Programación Concurrente y Tiempo Real

Laboratorio - Práctica 3
Paso de Mensajes

D. Vallejo, M.A. Redondo, J.A. Albusac,
C. González, J. Ruiz

Escuela Superior de Informática

Universidad de Castilla-La Mancha

Práctica 3. Paso de Mensajes

1. Introducción
2. Primitivas POSIX
3. Ejemplo de Envío y Recepción
4. Problema

Introducción

- Almacenamiento de mensajes en buffer intermedio “*Cola*” o “*Buzón*”.
- Manejo implícito de mecanismo para intercambio de información.
- Dos primitivas básicas:
  - **Envío** (*send*). Habitualmente se utiliza la versión no bloqueante.
  - **Recepción** (*receive*). Habitualmente se
  - utiliza la versión bloqueante.

Primitivas

Apertura o Creación de una Cola de Mensajes

```
#include <fcntl.h>
#include <sys/stat.h>
#include <mqueue.h>

/* Devuelve descriptor de la cola o -1 si error */
mqd_t mq_open (
  const char *name,       /* Nombre de la cola */
  int oflag               /* Flags (menos O_CREAT) */
);

mqd_t mq_open (
  const char *name,       /* Nombre de la cola */
  int oflag               /* Flags */
  mode_t perms,           /* Permisos */
  struct mq_attr *attr    /* Atributos */
);
```

Ejemplo de Uso

Ejemplo de Creación

```
mqd_t qHandler;              /* Descriptor de la cola */
struct mq_attr mqAttr;       /* Estructura de atributos */

mqAttr.mq_maxmsg = 10;       /* Máximo n° de mensajes */
mqAttr.mq_msgsize = 1024;    /* Tamaño máximo de mensaje */

qHandler = mq_open("/prueba", O_RDWR | O_CREAT,
                   S_IWUSR | S_IRUSR, &mqAttr);

if (qHandler == -1) {        /* Gestión de errores */
  fprintf(stderr, "%s\n", strerror(errno));
  exit(EXIT_FAILURE);
}
```

Primitivas

Cierre de la cola de mensajes

```
#include <mqueue.h>
int mq_close (
  mqd_t mqdes           /* Descriptor de la cola */
);
```

Eliminación de la cola de mensajes

```
#include <mqueue.h>
int mq_unlink (
  const char *name      /* Nombre de la cola */
);
```

Primitivas

Envío y Recepción de Mensajes

```
#include <mqueue.h>

int mq_send (
  mqd_t mqdes,              /* Descriptor de la cola */
  const char *msg_ptr,      /* Mensaje (contenido) */
  size_t msg_len,           /* Tamaño del mensaje */
  unsigned msg_prio         /* Prioridad */
);

ssize_t mq_receive (
  mqd_t mqdes,              /* Descriptor de la cola */
  char *msg_ptr,            /* Buffer para mensaje */
  size_t msg_len,           /* Tamaño del buffer */
  unsigned *msg_prio        /* Prioridad (o NULL) */
);
```

Ejemplo de Uso

Ejemplo de Envío de Mensajes **No** Bloqueante

```
mqd_t qHandler;
char buffer[512];

sprintf (buffer, "[ Saludos de %d ]", getpid());
mq_send (qHandler, buffer, sizeof(buffer), 1);
```

¡El buffer de envío debe ser **menor o igual** que *mq_msgsize*!

Ejemplo de Uso

Ejemplo de Recepción de Mensajes Bloqueante

```
mqd_t qHandler;
unsigned int prioridad;
int rc;
char buffer[1024];

rc = mq_receive (qHandler, buffer, sizeof(buffer),
                 &prioridad);
if (rc == -1) fprintf(stderr, "%s\n", strerror(errno));
else printf ("Recibiendo mensaje: %s\n", buffer);
```

¡El buffer de recepción debe ser **mayor o igual** que *mq_msgsize*!

Custom messages

- Muchos APIs utilizan `char*` como buffer genérico...

¡Considera un puntero a un buffer
junto con el tamaño del buffer
para usar tus TADs!

Custom messages

Ejemplo de uso de *custom data*

```
typedef struct {
  int valor;
} TData;

TData buffer;

/* Atributos del buzón */
mqAttr.mq_maxmsg = 10;
mqAttr.mq_msgsize = sizeof(TData);

/* Recepción */
mq_receive(qHandler, (char *)&buffer,
           sizeof(buffer), &prioridad);
/* Envío */
mq_send(qHandler, (const char *)&buffer,
        sizeof(buffer), 1);
```

Análisis del problema

Entero	**1**	**2**	**...**	**25**	**26**	**27**	**28**	**...**	**51**	**52**	**>52**
Traducción	**a**	**b**	**...**	**y**	**z**	**A**	**B**	**...**	**Y**	**Z**	**(Espacio Blanco)**
Código ASCII	*97*	*98*	*...*	*121*	*122*	*65*	*66*	*...*	*89*	*90*	*32*

Ejemplo (clave = 4):

"35.17.21.54.24.5.1.10" ➔ 39 21 25 59 28 9 5 14

39 21 25 59 28 9 5 14 ➔

'77' '117' '121' '32' '64' '105' '101' '110'

"**Muy Bien**"

Análisis del problema

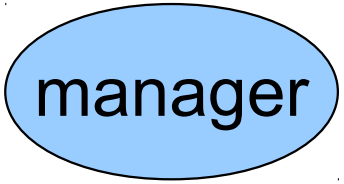
./exec/manager 35.17.21.54.24.5.1.10 4 2 4

Cadena a traducir

Clave

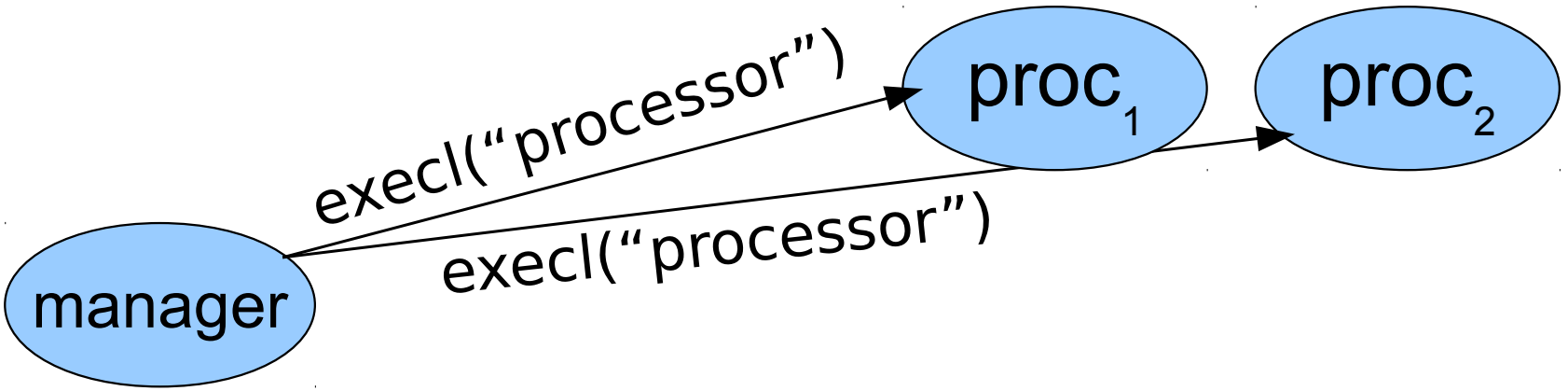
Nº Procesadores

Nº Subvectores



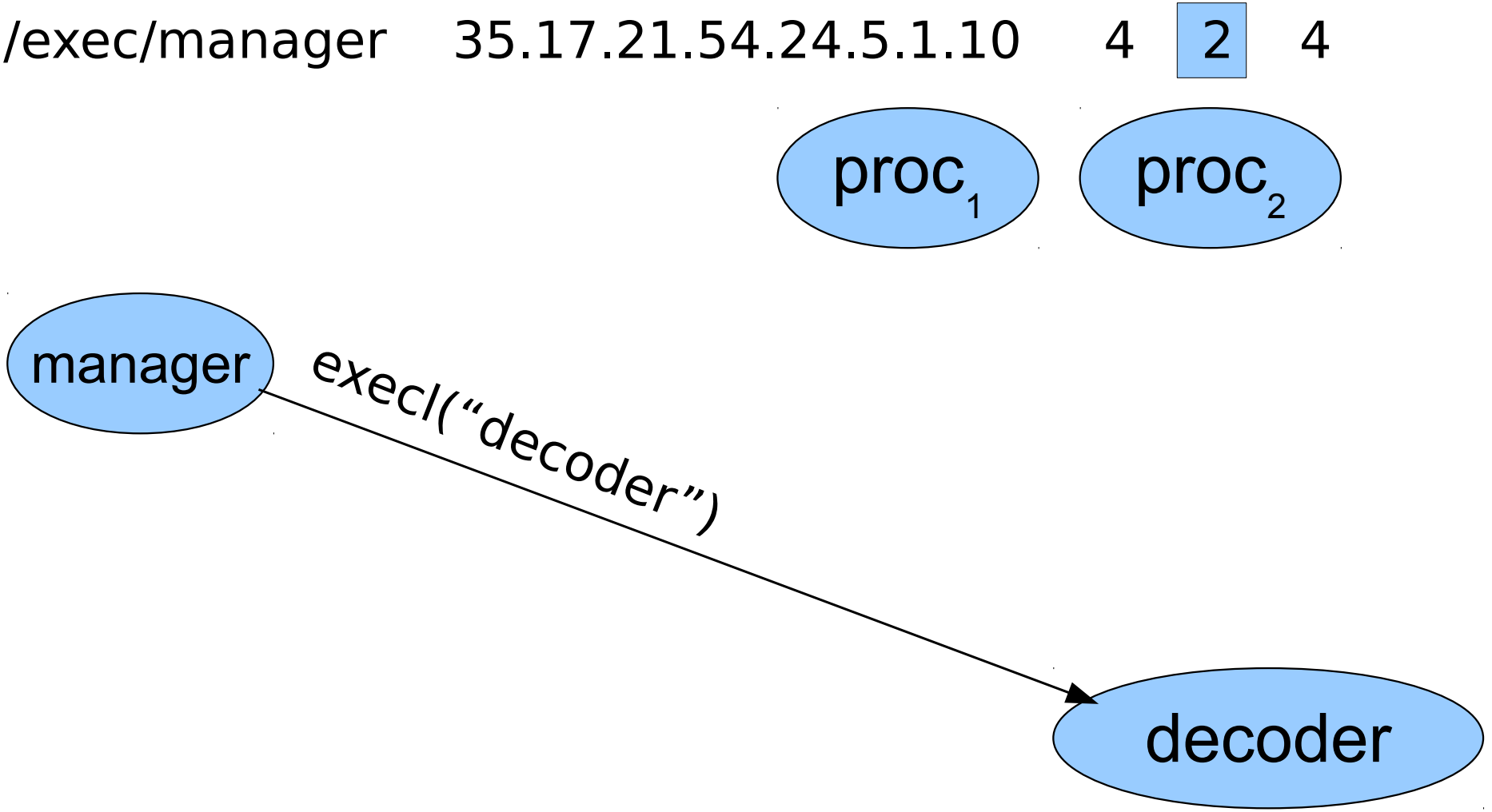
Análisis del problema

./exec/manager 35.17.21.54.24.5.1.10 4 2 4



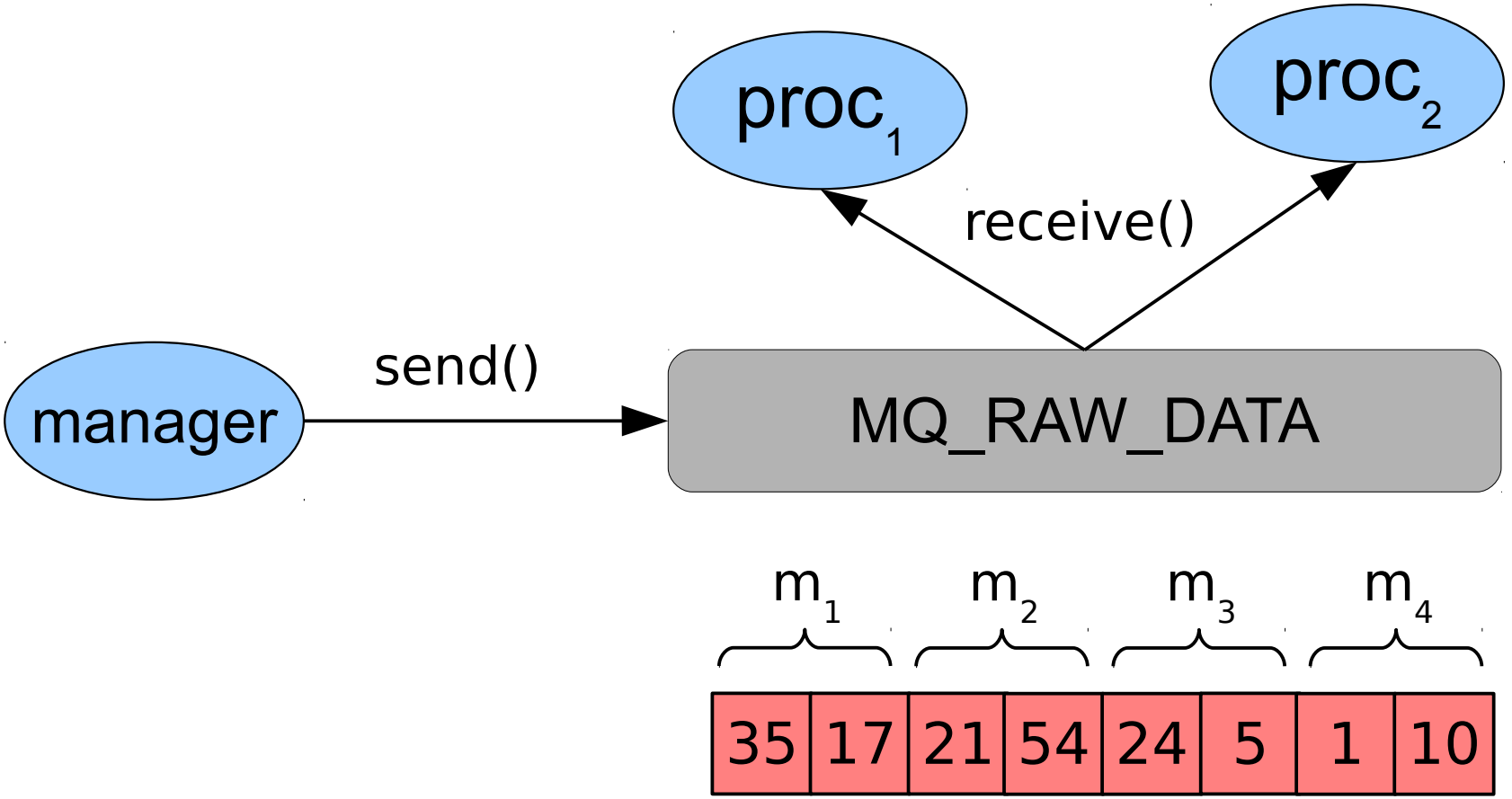
Análisis del problema

./exec/manager 35.17.21.54.24.5.1.10 4 2 4



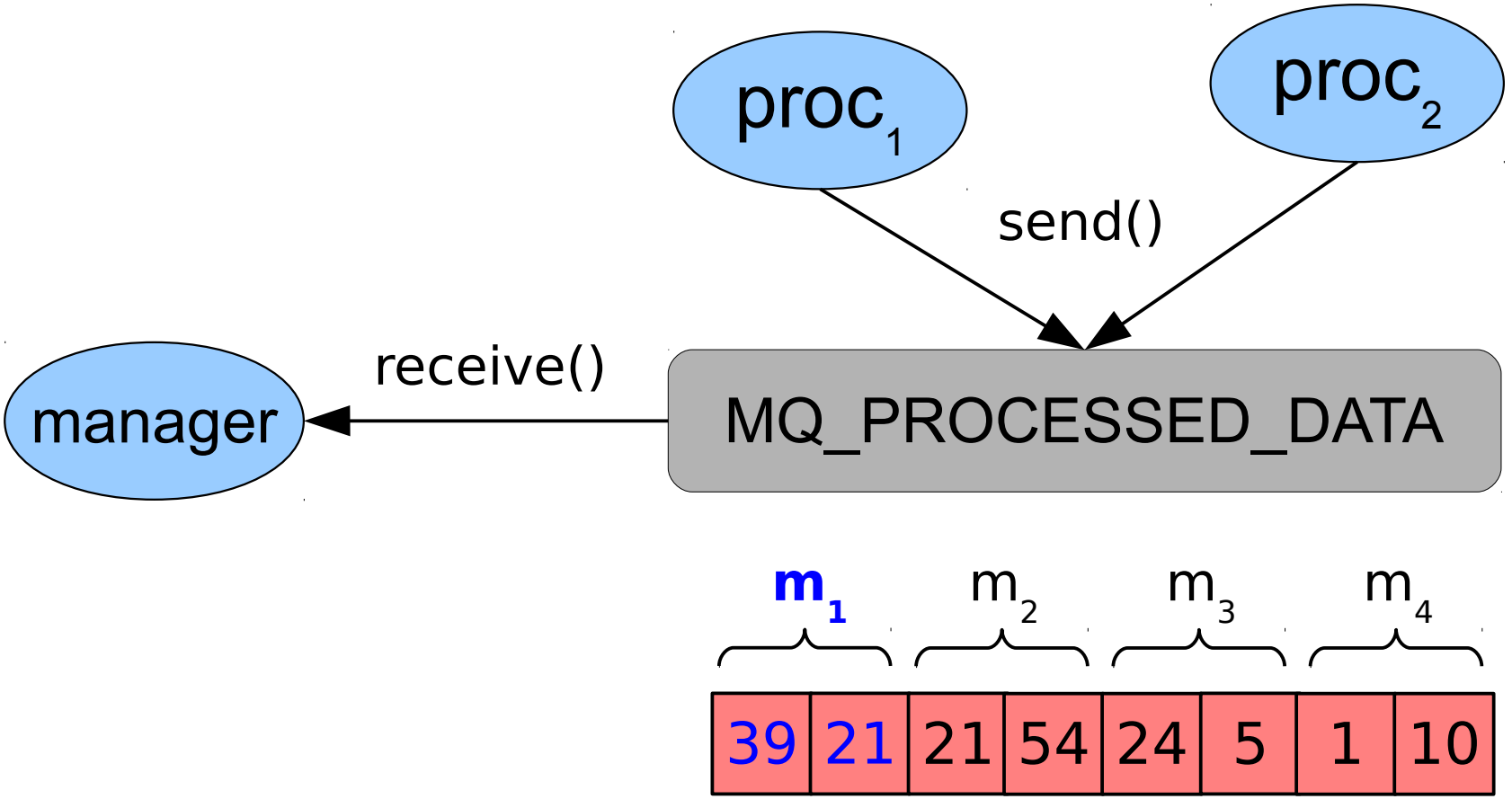
Análisis del problema

./exec/manager 35.17.21.54.24.5.1.10 4 2 4



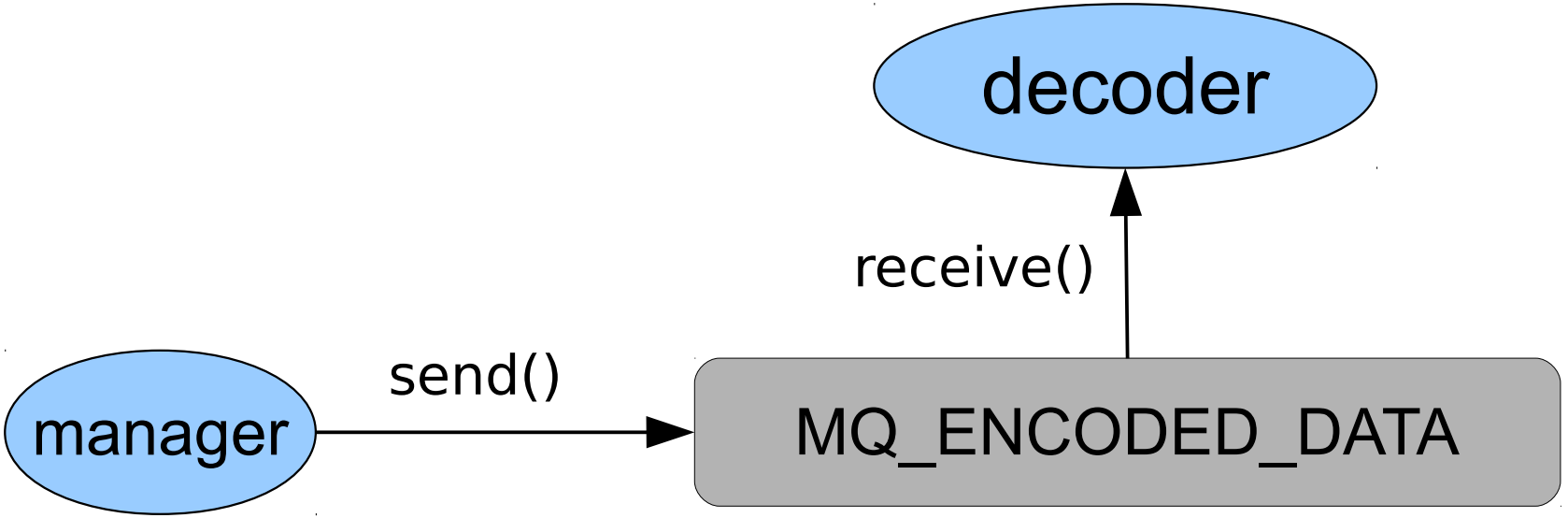
Análisis del problema

./exec/manager 35.17.21.54.24.5.1.10 4 2 4



Análisis del problema

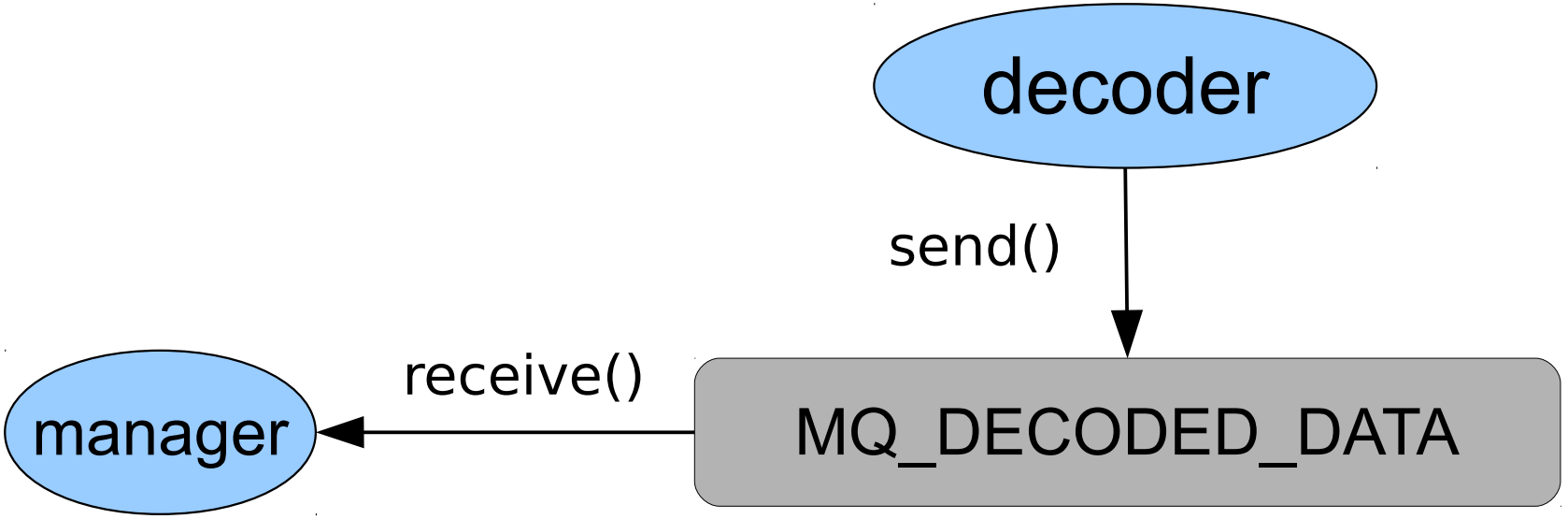
./exec/manager 35.17.21.54.24.5.1.10 4 2 4



39	21	25	59	29	9	5	14

Análisis del problema

./exec/manager 35.17.21.54.24.5.1.10 4 2 4



“**Muy Bien**”

77	117	121	32	64	105	101	110

Estructuras de datos

MQ_RAW_DATA y MQ_PROCESSED_DATA

```
struct MsgProcessor_t {
  /* Data of the subvector to be processed */
  char data[MAX_ARRAY_SIZE];
  /* Start subvector index */
  int  index_start;
  /* Number of elements in the subvector */
  int  n_elements;
  /* Key to carry out the 'processing' */
  int  key;
};
```

Estructuras de datos

MQ_ENCODED_DATA y MQ_DECODED_DATA

```
struct MsgDecoder_t {
  /* Full vector to be decoded */
  char data[MAX_ARRAY_SIZE];
  /* Number of elements to be decoded */
  int  n_elements;
};
```

Envío/recepción de tareas

manager.c

```
/* Envío de tareas a los processors */
for (i = 0; i < n_subvectors; i++) {
  crear_tarea(&msg_task);
  send(MQ_RAW_DATA, msg_task);
}
```

processor.c

```
receive(MQ_RAW_DATA, &msg_task);
process_task(&msg_task);
send(MQ_PROCESSED_DATA, msg_task);
```

Recepción de resultados

manager.c

```
 /* Recepción de resultados procesados
    (sin decodificar aún) */
for (i = 0;
     i < n_subvectors;
     i++) {
  receive(MQ_PROCESSED_DATA,
          &msg_task);
}
```

Sincronización con el traductor

manager.c

```
send(MQ_ENCODED_DATA, msg_task);
receive(MQ_DECODED_DATA, &msg_task);
```

decoder.c

```
receive(MQ_ENCODED_DATA, &msg_task);
decode(&msg_task);
send(MQ_DECODED_DATA, msg_task);
```

Patrón **Rendezvous**