Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3
Ejercicios de paso de mensajes

Sistemas Distribuidos
Grado en Ingeniería Informática

CC BY NC SA

Ejercicio 2

Se desea diseñar un modelo de vector distribuido.
Sobre un vector distribuido se definen los siguientes servicios:

- int **init** (char *nombre, int N). Este servicio permite inicializar un array distribuido de N números enteros. La función devuelve 1 cuando el array se ha creado por primera vez. En caso de que el array ya esté creado, la función devuelve 0. La función devuelve -1 en caso de error.
- int **set** (char *nombre, int i, int valor). Este servicio inserta el valor en la posición i del array nombre.
- int **get** (char*nombre, int i, int *valor). Este servicio permite recuperar el valor del elemento i del array nombre.

Diseñe un sistema distribuido que implemente el servicio con colas POSIX de forma que permita trabajar con varios clientes concurrentemente.

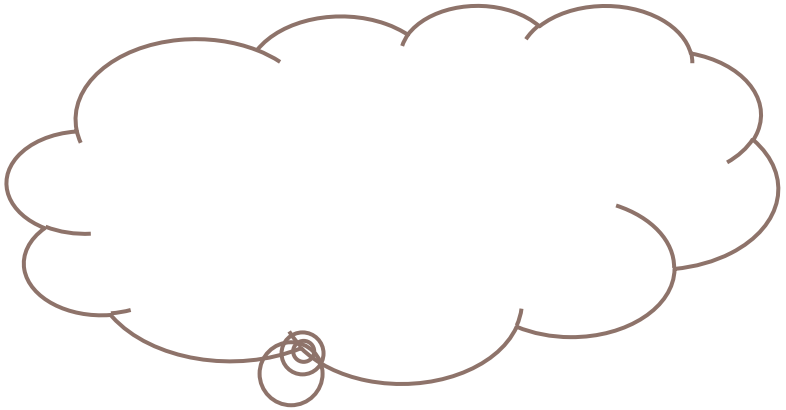
Ejercicio 2

Se desea diseñar un modelo de vector distribuido.
Sobre un vector distribuido se definen los siguientes servicios:

- int **init** (char *nombre, int N). Este servicio permite inicializar un array distribuido de N números enteros. La función devuelve 1 cuando el array se ha creado por primera vez. En caso de que el array ya esté creado, la función devuelve 0. La función devuelve -1 en caso de error.
- int **set** (char *nombre, int i, int valor). Este servicio inserta el valor en la posición i del array nombre.
- int **get** (char*nombre, int i, int *valor). Este servicio permite recuperar el valor del elemento i del array nombre.

Diseñe un **sistema distribuido** que implemente el servicio con **colas POSIX** de forma que permita trabajar con varios clientes **concurrentemente**.

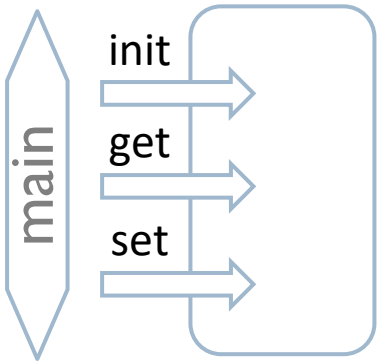
Diseño progresivo



1. Sistema
2. Distribuido
3. Con colas de mensajes POSIX
4. Concurrentes

Diseño NO distribuido (v0.2)

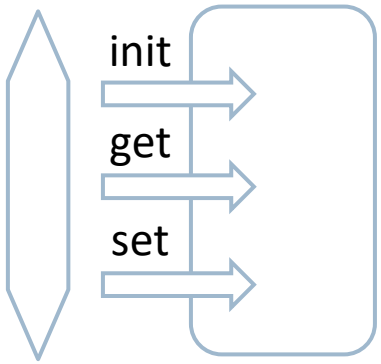
Biblioteca usada desde programa



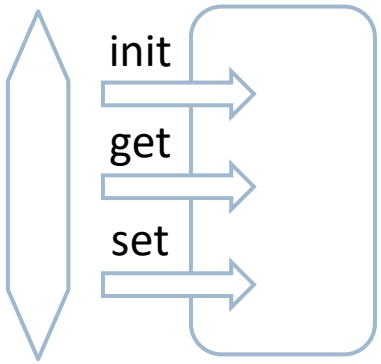
1. **Sistema**
2. Distribuido
3. Con colas de mensajes POSIX
4. Concurrentes

Diseño no distribuido…

```
int      a_neltos = 0 ;
int   * a_values[100] ; // = [ [0…N1], [0…N2], … [0…NN] ] ;
char * a_keys  [100] ; // = [ “key1”,    “key2”,   …   “keyN” ] ;
```



Diseño no distribuido…

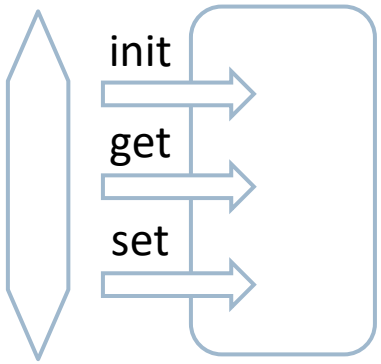



```
int      a_neltos = 0 ;
int   * a_values[100] ; // = [ [0…N1], [0…N2], … [0…NN] ] ;
char * a_keys  [100] ; // = [ "key1",    "key2",   …   "keyN" ] ;
```

```
int buscar (char *nombre)
{
    int index = -1 ;

    for (int i=0; i<a_neltos; i++)
    {
        if (!strcmp(a_keys[i], nombre)) {
            return i;
        }
    }
    return index;
}
```

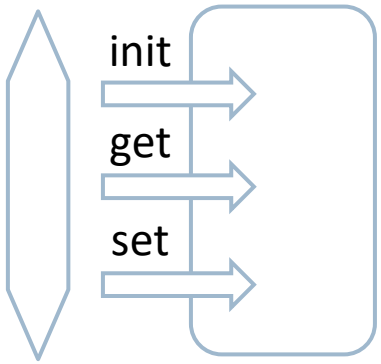
```
int insertar (char  *nombre,  int  N)
{
    a_values[a_neltos] = malloc(N*sizeof(int)) ;
    if (a_values[a_neltos] == NULL) {
        return -1 ; // en caso de error => -1
    }

    a_keys[a_neltos] = strdup(nombre) ;
    if (a_keys[a_neltos] == NULL) {
        free(a_values[a_neltos]);
        return -1 ; // en caso de error => -1
    }

    a_neltos++ ;
    return 1 ; // todo bien => devolver 1
}
```

Diseño no distribuido…



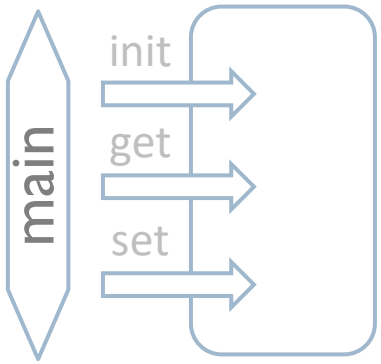

```
// Inicializar un array distribuido de N números enteros.
int init (char  *nombre,  int  N)
{
    int index = buscar(nombre) ;
    if (index != -1) return 0 ;   // Si array ya esté creado => devolver 0

    index = insertar(nombre, N) ;
    if (index == -1) return -1;  // en caso de error => -1

    return 1 ; // el array se ha creado por primera vez => devolver 1
}
```

Diseño no distribuido…




```
// Inserta el valor en la posición i del array nombre.
int set (char *nombre, int i, int valor)
{
    int index = buscar(nombre) ;
    if (index == -1) return -1 ;   // Si error => devolver -1
    a_values[index][i] = valor ;
    return 1;
}

// Recuperar el valor del elemento i del array nombre.
int get (char*nombre, int i, int *valor)
{
    int index = buscar(nombre) ;
    if (index == -1) return -1 ;   // Si error => devolver -1
    *valor = a_values[index][i] ;
    return 1;
}
```

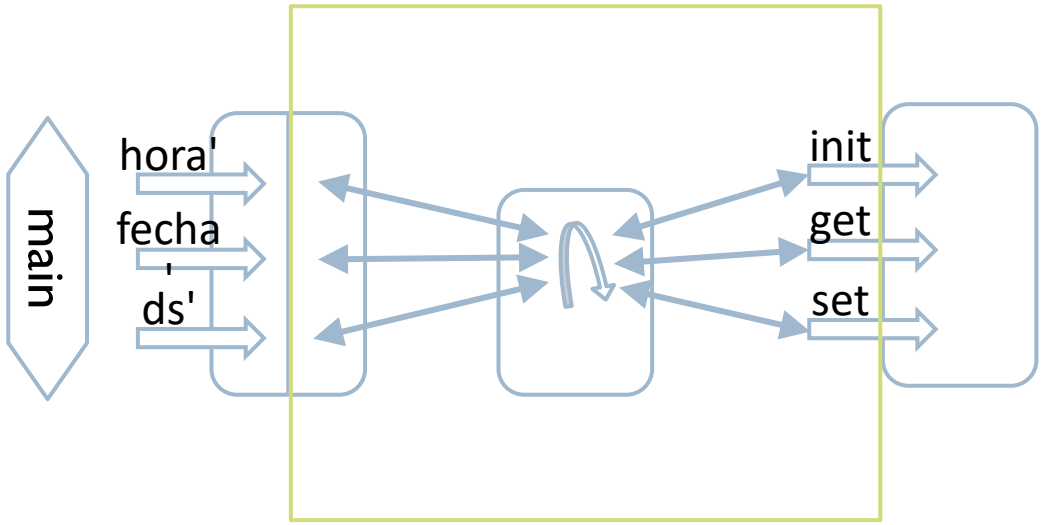
Diseño no distribuido…




```
#include <stdio.h>
#include <stdlib.h>
#include "lib.h"

int    N = 10 ;
char *A = "nombre" ;
int     E  =  1 ;
int    V  = 0x123 ;

int main ( int argc, char *argv[] ) {
    int ret,  val ;

    ret = init(A, N) ;
    if (ret < 0) {  printf("init: error code %d\n", ret); exit(-1);  }
    ret = set (A, E,V) ;
    if (ret < 0) { printf("set: error code %d\n", ret);   exit(-1);   }
    ret = get (A, E, &val) ;
    if (ret < 0) {  printf("get: error code %d\n", ret);   exit(-1);   }
    return 0 ;
}
```

Diseño distribuido (v0.5)

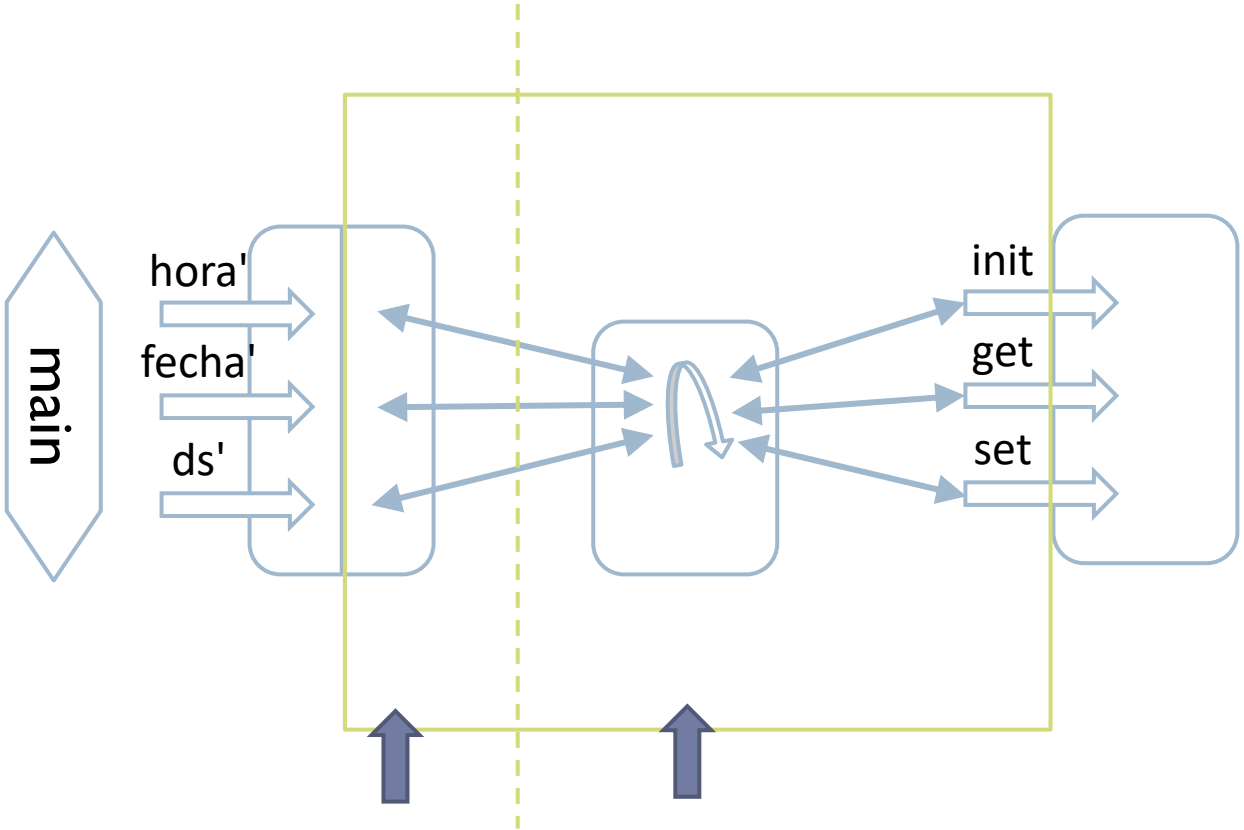
Usar un proxy para los servicios remotos



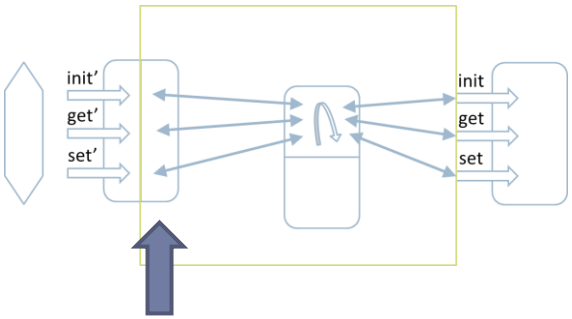
1. Sistema
2. **Distribuido**
3. Con colas de mensajes POSIX
4. Concurrentes

Diseño distribuido (v0.5)

Usar un proxy para los servicios remotos



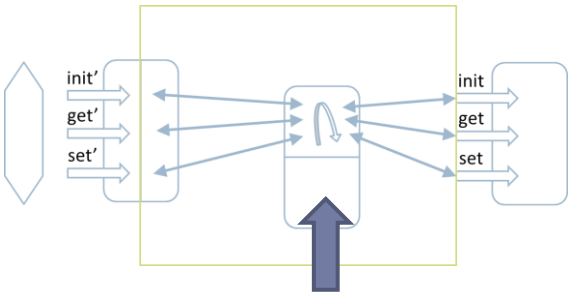
Diseño distribuido…




```
int send_recv ( mensaje *msg )
{
    c1 = "colamsg_conectar" /SERVIDOR
    "colamsg_enviar" c1  msg
    "colamsg_recibir" c1  msg
}

int init (char  *nombre,  int  N)
{
    petición = (init, nombre, N)
    send_recv(petición)
    return respuesta.status
}
```

```
int set (char *nombre, int i, int valor)
{
    petición = (set, nombre, i, valor)
    send_recv(petición)
    return respuesta.status
}

int get (char*nombre, int i, int *valor)
{
    petición = (get, nombre, i)
    send_recv(petición)
    *valor = respuesta.valor
    return respuesta.status
}
```

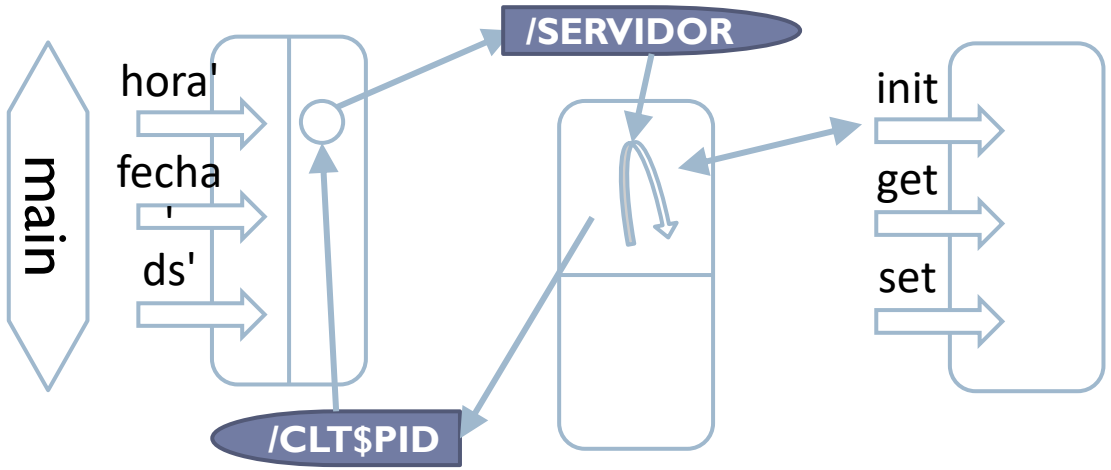
Diseño distribuido…




```
int main ( int argc, char *argv )
{
    c1 = "colamsg_crear" /SERVIDOR

    while (TRUE)
    {
       "colamsg_recibir" c1 petición
        switch( petición.operación)
        {
            case INIT:  respuesta.status = init(petición.nombre, petición.N) ;
                        break;
            case GET:  respuesta.status = set(petición.nombre, petición.i, &respuesta.valor) ;
                        break;
            case SET:   respuesta.status = set(petición.nombre, petición.i, petición.valor) ;
                        break;
       }
       "colamsg_enviar" c1   respuesta
    }
}
```

Diseño distribuido (v0.8)

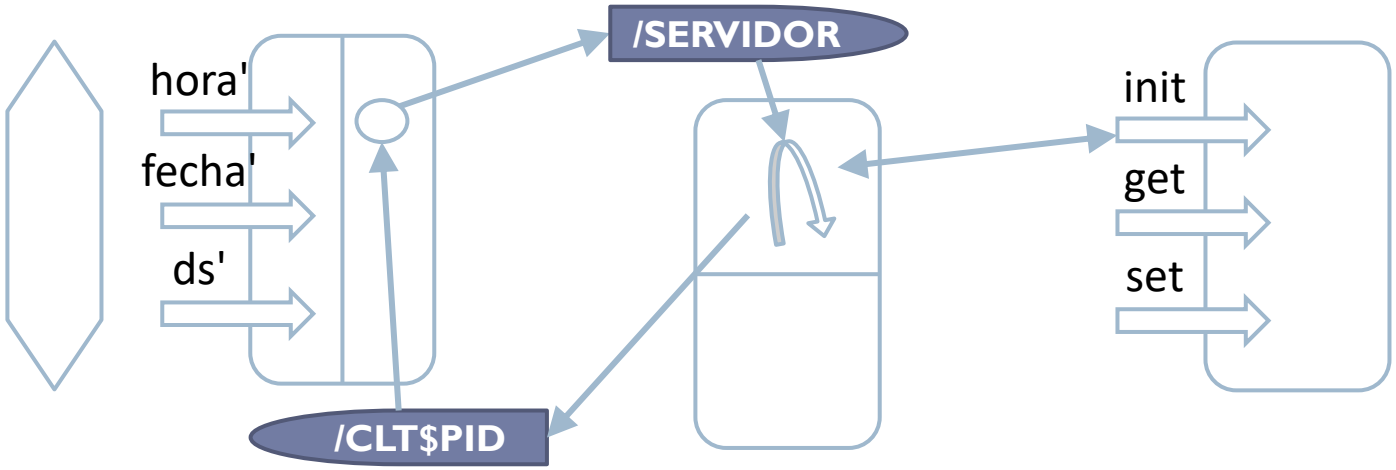
Adaptar proxy a colas de mensajes POSIX



1. Sistema
2. Distribuido
3. **Con colas de mensajes POSIX**
4. Concurrentes

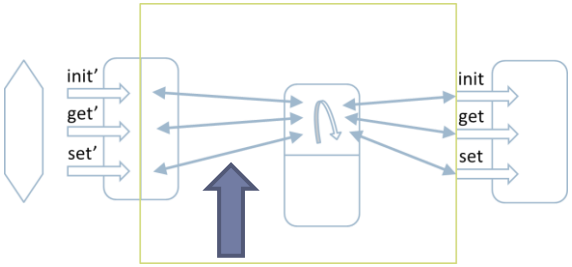
Diseño distribuido (v0.8)

Adaptar proxy a colas de mensajes POSIX



- El mensaje de petición ha de valer para todos los servicios
  - Establecer un identificador numérico para cada servicio
  - Establecer los parámetros para cada servicio y generar una petición la fusión de todos + identificador de servicio.
  - Establecer las respuestas para cada servicio y generar una respuesta fusión.

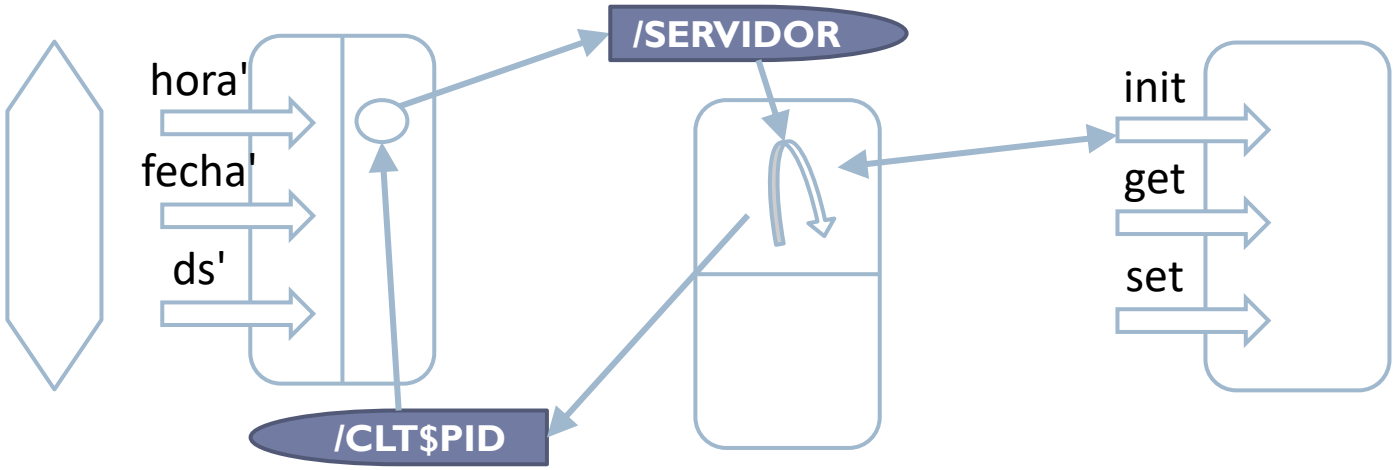
Diseño distribuido…




```
//  petición = op + q_name + (nombre, N) + (nombre, i, valor) + (nombre, i)
struct peticion
{
      int     op;
      char   name[MAX] ;
      int     value;
      int     i;
      char   q_name[MAX];
} ;


// respuesta = (valor, status)
struct respuesta
{
      int    value;
      char   status;
} ;
```

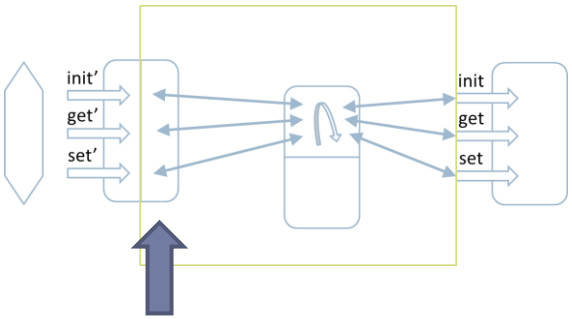
Diseño distribuido (v0.8)

Adaptar proxy a colas de mensajes POSIX



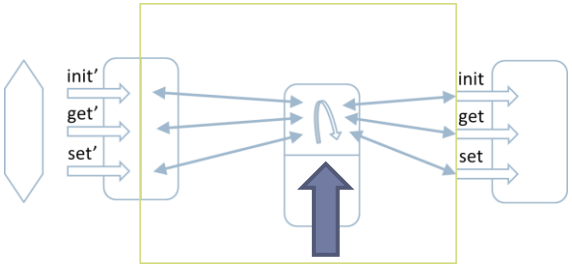
- Las colas POSIX son unidireccionales
  - Una cola general de peticiones creada por el servidor
  - Por cada cliente activo una cola (efímera) para recibir la respuesta.
    - La cola es privada para cada el cliente con nombre único (usar getpid())

Diseño distribuido…



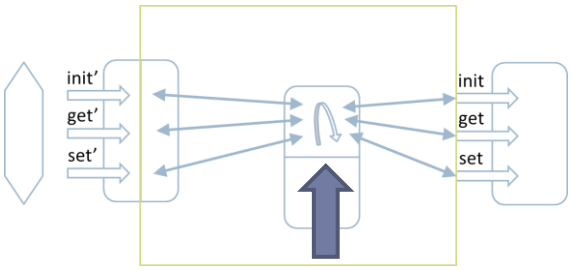

```
int get (char*nombre, int i, int *valor)
{
    struct petición p;
    struct respuesta r;
    char qr_name[1024];  int prio;

    sprintf(qr_name, "%s%d", "/CLIENTE_", getpid()) ;
    int qs = mq_open("/SERVIDOR", O_CREAT|O_WRONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    int qr = mq_open(qr_name, O_CREAT|O_RDONLY) ;
    if (qr == -1) { mq_close(qs) ; return -1; }
    p.op = 2; p.i = i;
    strcpy(p.nombre, nombre);
    strcpy(p.q_name, qr_name);
    mq_send(qs,  (char *)&p,  sizeof(struct petición),    0) ;
    mq_recv(qr,   (char *)&r,  sizeof(struct respuesta),  &prio) ;
    mq_close(qs); mq_close(qr);
    mq_unlink(qr_name);
    *valor = r.value ;
    return r.status ;
}
```

Diseño distribuido…




```
int main ( int argc, char *argv[] )
{
    struct petición p;
    int prio;

    int qs = mq_open("/SERVIDOR", O_CREAT | O_RDONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    while (1)
    {
        mq_recv(qs, &p, sizeof(p), &prio) ;
        tratar_petición(&p) ;
    }
}
```

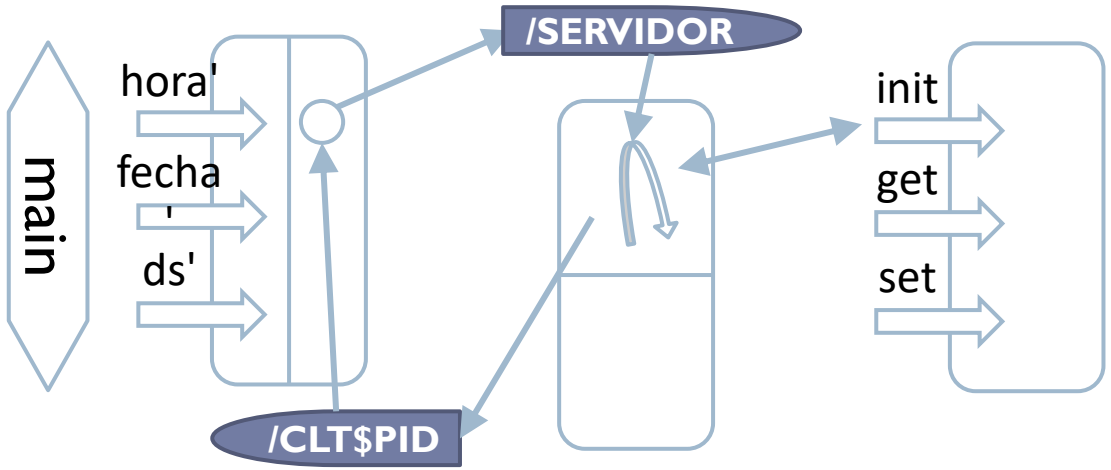
Diseño distribuido…




```
void tratar_petición ( struct petición * p )
{
    struct respuesta r ;
    switch (p->op)
    {
        case 0: // INIT
                    r.status = real_init(p->name, p->value) ;
                    break ;
        case 2: // GET
                    r.status = real_get(p->name, p->i, &(r.value)) ;
                    break ;
        case 3: // SET
                    r.status = real_set(p->name, p->i, p->value) ;
                    break ;
    }

    int qs = mq_open(p->q_name, O_CREAT|O_WRONLY, 0700, NULL) ;
    mq_send(qr, &r, sizeof(struct respuesta), prio) ;
    mq_close(qr);
}
```

Diseño distribuido (v1.0)

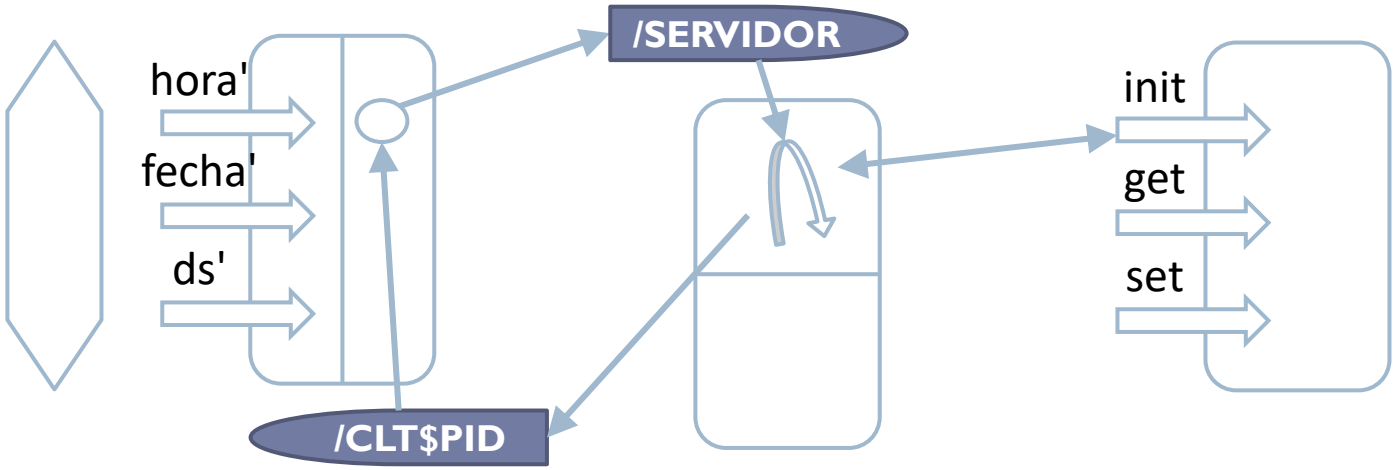
Añadir concurrencia con hilos POSIX



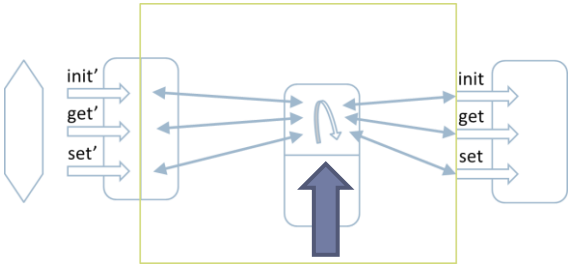
1. Sistema
2. Distribuido
3. Con colas de mensajes POSIX
4. **Concurrentes**

Diseño distribuido (v1.0)

Añadir concurrencia con hilos POSIX

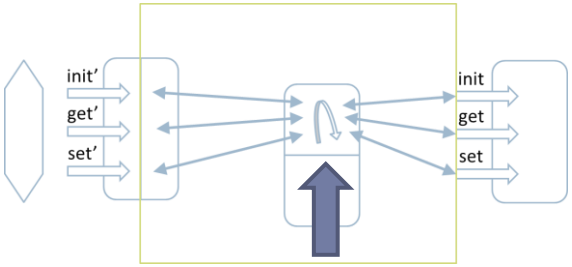


Diseño distribuido…



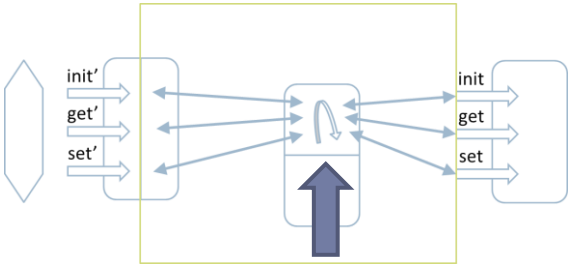

```
int main ( int argc, char *argv[] )
{
    struct petición p;
    int prio;  // y algunas variables más…

    pthread_attr_init(&attr) ;
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED) ;
    int qs = mq_open("/SERVIDOR", O_CREAT | O_RDONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    while (1) {
        mq_recv(qs, &p, sizeof(struct petición), &prio) ;
        pthread_create(&thid, &attr, tratar_petición, (void *)&p) ;
        <código de espera a que se haya creado el hilo y copiado &p>
    }
}
```

Diseño distribuido…



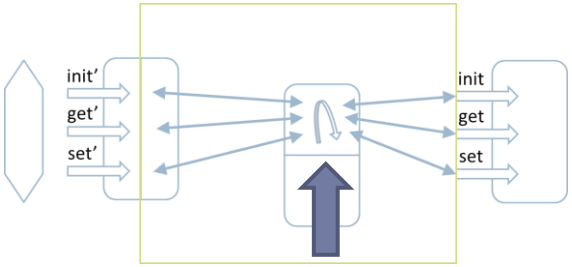

```
void tratar_petición ( struct petición * p )
{
     <código de sincronización para "p_local = *p" y señalizar que copiado>
    struct respuesta r;
    switch (p->op)
    {
        case 0:    r.status = real_init(p->name, p.value) ; // INIT
                   break ;
        case 2:    r.status = real_get(p->name, p->i, &(r.value)) ; // GET
                   break ;
        case 3:    r.status = real_set(p->name, p->i, p->value) ; // SET
                   break ;
    }

    int qs = mq_open(p->q_name, O_CREAT|O_WRONLY, 0700, NULL) ;
    mq_send(qr, &r, sizeof(struct respuesta), prio) ;
    mq_close(qr);
    pthread_exit(NULL) ;
}
```

Diseño distribuido…




```
int main ( int argc, char *argv[] )
{
    struct petición p;
    struct respuesta r;
    int prio;  // y algunas variables más…

    pthread_attr_init(&attr) ;
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED) ;
    int qs = mq_open("/SERVIDOR", O_CREAT | O_RDONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    while (1) {
        mq_recv(qs, &p, sizeof(struct petición), &prio) ;
        pthread_create(&thid, &attr, tratar_petición, (void *)&p) ;

        pthread_mutex_lock(&sync_mutex) ;
        while (sync_copied == FALSE) {
                pthread_cond_wait(&sync_cond, &sync_mutex) ;
        }
        sync_copied = FALSE ;
        pthread_mutex_unlock(&sync_mutex) ;
    }
}
```

Diseño distribuido…




```
void tratar_petición ( struct petición * p )
{
    struct petición p_local ;

    pthread_mutex_lock(&sync_mutex) ;
    p_local = *p ;
    sync_copied = TRUE ;
    pthread_cond_signal(&sync_cond) ;
    pthread_mutex_unlock(&sync_mutex) ;

    switch (p.op)
    {
      case 0:   r.status = real_init(p.nombre, p.ni) ;
                break ;
      case 2:   r.status = real_get(p.nombre, p.i, &(r.valor)) ;
                break ;
      case 3:   r.status = real_set(p.nombre, p.i, p.valor) ;
                break ;
    }

    int qs = mq_open(p->q_name, O_CREAT|O_WRONLY, 0700, NULL) ;
    mq_send(qr, &r, sizeof(struct respuesta), prio) ;
    mq_close(qr);
    pthread_exit(NULL) ;
}
```

Ejercicio 1

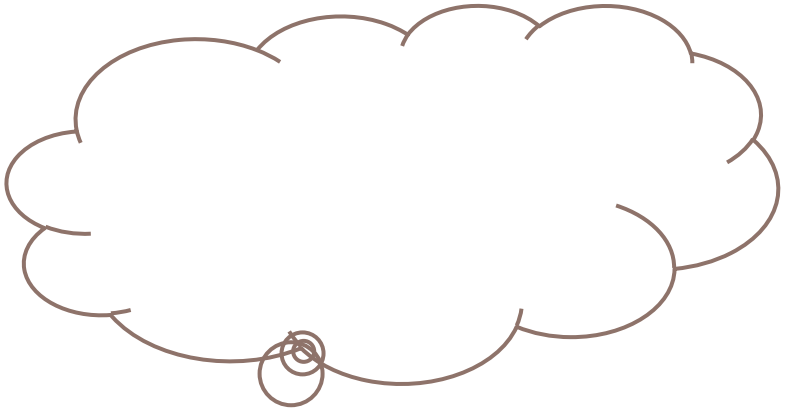
Desarrollar un servidor que permita obtener la hora, la fecha y el día de la semana en la que cae un día determinado.
Diseñar y desarrollar el cliente y el servidor en los dos siguientes supuestos:

a) Se dispone de un sistema con los siguientes servicios:
   - int **Connect**(int pid): este servicio estable una conexión con un proceso con identificador pid. Devuelve un identificador de conexión.
   - int **Accept**(): este servicio acepta una conexión de un proceso que ejecute el servicio Connect. Devuelve un identificador de conexión.
   - **Send** (int ic, char *mensaje, int long): este servicio envía un mensaje de una determinada longitud a través del identificador de conexión "ic".
   - **Receive** (int ic, char *mensaje, int long): este servicio recibe un mensaje de una determinada longitud de la conexión con identificador "ic."

   Asuma que los identificadores de los procesos son números enteros y que el servidor viene identificado por el número 1000

b) Considere un sistema que utiliza colas de mensajes POSIX.

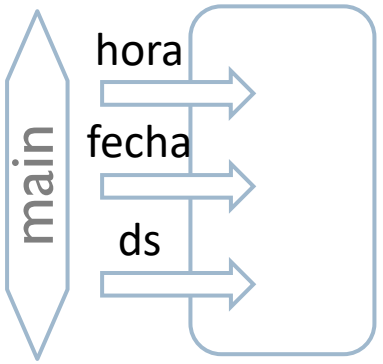
Diseño progresivo



1. Sistema
2. Distribuido
3. Con colas de mensajes POSIX

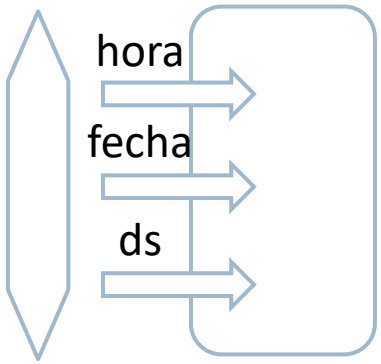
Diseño NO distribuido (v0.2)

Biblioteca usada desde programa



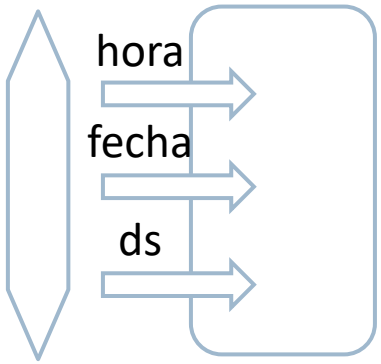
1. **Sistema**
2. Distribuido
3. Con colas de mensajes POSIX

Diseño no distribuido…




```
#include <time.h>

// Devuelve hora.
int hora ( char *hour )
{
    time_t now = time(NULL);
    struct tm *tm_struct = localtime(&now);
    sprint(hour, "%d", tm_struct->tm_hour);
    return 1;
}

// Devuelve la fecha.
int fecha ( char *fecha )
{
    time_t  clk = time(NULL);
    strcpy(fecha, ctime(clk)) ;
    return 1;
}
```

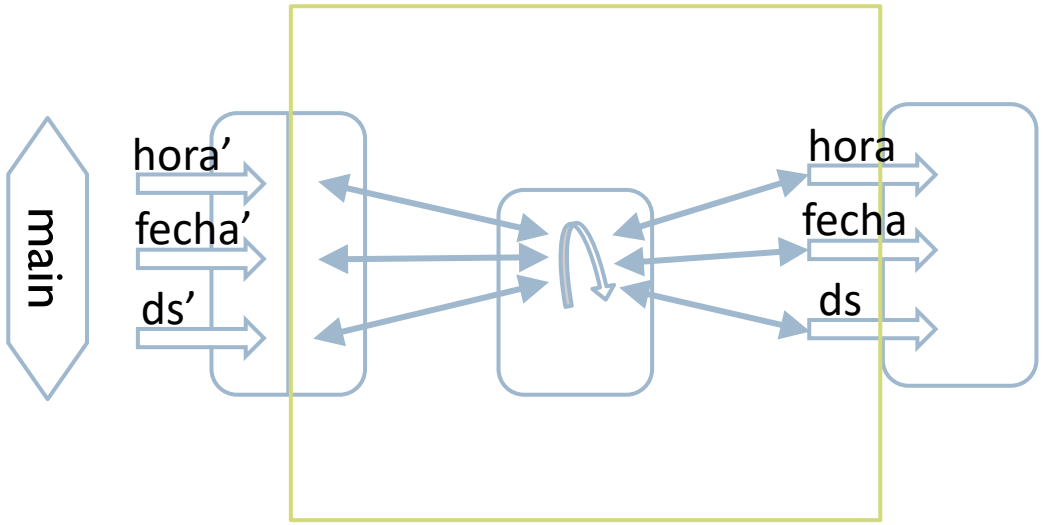
Diseño no distribuido…




```
#include <time.h>

// Devuelve hora.
int ds ( char *dia_semana )
{
    // https://stackoverflow.com/questions/6054016/c-program-to-find-day-of-week-given-date
    struct tm tm;
    memset((void *) &tm, 0, sizeof(tm));
    if (strptime(str, "%d-%m-%Y", &tm) != NULL) {
        time_t t = mktime(&tm);
        if (t >= 0) {
            int ds = localtime(&t)->tm_wday; // Sunday=0, Monday=1, etc.
            sprint(dia_semana, "%d", ds);
            return 1;
        }
    }

    return -1;
}
```

Diseño distribuido (v0.5)

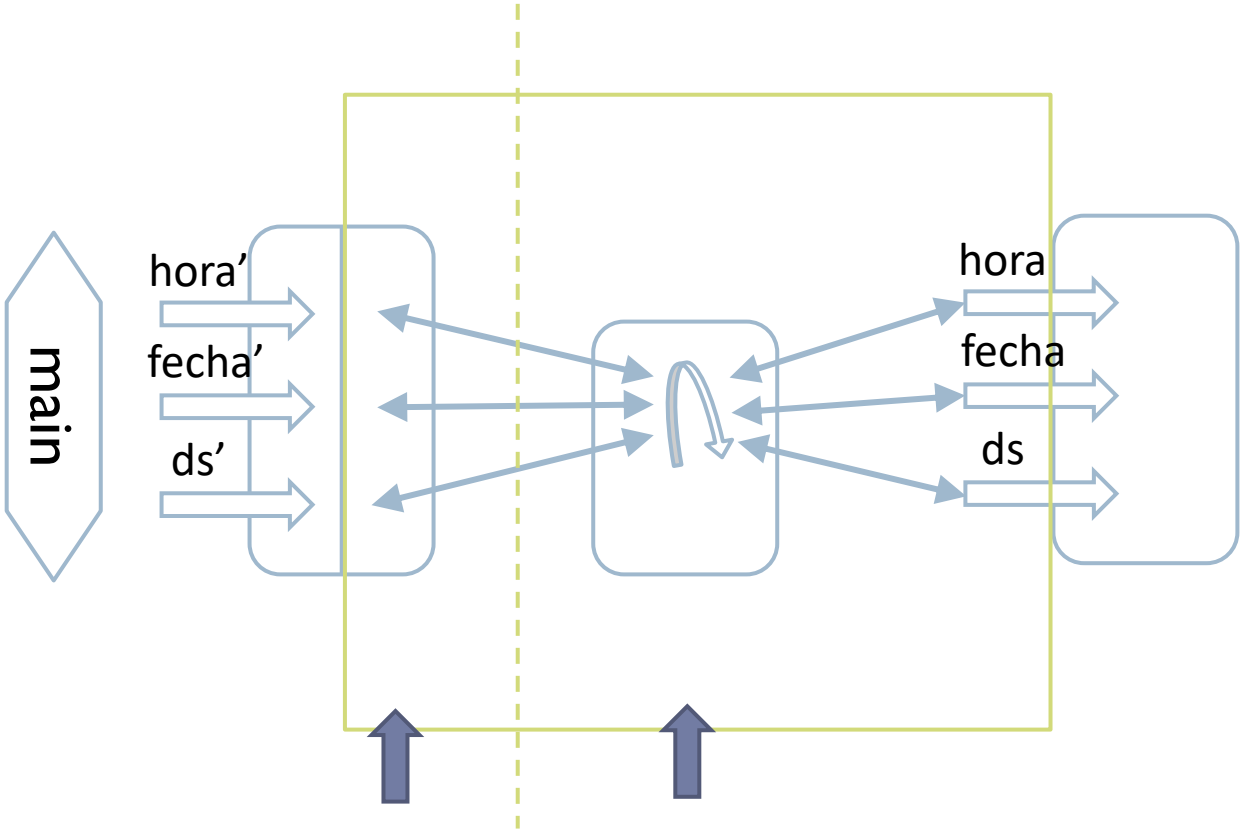
Usar un proxy para los servicios remotos



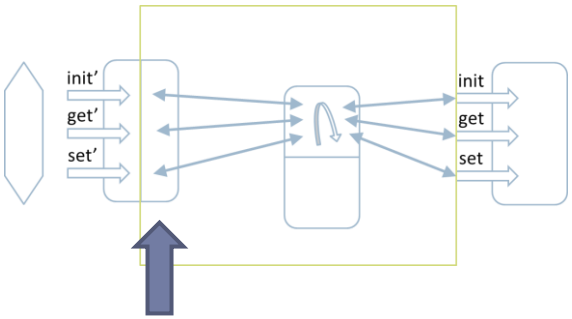
1. Sistema
2. **Distribuido**
3. Con colas de mensajes POSIX

Diseño distribuido (v0.5)

Usar un proxy para los servicios remotos



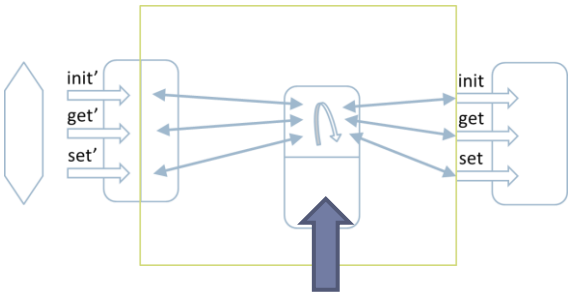
Diseño distribuido…




```
int send_recv ( char *op,
                char *value )
{
    char msg[1024]
    strcpy(msg, op)

    c1 = connect(1000)
    Send(c1, msg, 1024)
    Receive(c1, msg, 1024)

    strcpy(value, msg+1)
    return (int)msg[0]
}
```

```
int hora ( char *hour )
{
    return send_recv("hora", hour)
}

int fecha ( char *fecha )
{
    return send_recv( "fecha", fecha)
}

int ds ( char *dia_semana )
{
    return send_recv("ds", día_semana)
}
```

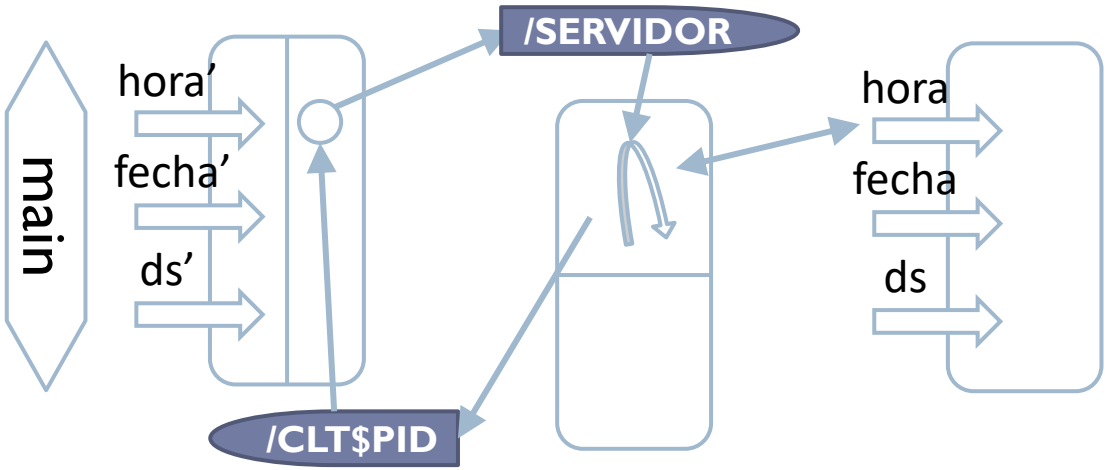
Diseño distribuido…




```
int main ( int argc, char *argv )
{
    while (TRUE)
    {
        c1 = Accept()
        Receive(c1, msg, 1024)
        switch(msg)
        {
            case "hora":  msg[0] = hora(msg+1) ;
                          break;
            case "fecha": msg[0] = fecha(msg+1) ;
                          break;
            case "ds":    msg[0] = ds(msg+1) ;
                          break;
        }
        Send(c1, msg, 1024) ;
    }
}
```

Diseño distribuido (v0.8)

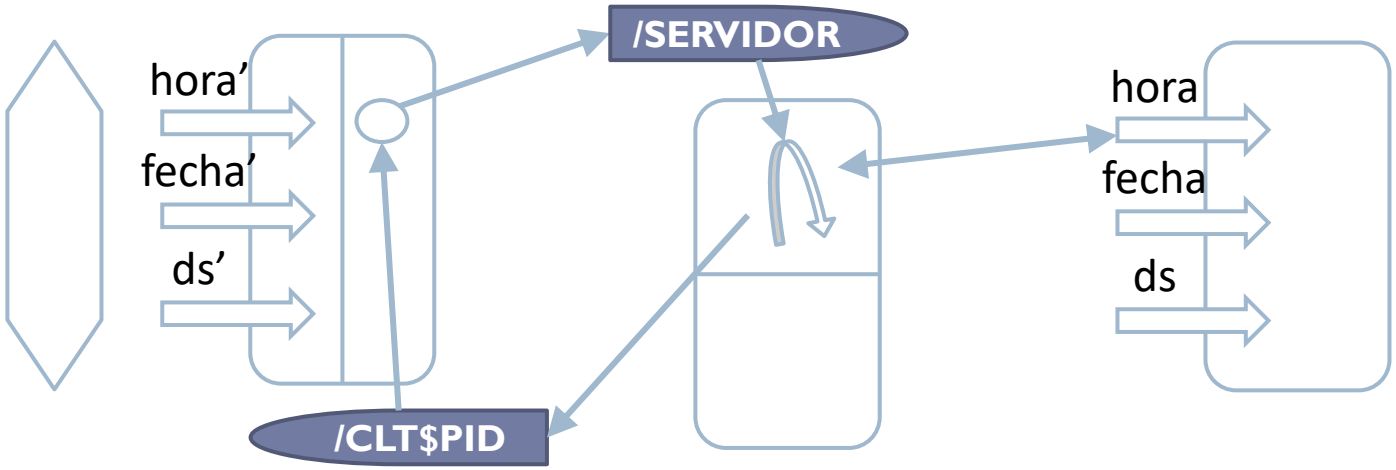
Adaptar proxy a colas de mensajes POSIX



1. Sistema
2. Distribuido
3. **Con colas de mensajes POSIX**

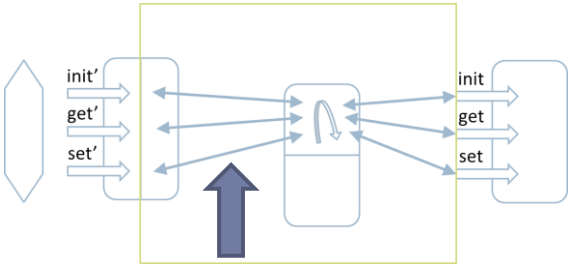
Diseño distribuido (v0.8)

Adaptar proxy a colas de mensajes POSIX



- El mensaje de petición ha de valer para todos los servicios
  - Establecer un identificador numérico para cada servicio
  - Establecer los parámetros para cada servicio y generar una petición la fusión de todos + identificador de servicio.
  - Establecer las respuestas para cada servicio y generar una respuesta fusión.

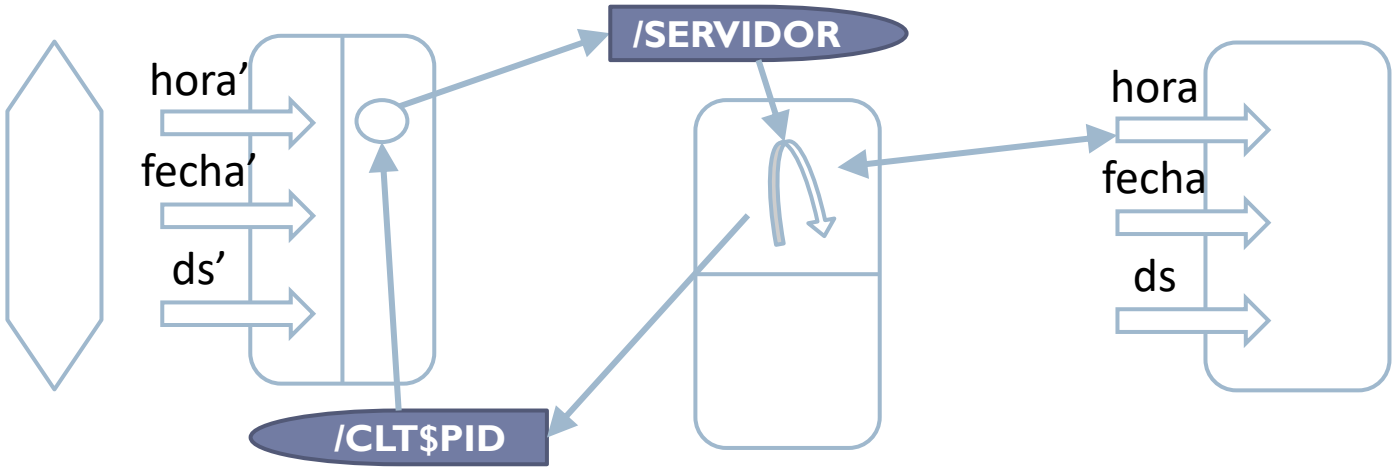
Diseño distribuido…




```
//  petición = op + q_name
struct peticion
{
      int     op;
      char  q_name[MAX];
} ;


// respuesta = (value, status)
struct respuesta
{
      char  value[32];
      char  status;
} ;
```

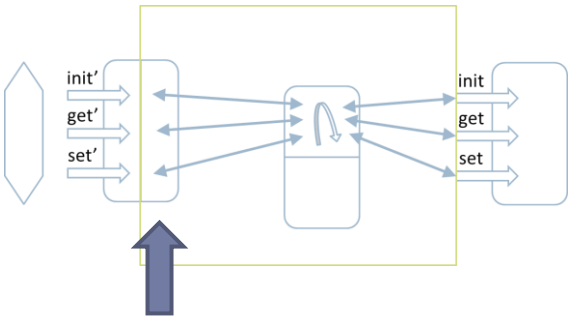
Diseño distribuido (v0.8)

Adaptar proxy a colas de mensajes POSIX



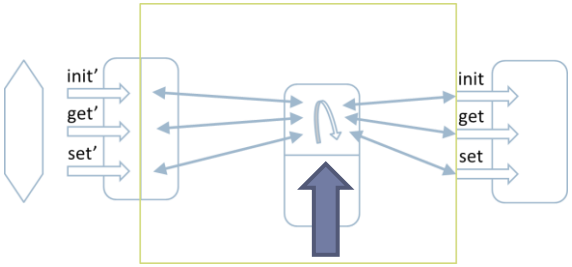
- Las colas POSIX son unidireccionales
  - Una cola general de peticiones creada por el servidor
  - Por cada cliente activo una cola (efímera) para recibir la respuesta.
    - La cola es privada para cada el cliente con nombre único (usar getpid())

Diseño distribuido…



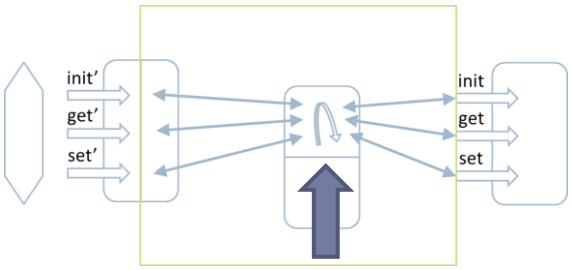

```
int hora  ( char *hour )
{
    struct petición p;
    struct respuesta r;
    char qr_name[1024];  int prio;

    sprintf(qr_name, "%s%d", "/CLIENTE_", getpid()) ;
    int qs = mq_open("/SERVIDOR", O_CREAT|O_WRONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    int qr = mq_open(qr_name, O_CREAT|O_RDONLY) ;
    if (qr == -1) { mq_close(qs) ; return -1; }
    p.op = 1;
    strcpy(p.q_name, qr_name);
    mq_send(qs,  (char *)&p,  sizeof(struct petición),    0) ;
    mq_recv(qr,   (char *)&r,  sizeof(struct respuesta),  &prio) ;
    mq_close(qs);
    mq_close(qr);
    mq_unlink(qr_name);
    strcpy(hour, r.value) ;
    return r.status ;
}
```

Diseño distribuido…




```
int main ( int argc, char *argv[] )
{
    struct petición p;
    int prio;

    int qs = mq_open("/SERVIDOR", O_CREAT | O_RDONLY, 0700, NULL) ;
    if (qs == -1) { return -1 ; }
    while (1)
    {
        mq_recv(qs, &p, sizeof(p), &prio) ;
        tratar_petición(&p) ;
    }
}
```

Diseño distribuido…




```
void tratar_petición ( struct petición * p )
{
    struct respuesta r ;
    switch (p->op)
    {
        case 1: // HORA
                    r.status = real_hora(p->value) ;
                    break ;
        case 2: // FECHA
                    r.status = real_fecha(p.value) ;
                    break ;
        case 3: // DIA_SEMANA
                    r.status = real_ds(p->value) ;
                    break ;
    }

    int qs = mq_open(p->q_name, O_CREAT|O_WRONLY, 0700, NULL) ;
    mq_send(qr, &r, sizeof(struct respuesta), prio) ;
    mq_close(qr);
}
```

Grupo ARCOS
Universidad Carlos III de Madrid

Lección 3
Ejercicios de paso de mensajes

Sistemas Distribuidos
Grado en Ingeniería Informática

CC BY NC SA