

# Progetto Sistemi Operativi

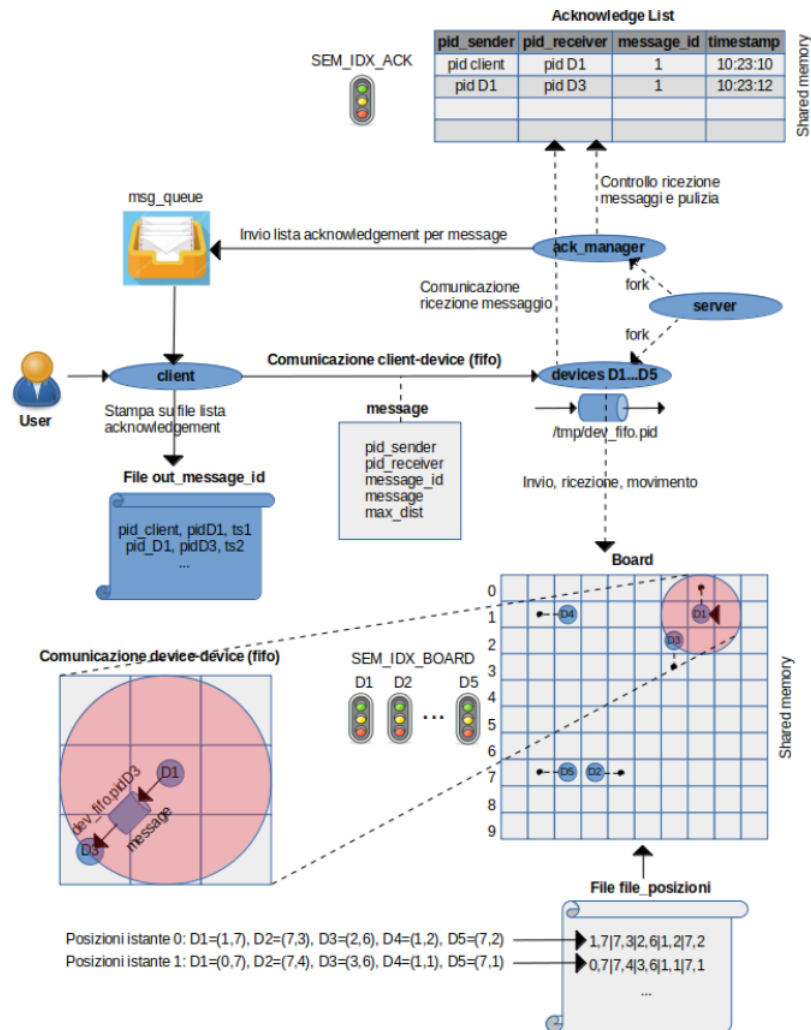
Davide Bleggi

May 17, 2020

# 1 Introduzione

Trasferimento di messaggi tra dispositivi (2.1). I device si muovono all'interno di una scacchiera.

## 1.1 Schema funzionamento



# 2 Elementi

## 2.1 Device

5 Processi figli del processo server (2.2). Ognuno gestisce la propria FIFO (2.6.2). Invia i messaggi che ha (e che gli altri non hanno) agli altri device nell'area del suo raggio di comunicazione (2.1.2) e così faranno anche gli altri. I dispositivi devono poter memorizzare e gestire più messaggi contemporaneamente.

### 2.1.1 Funzionamento

1. Controllo dell'acknowledgement (2.4) per verificare a quale device il messaggio non sia ancora arrivato.
2. Invio dei messaggi (se disponibili) tramite le relative FIFO dei messaggi.
3. Ricezione di messaggi
4. Movimento

### 2.1.2 Device $\rightarrow$ Device

Il nome della FIFO è *dev\_fifo.pid*. La fifo è contenuta in */tmp/dev\_fifo.pid*. Deve sempre rimanere aperta.

**Messaggio** La struttura del messaggio inviato è:

- *pid\_sender*: pid del device
- *pid\_receiver*: pid del del device ricevente
- *message\_id*: id del messaggio
- *message*: stringa di testo.
- *max\_dist*: numero positivo = raggio di invio del messaggio.

## 2.2 Server

Processo padre dei device 2.1 e di Ack\_manager 2.3. Genera i segmenti di memoria relativi a acknowledge 2.4 e board 2.5. Crea i semafori per l'accesso ai segmenti di memoria in acknowledge, in board e al cambio posizione (movimento). Termina solo con SIGTERM (2.2.1). Scandisce il tempo dei movimenti (fa muovere device1 ogni due secondi che farà partire tutti gli altri a cascata). Ogni due secondi stampa le posizioni dei devices e gli id dei messaggi in essi contenuti.

### Esempio stampa

```
# Step i: device positions #####
pidD1 i_D1 j_D1 msgs: lista message_id
pidD2 i_D2 j_D2 msgs: lista message_id
pidD3 i_D3 j_D3 msgs: lista message_id
pidD4 i_D4 j_D4 msgs: lista message_id
pidD5 i_D5 j_D5 msgs: lista message_id
#####
```

### Avvio

```
./server msg_queue_key file_posizioni
```

### 2.2.1 SIGTERM

- Termina processi devices (2.1)
- Termina ack\_manager (2.3)
- Termina coda di messaggi (2.3)
- Termina le FIFO (2.1)
- termina memoria condivisa (2.5, 2.4)
- Termina semafori

## 2.3 Ack\_manager

Processo figlio del processo Server2.2 Gestisce la lista condivisa di acknowledgement (2.4). Scandisce ad intervalli regolari di 5 secondi la lista 2.4 per controllare se tutti i dispositivi hanno ricevuto il messaggio. In caso positivo invia subito la lista di acknowledgements al Client (2.3.1). Ack\_manager comunica con Client tramite coda di messaggi. "Libera" , contrassegnando, i messaggi coinvolti dalla lista condivisa(2.4).

Ogni cliente deve ricevere la lista relativa al messaggio che ha immesso nel sistema.

### 2.3.1 Ack\_manager → Client

Il nome della coda di messaggi è *msg\_queue*

## 2.4 Acknowledgment List

Segmento di memoria condivisa generato da server 2.2. Gestisce il tracciamento di messaggi tra devices.

Numero finito di messaggi contenibili.

**Messaggio** La struttura del messaggio inviato è:

- *pid\_sender*: pid del device
- *pid\_receiver*: pid del del device ricevente
- *message\_id*: id del messaggio
- *date\_time*: data e ora di un passaggio

**Struttura dati messaggio** Quindi la struttura dati è:

```
typedef struct {
    pid_t pid_sender;
    pid_t pid_receiver;
    int message_id;
    time_t timestamp;
} Acknowledgment;
```

## 2.5 Board (Scacchiera)

Scacchiera 10x10. Segmento di memoria condivisa generato da server 2.2. In posizione  $i,j$  ha scritto il PID del device (2.1) che è in quella posizione. Default cella = 0. I movimenti dei devices sulla scacchiera avvengono a turno ogni 2 secondi (tempo dato dal server). La sincronizzazione dei movimenti avviene tramite semaforo (2.5.2).

### 2.5.1 Posizioni

File posizione. Direttive di spostamento dei device.

#### formato

```
1,7|7,3|2,6|1,2|7,2
0,7|7,4|3,6|1,1|7,1
...
```

Ciascuna riga rappresenta la posizione (coordinate  $x,y$ , dove  $x$  è la riga ed  $y$  la colonna) dei 5 device nella scacchiera in un certo istante

### 2.5.2 Semaforo

Il semaforo si chiama *SEM\_IDX\_BOARD*.

## 2.6 Client

Processo generato dall'utente. Il client comunica con il Device tramite FIFO (2.6.2). Più client possono inviare messaggi contemporaneamente ai dispositivi. Quando riceve il messaggio da parte di *ack\_manager* (2.3), genera un file di nome *out\_message\_id.txt* (2.6.1) dove *message\_id* è l'id del messaggio. Una volta generato il file il client termina.

**Il message\_id deve essere univoco.**

#### Avvio

```
./client msg_queue_key
```

#### Richiesta client → utente

- Inserire pid device a cui inviare il messaggio (pid\_t pid\_device)
- Inserire id messaggio (int message\_id)
- Inserire messaggio (char\* message)
- Inserire massima distanza comunicazione per il messaggio (double max\_distance)

### 2.6.1 Output file

Lista di 5 acknowledgement che identificano i passaggi fatti dal messaggio con i relativi istanti di tempo.

## Formato

Messaggio 'message\_id': 'message'

Lista acknowledgment:

pid\_client, pid D1, date\_time

pid D1, pid D2, date\_time

pid D2, pid D3, date\_time

pid D3, pid D4, date\_time

pid D4, pid D5, date\_time

### 2.6.2 Client → Device

Il nome della FIFO è *dev\_fifo.pid*.

**Messaggio** La struttura del messaggio inviato è:

- *pid\_sender*: pid del client
- *pid\_receiver*: pid del del device ricevente
- *message\_id*: id del messaggio
- *message*: stringa di testo.
- *max\_idist*: numero positivo = raggio di invio del messaggio.

**Struttura dati messaggio** La struttura dati è quindi:

```
typedef struct {  
    pid_t pid_sender;  
    pid_t pid_receiver;  
    int message_id;  
    char message[256];  
    int max_distance;  
} Message;
```

## 3 Ipotesi

### 3.1 Client

Input arg *msg\_queue\_key*

**Pseudo codice**

```
// Struttura  
struct message = {pid_t pid_sender, pid_t pid_receiver,  
int message_id, char message[256], int max_dist)  
  
// Scrittura del messaggio  
Crea this_message;  
Chiedi informazioni(this_message);  
my_message_id = this_message.message_id;
```

```

Dev_FIFO = Apri in scrittura la "dev_fifo".(FIFO this_message.pid_receiver);
Scrivi su Dev_FIFO (this_message);

// Lettura conferma
Crea msg_queue(msg_queue_key);
do{
    Attendi in lettura sulla msg_queue -> acknowledge *Acknowledge_list;
}while (Acknowledge_list[0].message_id != this_message.message_id);

// Scrittura su file
file = crea "out"_(this_message.message_id);
while (Acknowledge_list[n] != NULL)
    Stampa su file(Acknowledge_list[n]);

Chiudi FIFO;
Chiudi file;
Termina;

```

### 3.2 Server

Input arg *msg\_queue\_key, file\_posizioni*

#### Pseudo codice

```

// Scacchiera
Board = pid_t [10][10];
Key Board_memory = Genera segmento di memoria Board;
Crea semaforo SEM_IDX_BOARD(0,0,0,0,0);
// Acknowledge_list
Crea semaforo SEM_IDX_ACK;
Key Acknowledge_memory = Genera segmento di memoria Acknowledge_list[100];
pid_t Device[5];

// Apri file posizioni
Apri file_posizioni;

// Crea 5 processi figlio
for (d = 0; d < 5, d++){
    Device[d] = fork();

    // Crea un processo figlio
    if (Device[d] == 0){
        // Crea la fifo
        Crea la FIFO "/tmp/dev_fifo".getpid();
        // !Inizializzo struct message message_list[100];
        // continua finche non viene terminato dal server
        while(1){
            // Crea il proprio posto nella scacchiera
            Leggi da file_posizioni i,j lo spazio d-esimo;
            Board[i][j] = getpid();

```

```

Controlla SEM_IDX_ACK;

Leggo la FIFO aperta;
// controllo se nel raggio d'azione
    sono presenti altri device
n = 0;
while (message_list[n+1] != NULL){
    // Controlla la scacchiera alla ricerca delle
    // poszioni degli altri device
    for(i = 0; i < 10; i++){
        for (j = 0; j < 10; j++){
            // Controlla il device trovato
            if(Board[i][j] == message_list[n].pid_receiver &&
                sqrt((i*i + j*j)) < message_list[n].max_dist){

                // invia messaggio e scrivilo su acknowledge list
                Controlla SEM_IDX_ACK;
                Scrivi message_list[n]
                    sulla FIFO "/tmp/dev_fifo".list[n].pid_receiver;
                Libera SEM_IDX_ACK;
                Crea un new_acknowledgement con message_list[n];
                Scrivi new_acknowledgement su Acknowledge_list;
            }
        }
        n++;
    }
    Libera SEM_IDX_ACK del device successivo;
    // DA CREARE HANDLER per la rimozione
    // della FIFO relativa al processo
}

struct identify {
    int message_id;
    pid_t pid[5] = {0};
}
ack_mager =fork();
if(ack_manager == 0){
    while(1){
        Crea message_queue (message_queue_key);
        Legge acknowledgement_list;
        struct indentify Array_code[100];
        while (acknowledgement_list[i] != NULL)
            while( Array_code[j].message_id != acknowledgement_list[i].message_id)
                j++;
            if ( Array_code[j] == NULL){
                Array_code[j].message_id = acknowledgement_list[i].message_id;
                for(n = 0; n < 5; n++){
                    if (Array_code[j].pid == Device[n]){

```



```

        Array_code[j].pid[n]++;
        if (Array_code[j].pid[n] > 0){
            counter++;
        }
        if (couter == 5){
            invia tutti gli acknowledge con questo message_id al client;
            Contrassegna come letti i rimovibili gli acknowledge;
        }
    }
    if(Array_code[j].message_id == acknowledgement_list[i].message_id){
        for(n = 0; n < 5; n++){
            if (Array_code[j].pid == Device[n]){
                Array_code[j].pid[n]++;
            }
        }
    }
}

    i++;
}
// DA CREARE HANDLER per la rimozione della message queue;
}

}

while (ci sono righe in file poszioni){
    print("# Step i: device positions #####");\\

    for(i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            if(Board[i][j] == D[n])
                leggi FIFO relativa a D[n];
                print (D[n] i j msgs: lista message_id);

    print("#####");
    sleep(2);
    SEM_IDX_BOARD[n] (1,0,0,0,0);
    n = 0;
}

Termina Ack_manager;
Termina Devices,
Rimuovi SEM_IDX_BOARD;
Rimuovi Acknowledge_access;
Rimuovi Board_memory;
Rimuovi Acknowledge_memory;
Termina;

```