

# 1 Installing the Service

The service runs on any type of POSIX compliant Unix/Linux. It has been successfully tested with Sun Solaris, RedHat Linux, and SuSe Linux. The network in which the service is to be installed has to allow IP-multicast message between the machines running the service. Clock synchronization is not mandatory, but is highly recommended to achieve better performance. Here is how to install the service (replace *service-robust* by *service-scalable* throughout this section if you want to use the most scalable version of the service):

1. Decompress the tar archive of the source distribution: `unzip service-robust.zip`.
2. Compile the service and the shared library: `gmake all`.
3. (Optional step) Install the service executable and the shared library (you have to have root access): `gmake install`.

If you want to clean up the object files created during the compilation, execute the command `gmake clean` and if you want to uninstall the service, do a `gmake uninstall`.

A few flags are used to configure the service during compilation, comment out the flag if you do not want a specific feature to be enabled. These flags are found in the file `src/Makefile` and have the following meaning:

- **SYNCH\_CLOCKS** tells the service that the clocks are synchronized. Do not use this flag if they are not, as it could violate the required FD QoS.
- **REALTIME** can be used only if you run the service as root. This flag makes the service run at the highest possible priority and locks the pages of the service in RAM (no disk swapping will occur).
- **OUTPUT** enables verbose output for the failure detector service.
- **LOG** writes verbose information into a log file named `Fdd-<hostname>-<start_time>`.
- **OMEGA\_OUTPUT** same thing as **OUTPUT**, but for the leader election algorithm.
- **OMEGA\_LOG** same thing as **LOG**, but writes information in another file named `omega_log`.

**Using the shared library:** In order for an external program named `app.c` to use the service, two things have to be done. First, the file `service-robustlib.h` has to be included in `app.c`. Second, `app` has to be compiled and linked with the shared library as follows: `gcc -o app app.c -lservice-robust`.

## 2 The Interface of the Service

The leader election service provides an easy to use and flexible interface. The two versions of the service, service-robust and service-scalable, share the same interface. In Figure 1, the full interface is shown. We now explain how the interface is to be used. First note that all functions return an integer. Except for `omega_register`, this integer tells whether the function completed successfully, i.e., if it is positive (including 0), everything went fine, if it is negative, an error occurred.

---

```
int omega_register(unsigned int pid)

int omega_unregister(int omega_int)

int omega_startOmega(int omega_int, unsigned int gid, int candidate,
                    int notif_type, unsigned int TdU, unsigned int TmU,
                    unsigned int TmrL)

int omega_stopOmega(int omega_int, unsigned int gid)

int omega_parse_notify(int omega_int, struct omega_proc_struct *leader)

int omega_getleader(int omega_int, unsigned int gid,
                   struct omega_proc_struct *leader)

int omega_interrupt_any_change(int omega_int, unsigned int gid)

int omega_interrupt_none(int omega_int, unsigned int gid)
```

---

Figure 1: Interface of the Leader Election Service

In order to use the service, a process  $p$  must first register itself using a unique process identifier. This is done using the `omega_register` function. Upon successful completion, this function returns a pipe file descriptor that will be used in all the subsequent interface calls (this file descriptor is denoted as `omega_int` in Figure 1). In particular, it is through this pipe that  $p$  will receive the new group leader's identity, whenever it changes. The `omega_unregister` function is used to unregister  $p$ .

After having successfully registered itself with the service,  $p$  can join/leave groups at anytime using the `omega_startOmega` and `omega_stopOmega` calls respectively. To join a group,  $p$  must specify seven parameters:

1. The pipe file descriptor `omega_int`
2. The group identifier `gid`
3. Whether  $p$  wants to be a candidate for leadership (`candidate = 1`) or not (`candidate = 0`)

4. The way  $p$  wants to find out the group leader identity, either by an *interrupt*, whenever the group leader changes (`notif_type = 1`) or by a *query*, using the `omega_getleader` function, whenever  $p$  wants to do so (`notif_type = 0`)
5. The upper bound on the time it takes to detect the crash of a process in the group,  $T_D^U$  (c.f. [1])
6. The lower bound on the average time frame during which a process in the group is wrongly suspected,  $T_m^U$  (c.f. [1])
7. The lower bound on the average time frame between two mistakes,  $T_{mr}^L$  (c.f. [1])

Depending on whether  $p$  asked to learn about the group leader's identity by interrupt or by query,  $p$  will respectively use the functions `omega_parse_notify` or `omega_getleader`. The first function returns the group leader in a structure called `omega_proc_struct`:

```

struct omega_proc_struct {
    struct sockaddr_in addr;
    unsigned int pid;
    unsigned int gid;
    int leader_stable;
};

```

The fields of this structure are respectively (from top to bottom): the leader's IP address, the leader's process identifier, the leader's group identifier, and an integer telling whether several processes are currently competing for the leadership or not. If `leader_stable = 0`,  $p$  should be careful about using the leader for it might be demoted in a short period of time. Note that this function blocks  $p$  until the group leader changes. Moreover, this function does not take a group identifier as parameter because  $p$  is notified of leader changes in all groups it has joined. The second function, `omega_getleader`, is not blocking, i.e., it returns the current leader of the group `gid` passed in parameter.

Finally, the last two procedures, `omega_interrupt_any_change` and `omega_interrupt_none`, are used to change the leader notification type of a given group.

## References

- [1] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on computers*, 51(5):561–580, May 2002.