Esercitazione: Primitive UNIX per lo scambio dei messaggi

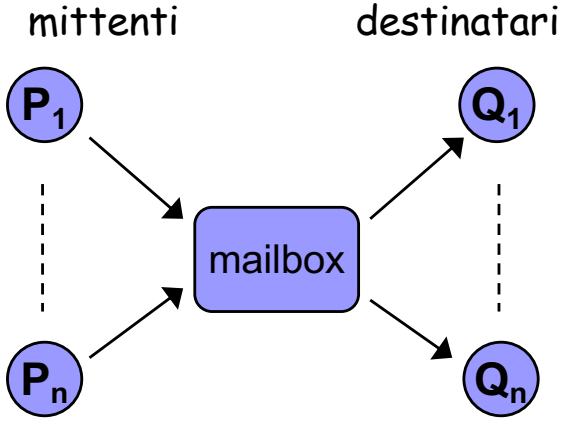
Corso di Laurea in Ingegneria Informatica

Università degli Studi di Napoli Federico II

Anno Accademico 2024/2025, Canale San Giovanni

Scambio messaggi

- UNIX supporta la comunicazione indiretta fra processi tramite mailbox (dette "code di messaggi")
- Primitive del SO:
  - Send asincrona
  - Receive (bloccante e non bloccante)
- In ingresso/uscita, un ***messaggio (vettore di byte)***



msgget

- Creazione di una coda di messaggi

```
int msgget (chiave, flag)
```

  - chiave e flag hanno lo stesso ruolo di semget() e shmget()

  - Nota: non occorre indicare la dimensione della coda e il numero massimo di messaggi, sono decisi autonomamente dal SO

Esempio

```
#include <sys/ipc.h>
#include <sys/msg.h>

int main() {
  key_t chiave = ... // IPC_PRIVATE oppure ftok()
  int qid;
  qid = msgget( chiave, IPC_CREAT | 0664 );

  if(qid < 0) {
    perror("errore msgget");
    exit(1);
  }

  ...
}
```

msgctl

- msgctl(): primitiva per gestione della mailbox
- Per deallocare una mailbox:

```
msgctl (msqid, IPC_RMID, 0)
```

- Nota: la mailbox è deallocata immediatamente (senza attendere l'uscita di tutti i processi)

msgsnd

- Invio (con accodamento) di un messaggio alla mailbox

```
int msgsnd(int msqid,
           void * msgp,
           size_t msgsz,
           int msgflg)
```

Puntatore a una struttura-messaggio

Dimensione struttura-messaggio

Il sistema fornisce una struttura-messaggio (struct msgbuf), definita in msg.h come:

```
struct msgbuf {
    long message_type;
    char message_text [MAX_SIZE];
};
```

msgsnd

- **Non è obbligatorio** inviare un vettore di caratteri
- Il programmatore può definire una propria struttura, con qualsiasi altro tipo di dato (intero, double, ...)

```
struct mymessage {
    long message_type;
    int x;
    float y;
    char sometext[10];
    ....
};
```

È obbligatorio che vi sia un primo campo di tipo long ("tipo" di messaggio)

Esempio msgsnd

```
typedef struct {
    long type;
    int val;
    char text[25];
} mymessage;

mymessage msg;
msg.type = 1;
msg.value = 5;
strcpy(msg.text, "Il mio primo messaggio.");

// send asincrona (non si blocca, eccetto se la mailbox è piena)
result = msgsnd(msqid, &msg, sizeof(mymessage) - sizeof(long), 0);

if(result<0) {
    perror("Errore msgsnd");
}
```

Esempio msgsnd

```
typedef struct {
    long type;
    int val;
    char text[25];
} mymessage;

mymessage msg;
msg.type = 1;
msg.value = 5;
strcpy(msg.text, "Il mio primo messaggio.");

// send asincrona (non si blocca, eccetto se la mailbox è piena)
result = msgsnd(msqid, &msg, sizeof(mymessage) - sizeof(long), 0);

if(result<0) {
    perror("Errore msgsnd");
}
```

Il campo tipo deve essere sempre strettamente positivo (type > 0).

I valori 0 e negativi sono riservati per altri usi.

Esempio msgsnd

```
typedef struct {
    long type;
    int val;
    char text[25];
} mymessage;

mymessage msg;
msg.type = 1;
msg.value = 5;
strcpy(msg.text, "Il s

// send asincrona (non si blocca, eccetto se la mailbox è piena)
result = msgsnd(msqid, &msg, sizeof(mymessage) - sizeof(long), 0);

if(result<0) {
    perror("Errore msgsnd");
}
```

In msgsnd(), è consentito passare un puntatore "struct mymessage" invece che di tipo "msgbuf" (casting tra puntatori)

Esempio msgsnd

```
typedef struct {
    long type;
    int val;
    char text[25];
} mymessage;

mymessage msg;
msg.type = 1;
msg.value = 5;
strcpy(msg.text, "Il mio primo mess          );

// send asincrona (non si blocca, eccetto se la mailbox è piena)
result = msgsnd(msqid, &msg, sizeof(mymessage) - sizeof(long), 0);

if(result<0) {
    perror("Errore msgsnd");
}
```

Dalla dimensione del messaggio, bisogna escludere la dimensione del campo "tipo" (long)

msgsnd

- Alcune convenzioni da rispettare:
  - Il primo campo della struct ("type") deve sempre essere di tipo long
  - Il valore del long deve essere strettamente positivo (>= 1)
  - Il terzo parametro di msgsnd (size_t) deve indicare la dimensione in byte della struct, sottraendo la dimensione del long

msgrcv

- Ricezione di un messaggio da una mailbox

```
int msgrcv(int msqid,
           void * msgp,
           size_t msgsz,
           long msgtyp,
           int msgflg)
```

Puntatore a una struttura-messaggio (parametro di **uscita**)

Dimensione struttura-messaggio

Tipo di messaggio da ricevere (ricezione selettiva)

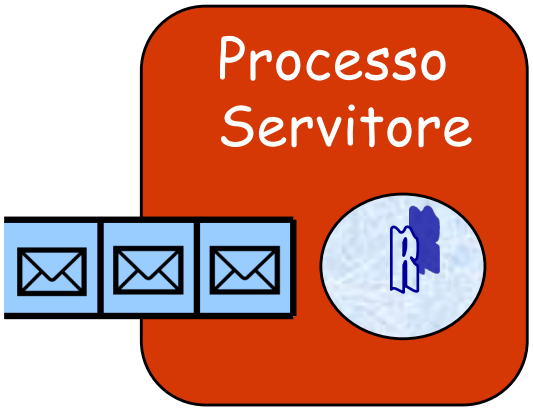
Esempio msgrcv

```
typedef struct {
    long type;
    int val;
    char text[25];
} mymessage;

mymessage msg;     // non è necessario inizializzarlo,
                   // è un parametro di sola uscita

// receive bloccante (default)
result = msgrcv(msqid, &msg, sizeof(mymessage)-sizeof(long), 0, 0);

if(result<0) {
    perror("Errore msgrcv");
}

printf("Messaggio: tipo=%d, val=%d, text=%s\n",
                        msg.type, msg.val, msg.text);
```

Tipo di messaggi

```
struct mymessage {
    long message_type;   // campo "long" obbligatorio
    ....
};
```

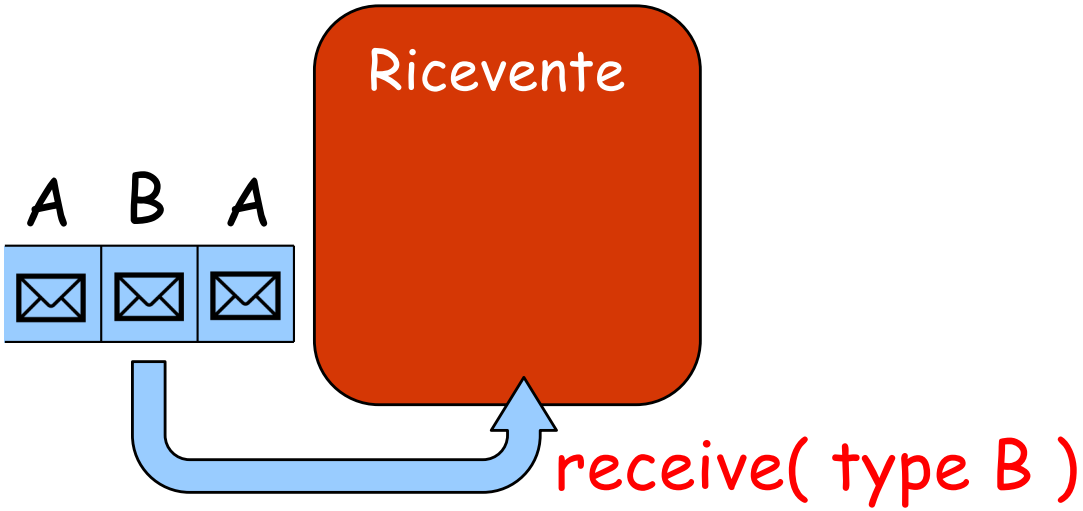
Il campo **message_type** (obbligatorio) permette al programmatore di distinguere tra più tipi di messaggi




```
if(msg.type == ACCENDI) {
  R.stato = ACCESO
}
else if(msg.type == SPEGNI) {
  R.stato == SPENTO
}
else if(msg.type == LEGGI) {
  risposta = R.temperatura
  send(risposta)
}
```

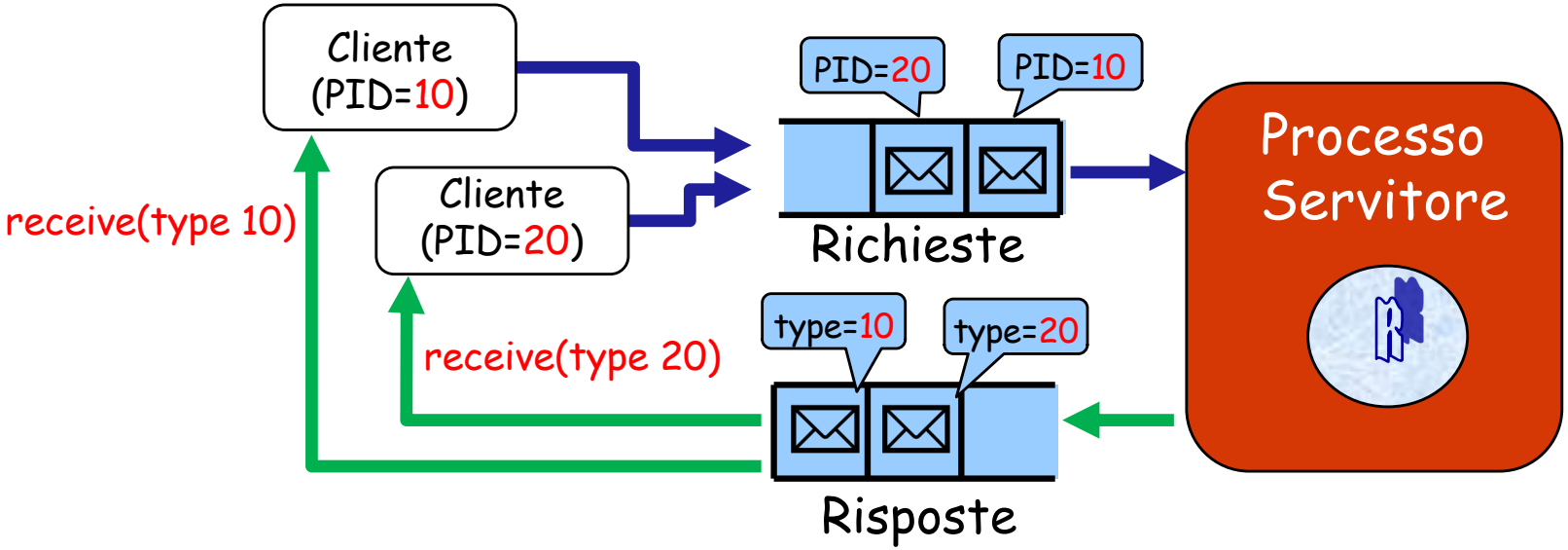
Tipo di messaggi

- Il campo type è utile per la **ricezione selettiva**
- Il ricevente preleva il primo messaggio contenente uno specifico valore nel campo tipo (**non** in ordine **FIFO**)



Tipo di messaggi

- In alcuni nostri esercizi, utilizzeremo il campo tipo per la **comunicazione indiretta simmetrica**



- Due code condivise fra server e client (richieste, risposte)
- Ogni client invia una richiesta includendo il suo **PID**
- Il server invia le risposte usando il PID nel **campo tipo**
- Ogni client **riceve selettivamente** la risposta con il suo PID

Tipo di messaggi

- Il terzo parametro in msgrcv() (***msgtyp***, intero) in indica il tipo di messaggi ricevere

```
int msgrcv(int msqid,
           void * msgp,
           size_t msgsz,
           long msgtyp,
           int msgflg)
```

Tipo di messaggi

- Il terzo parametro in msgrcv() (***msgtyp***, intero) in indica il tipo di messaggi ricevere

```
#define RICHIESTA_LETTURA 1
#define RICHIESTA_SCRITTURA 2
...

long type = RICHIESTA_LETTURA;

// receive bloccante, e selettiva (riceve solo richieste di "lettura")
msgrcv(msqid, &msg, sizeof(mymessage)-sizeof(long), type, 0);
```

- *msgtyp* = 0: preleva in ordine **FIFO**
- *msgtyp* > 0: preleva il primo messaggio tale che ***tipo == msgtyp***
- *msgtyp* < 0: preleva il primo messaggio tale che ***tipo >= |msgtyp|***

Flags (msgrcv)

- Il parametro `msgflg` determina se la msgrcv() effettua una ricezione bloccante

```
int msgrcv(int msqid,
           void * msgp,
           size_t msgsz,
           long msgtyp,
           int msgflg)
```

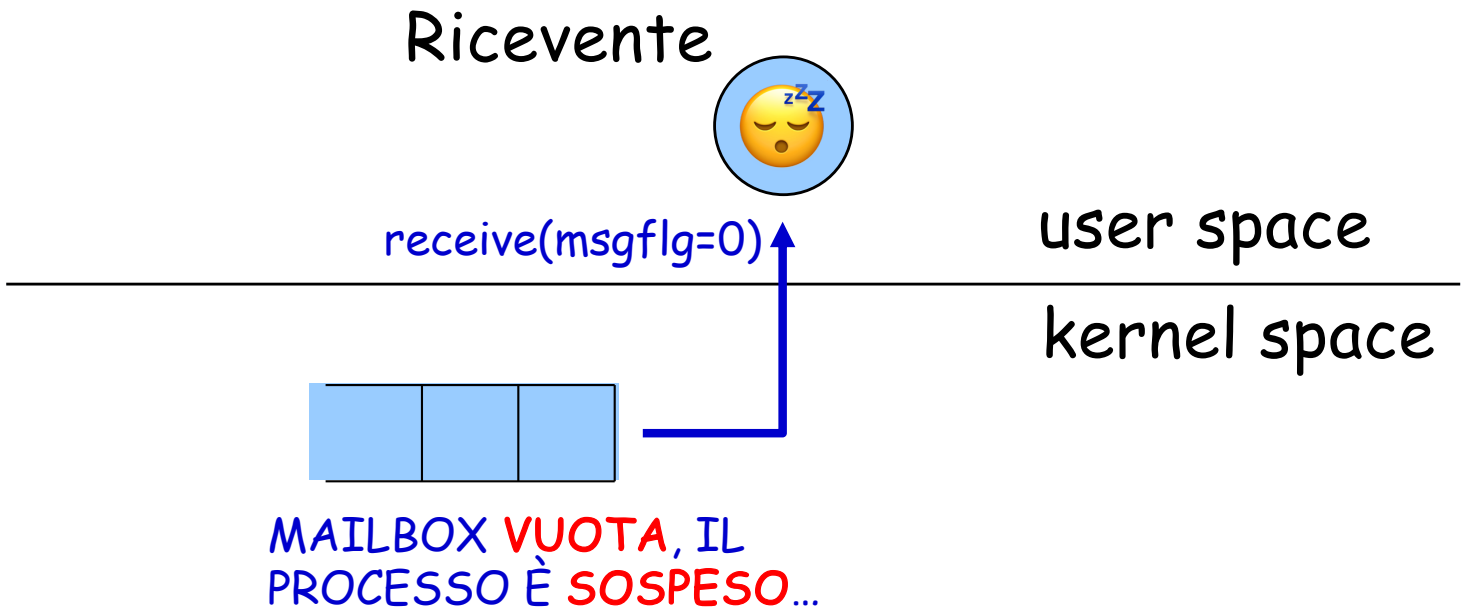
Flags (msgrcv)

- Se `msgflg = 0`, msgrcv() è una ricezione bloccante
- Il processo si sospende sulla *msgrcv()* fino al giungere del messaggio

```
mymessage msg;

// receive bloccante
result = msgrcv(msqid, &msg, sizeof(mymessage)-sizeof(long), 0, 0);
```

Flags (msgrcv)

- Se `msgflg = 0`, msgrcv() è una ricezione bloccante

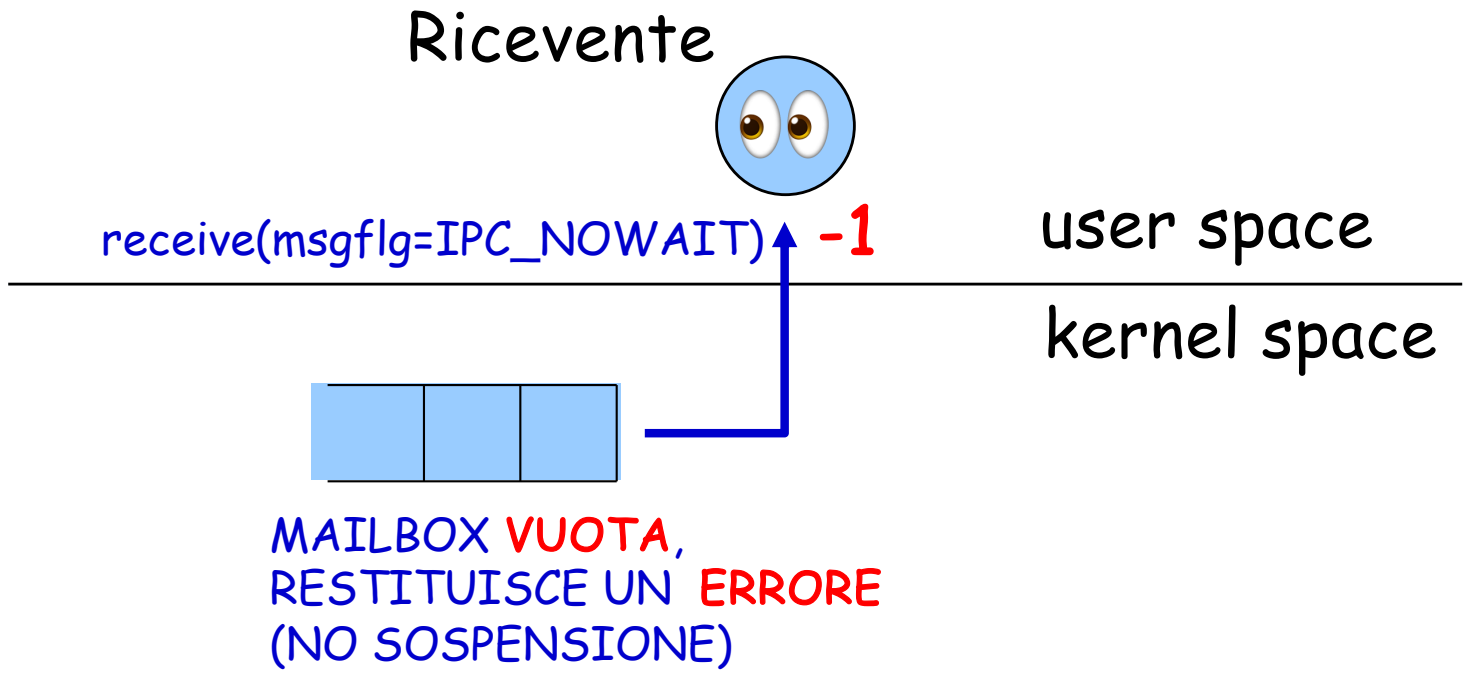


Flags (msgrcv)

- Se `msgflg = IPC_NOWAIT`, la msgrcv() effettua una ricezione non-bloccante
- Il processo non si sospende se non ci sono messaggi da ricevere
- Restituisce immediatamente un valore di ritorno:
  - positivo, se è disponibile un messaggio
  - negativo, se non ci sono messaggi

Flags (msgrcv)

- Se `msgflg = IPC_NOWAIT`, la msgrcv() effettua una ricezione non-bloccante



Flags (msgrcv)

```
mymessage msg;

// receive non-bloccante
result = msgrcv(msqid, &msg, sizeof(mymessage)-sizeof(long), 0,
                                                    IPC_NOWAIT);
if(result >= 0) {

    printf("Messaggio disponibile: tipo=%d, val=%d, text=%s\n",
                                        msg.type, msg.val, msg.text);
}
else if(result < 0 && errno == ENOMSG) {

    printf("Nessun messaggio disponibile\n");
    // ... effettua altro lavoro ...
}
else {
    perror("Errore msgrcv");
}
```

Flags (msgrcv)

```
mymessage msg;

// receive non-bloccante
result = msgrcv(msqid, &

if(result >= 0) {

    printf("Messaggio di

}
else if(result < 0 && errno == ENOMSG) {

    printf("Nessun messaggio disponibile\n");
    // ... effettua altro lavoro ...
}
else {
    perror("Errore msgrcv");
}
```

Occorre controllare anche la **variabile globale "errno"**.

Oltre alla assenza di messaggi, possono verificarsi **altre "anomalie"** (mancanza di permessi, etc.)

Flags (msgrcv)

RETURN VALUE top

Upon successful completion, *msgrcv*() shall return a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv*() shall return -1, and *errno* shall be set to indicate the error.

ERRORS top

The *msgrcv*() function shall fail if:

E2BIG The value of *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is 0.

EACCES Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EIDRM The message queue identifier *msqid* is removed from the system.

EINTR The *msgrcv*() function was interrupted by a signal.

EINVAL *msqid* is not a valid message queue identifier.

ENOMSG The queue does not contain a message of the desired type and (*msgflg* & IPC_NOWAIT) is non-zero.

https://man7.org/linux/man-pages/man3/msgrcv.3p.html

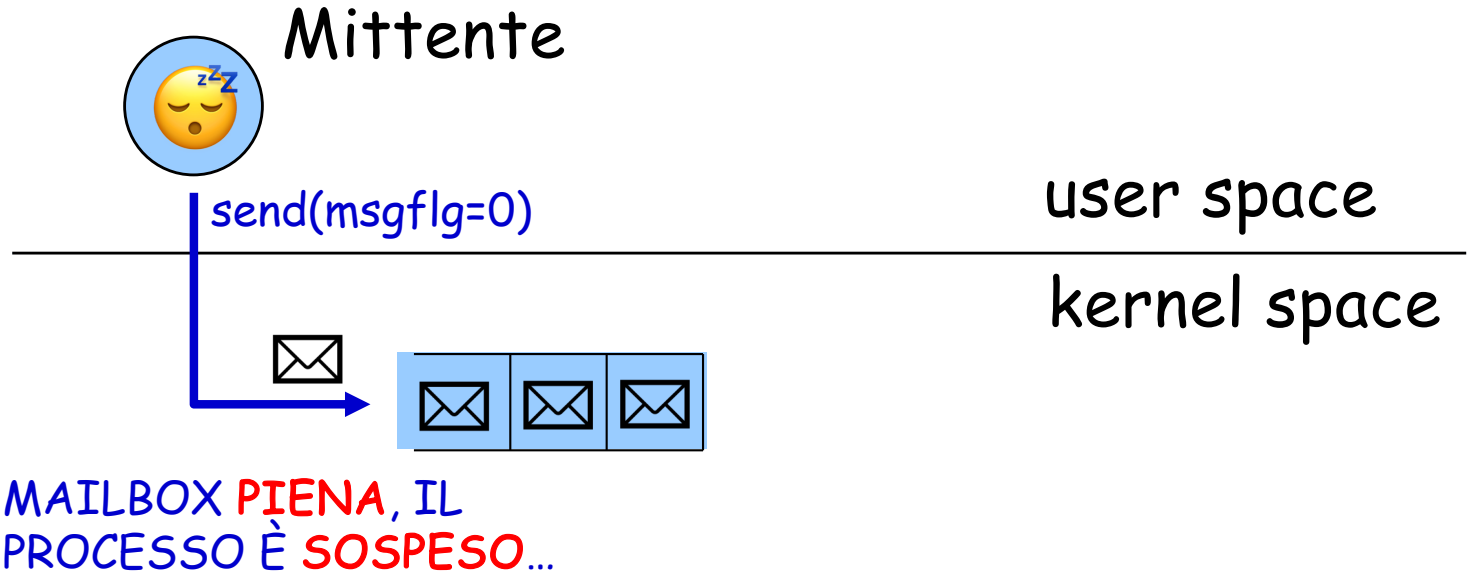
Flags (msgsnd)

- Il parametro ***msgflg*** permette di cambiare il comportamento di msgsnd() in caso di **mailbox piena**

```
int msgsnd(int msqid,
           struct msgbuf * msgp,
           size_t msgsz,
           int msgflg)
```

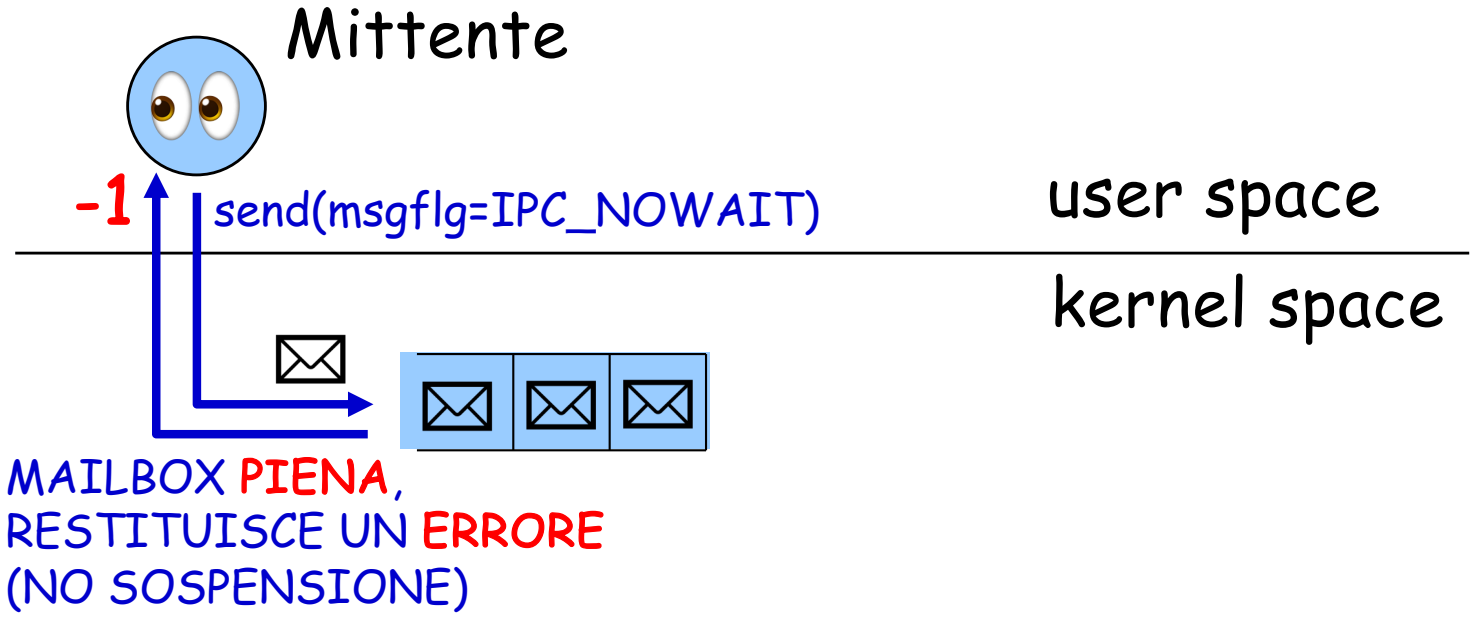
Flags (msgsnd)

- `msgflg = 0`: si sospende il processo **solo nel caso che la mailbox sia già piena**
- Raggiunto il limite di messaggi accodabili dal SO



Flags (msgsnd)

- *msgflg* = IPC_NOWAIT: non si sospende mai il processo, neanche se mailbox è piena
- msgsnd() ritorna il valore "-1"



Flags (msgsnd)

Nota importante

La msgsnd() di UNIX è **sempre asincrona** (indipendentemente da IPC_NOWAIT)

Per ottenere una **send sincrona**, è necessario applicare lo schema del **rendezvous**

Errori comuni

- Le primitive per lo scambio di messaggi sono inclini a errori di utilizzo, a causa delle **convenzioni** da rispettare
- Il mancato rispetto di queste regole spesso non è rilevato dal compilatore, e porta a malfunzionamenti difficili da interpretare:
  - **Corruzione del contenuto dei messaggi**. Ad esempio, la lettura di campi numerici nel messaggio produce valori inattesi e negativi, come «-143214321».
    È indicativo che l'area di memoria che il programma sta leggendo non è valida (es. non è stata mai riempita con dei dati)
  - **Stallo** oppure **terminazione prematura** della primitiva msgrcv(), nonostante vi siano messaggi
  - **Valori di ritorno negativi** dalle chiamate a msgsnd() e msgrcv()

Errori comuni

- **Mancanza del campo "tipo" nella definizione del messaggio**
  - È richiesto che la struttura-messaggio **includa sempre un campo "tipo"**, anche se non se fa uso
  - Se la omettiamo, il SO interpreterà comunque i primi byte della struttura come se fossero il campo tipo
- **Posizione non corretta del campo "tipo" nel messaggio**
  - È richiesto che il campo **"tipo" sia sempre il primo campo** all'interno della struttura messaggio, poiché il sistema operativo si aspetta di trovare nei primi byte della struttura il valore del tipo di messaggio
- **Campo "tipo" int invece che long**
  - È richiesto che il campo **"tipo" sia definito come long**. Non è corretto utilizzare "int", poiché su alcune architetture hardware i tipi int e long sono trattati diversamente dal compilatore
  - Ad esempio, nelle CPU a 32 bit, i tipi int e long hanno spesso la stessa dimensione (4 byte), mentre nelle CPU a 64 bit, i tipi int e long spesso hanno, rispettivamente, dimensione di 4 e 8 byte

Errori comuni

- **Campo "tipo" nel messaggio impostato a zero**
  - Il valore 0 è un valore riservato per usi speciali, e non va utilizzato come valore del tipo per i messaggi
  - Ad esempio, in msgrcv(), il parametro "msgtyp = 0" indica che si vuole ricevere messaggi di qualunque tipo
- **Campo "tipo" non inizializzato**
  - Se non inizializzato, il valore del campo tipo è imprevedibile e arbitrario (dipende dal SO, dall'hw, e dal compilatore)
- **Errato calcolo della dimensione del messaggio**
  - La dimensione di una struttura non è necessariamente uguale alla somma dei singoli campi (il compilatore fa delle ottimizzazioni)
  - Utilizzare "sizeof(messaggio) - sizeof(long)"

Errori comuni

- **Errato uso di puntatori alla struttura messaggio**
  - Ad esempio, passaggio erroneo di un "puntatore a puntatore" in ingresso a msgsnd/msgrcv
  - È un indirizzo ad una area di memoria che non contiene una struttura messaggio, ma dati arbitrari (e non validi)
  - Il sistema operativo interpreta l'area puntata come se fosse una struttura messaggio (ad esempio, i primi byte dell'area puntata sono interpretati come il "campo tipo" del messaggio)
  - Può causare l'invio/ricezione di messaggi corrotti, o il blocco di msgsnd/msgrcv

```
void myfunction(mymessage * msg) {
    ...
    // ERRORE! "msg" è già un puntatore, non serve "&"
    msgsnd(msqid, &msg, sizeof(mymessage)-sizeof(long), 0);
    ...
}
```

Errori comuni

- **Errato uso degli identificatori delle code**
  - Quando si usano molte risorse UNIX (es. molte code), può capitare di riutilizzare la stessa chiave su chiamate diverse
  - Oppure, può capitare di usare un msgid diverso da quello che si intendeva usare

```
key_t prima_chiave = ftok("./", 'a');

int richieste_id = msgget(prima_chiave, IPC_CREAT|0644);

key_t seconda_chiave = ftok("./", 'b');

// ERRORE! Intendevamo usare "seconda_chiave" qui
int risposte_id = msgget(prima_chiave, IPC_CREAT|0644);
```