Progetto Sistemi Operativi

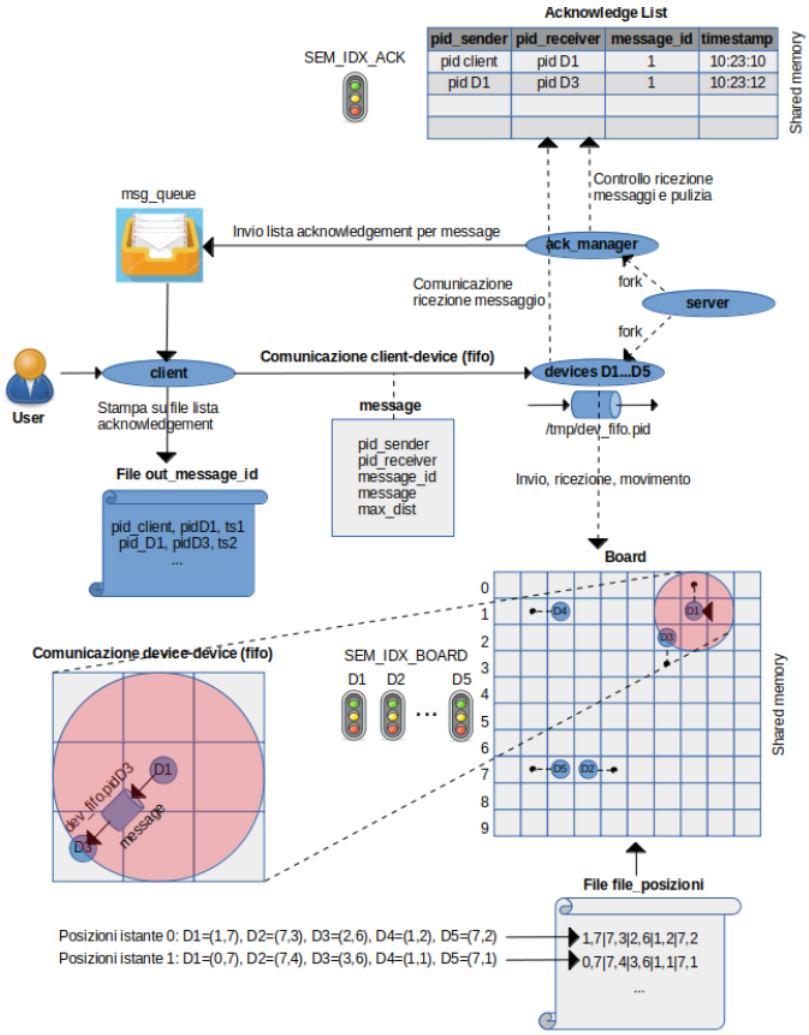
Davide Bleggi

May 9, 2020

1 Introdzione

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 hano) 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 $\longrightarrow$ 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 ackowledge, in board e al cambio posizone (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 ackowlodgement (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 *o*ut_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 $\longrightarrow$ 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;
```