `vm_deallocate` once the items are no longer needed.

`env_list_ports` returns a complete list of all the port variables in the environment specified by `env_port`. The two arrays `env_names` and `env_port_vals` are returned in newly allocated virtual memory. This memory should be released by a call to `vm_deallocate` once the items are no longer needed.

`env_set_stlist` sets a number of string variables environment specified by `env_port`. This primitive is provided for efficiency and is mainly intended to be used to set a Mach environment to be the same as the Unix **environ** area.

`env_set_ptlist` sets a number of port variables environment specified by `env_port`. This primitive is provided for efficiency.

Returns

`KERN_SUCCESS` operation succeeded

`ENV_UNKNOWN_PORT`
`env_port` does not reference a known environment

`ENV_READ_ONLY` `env_port` only allows read access to the environment (for env_set_stlist and env_set_plist).

`ENV_WRONG_VAR_TYPE`
one of the variables was already defined as a port variable (for env_set_stlist) or as as string variable (for env_set_ptlist).

See Also

`env_string[3], env_port[3], env_conn[3]`

env_conn

```
#include <servers/env_mgr.h>

void init_env_mgr(reply_port);
    port_t      reply_port;

kern_return_t env_new_conn(env_port,new_env_port)
    port_t      env_port;
    port_t      *new_env_port)

kern_return_t env_copy_conn(env_port,new_env_port)
    port_t      env_port;
    port_t      *new_env_port)

kern_return_t env_restrict_conn(env_port,new_env_port)
    port_t      env_port;
    port_t      *new_env_port)

kern_return-t env_disconnect(env_port)
    port_t      env_port;
```

Arguments

`reply_port`	if equal `PORT_NULL`, a reply port will be allocated, otherwise `reply_port` will be used by the interface to receive the message replies.
`env_port`	port identifying environment
`new_env_port`	returned pointing to value of new port

Description

`init_env_mgr` initializes the user interface to the Environment Manager. Must be called before any of the other primitives are used. However,it is called by the library initialization program `mach_init`, so the user does not need to make this call unless a different value of `Reply_port` is desired.

`env_new_conn` create a new, default environment to be handed to a new process. This could be initialized with public values such as host_name and Network nameserver port.

`env_copy_conn` makes a complete copy of the environment specified by `env_port`, allocates the `new_env_port` and returns it to the caller. Subquently, all requests on `new_env_port` will use the new copy, and requests on `env_port` will continue to refer to the original version. Used by parent to pass a copy of its environment to a child process.

`env_restrict_conn` makes a new port to the environment specified by `env_port`, through which only reading will be allowed.

`env_disconnect` informs the environment manager that this enviroment is no longer needed. EnvMgr will deallocate `env_port`;

Ownership rights to all the new environment ports are returned to the user. Thus when a process that has created and environment dies, these rights are returned to the environment manager who will disconnect (and destroy) the environment. If a creator process wishes its environment to live on after its death, it must pass the ownership rights a process that will continue to exist as long as the environment should exist.

Returns

KERN_SUCCESS new_env_port references a new default environment

ENV_NO_MORE_CONN
implementation restriction, no more connections to the Environment Manager are available.

ENV_UNKNOWN_PORT
env_port does not reference a known environment

See Also

env_string[3], env_port[3], env_list[3], mach_init[3]

4. Integration with Unix environ

In order to allow binary compatibility with Unix, the `environ` area will have to be maintained and passed on as usual by `execve`. If `execve` also clones the current Mach environment and passes the new port on to the child, then the Mach environment will have the same copy semantics as the Unix `environ`. If `execve` were to enter all the variables in `environ` into the Mach environment then new programs could get all their variables from the Mach environment. The primitive `env_set_slist` is provided to allow a set of environment string variables to be entered with one message.

If `getenv` is changed to call `get_env_string`, after failing to find a variable in the `environ` area, then old programs will be able to find environment variables set by new programs in the Mach environment. The `environment_port` will be passed to a new task as part of the few special ports that all processes know about.

At some point, the shells will have to be changed to use the Mach environment in addition to the `environ` area. Existing Unix variables could be entered in both places while Mach variables would be put in the Mach environment only.

I. Summary of Calls

The following is a summary of the C calls to the Environment Manger. The page on which the operation is fully described appears within square brackets.

```
[2]   kern_return_t env_get_string(env_port,env_name,env_val)
          port_t              env_port;
          env_name_t          env_name;
          env_str_val_t       env_val;


[2]   kern_return_t env_set_string(env_port,env_name,env_val)
          port_t              env_port;
          env_name_t          env_name;
          env_str_val_t       env_val;


[2]   kern_return_t env_del_string(env_port,env_name)
          port_t              env_port;
          env_name_t          env_name;


[3]   kern_return_t env_get_port(env_port,env_name,env_val)
          port_t              env_port;
          env_name_t          env_name;
          port_t              *env_val;


[3]   kern_return_t env_set_port(env_port,env_name,env_val)
          port_t              env_port;
          env_name_t          env_name;
          port_t              env_val;


[3]   kern_return_t env_del_port(env_port,env_name)
          port_t              env_port;
          env_name_t          env_name;


[4]   kern_return_t env_list_strings(env_port,env_names,name_cnt,
                                     env_string_vals,string_cnt)
          port_t          env_port;
          env_name_list *env_names
          int           *name_cnt;
          env_str_list  *env_string_vals;
          int           *string_cnt);


[4]   kern_return_t env_list_ports(env_port,env_names,name_cnt,
                                     env_port_vals,port_cnt)
          port_t          env_port;
          env_name_list *env_names
          int           *name_cnt;
          port_array_t  *env_port_vals;
          int           *port_cnt);


[4]   kern_return_t env_set_stlist(env_port,env_names,name_cntf,
                                   env_string_vals,string_cnt)
          port_t          env_port;
          env_name_list   env_names
          int             name_cnt;
```

```
        env_str_list   env_string_vals;
        int            string_cnt);


[4]    kern_return_t env_set_ptlist(env_port,env_names,name_cnt,
                                   env_port_vals,port_cnt)
        port_t         env_port;
        env_name_list  env_names;
        int            name_cnt;
        port_array_t   env_port_vals;
        int            port_cnt);


[6]    void init_env_mgr(reply_port);
        port_t      reply_port;


[6]    kern_return_t env_new_conn(env_port,new_env_port)
        port_t      env_port;
        port_t      *new_env_port)


[6]    kern_return_t env_copy_conn(env_port,new_env_port)
        port_t      env_port;
        port_t      *new_env_port)


[6]    kern_return_t env_restrict_conn(env_port,new_env_port)
        port_t      env_port;
        port_t      *new_env_port)


[6]    kern_return-t env_disconnect(env_port)
        port_t      env_port;
```

Table of Contents