PROGETTO SISTEMI OPERATIVI – LINUX SYSTEM CALL 2019/2020

- **Richiesta partecipazione esami-orali entro Venerdì 15 Maggio 2020, 23:59**
- **Prenotazione slot nel calendario esami-orali entro Sabato 31 Maggio 2020, 23:59**
- **Consegna elaborato: entro il Giovedì prima della settimana dell'esame (vedere date su sito e-learning del corso)**
- **Date esami: 22 Giugno – 17 Luglio 2020**

Descrizione generale

Si vuole realizzare un'applicazione per il trasferimento di messaggi tra 5 dispositivi mobili (chiamati device D1, …, D5 in figura).

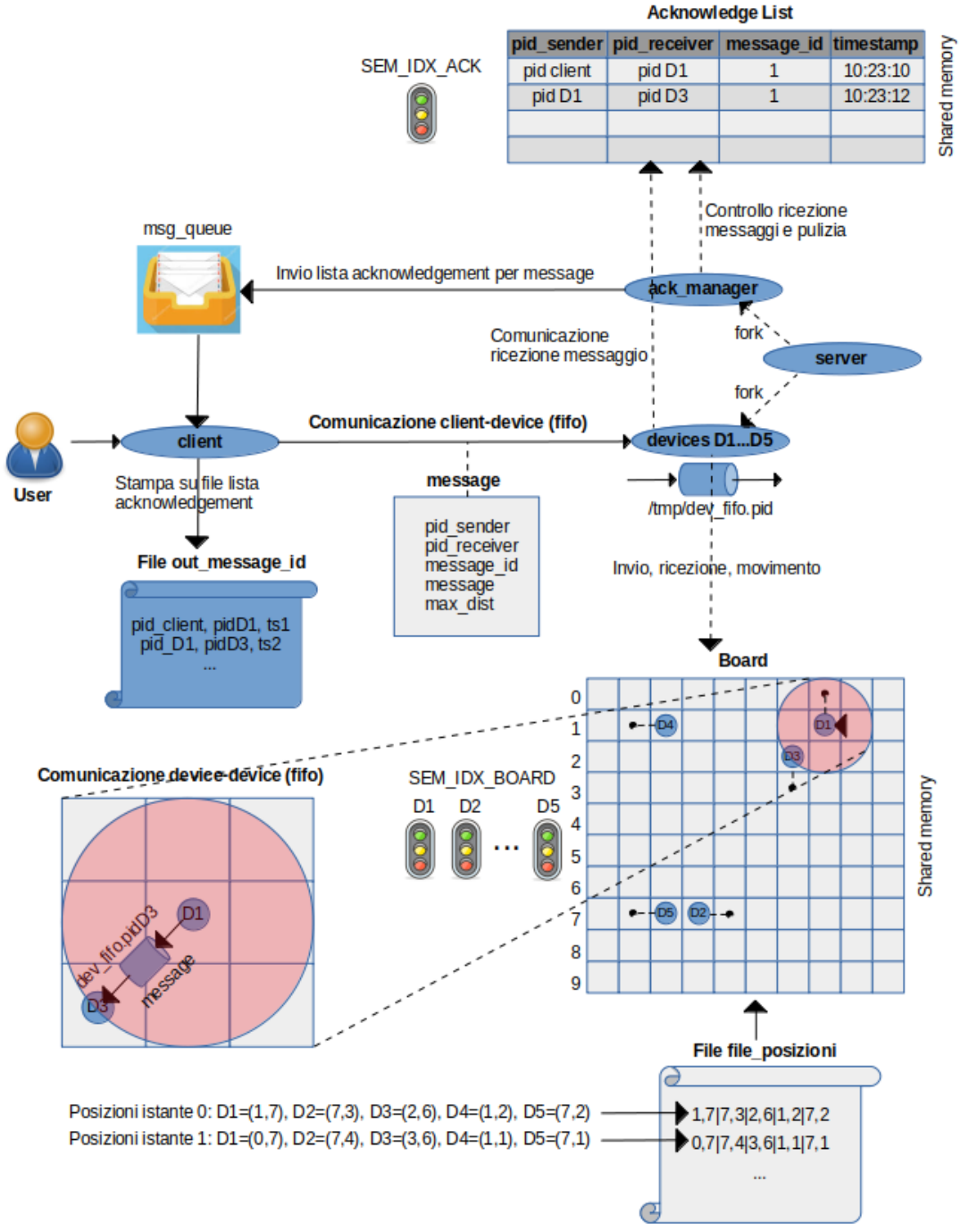

Figura 1: architettura generale del sistema da sviluppare

I **device** si muovono all'interno di una scacchiera 10x10 (denominata **board** in figura) seguendo le direttive (posizioni) scritte in un **file posizioni** (file_posizioni in figura). La scacchiera è un segmento di memoria condivisa contenente una matrice 10x10, in cui in posizione (i,j) è scritto il pid del device che occupa tale cella. Gli indici delle celle partono dall'angolo in alto a sinistra. Se non vi è un device in una cella allora la cella contiene 0.

Mentre i device si muovono, un **client** può connettersi ad un device per mezzo della fifo *dev_fifo.pid*, gestita dal device stesso (pid è il pid del device), per passargli un messaggio (**message** in figura) contenente il proprio pid (*pid_sender*), il pid del device ricevente (*pid_receiver*), un id del messaggio (*message_id*), una stringa di testo (*message*), ed un numero positivo (*max_dist*).

Il device che possiede il messaggio deve individuare altri device nel suo raggio di comunicazione (i.e., distanza Euclidea sulle posizioni nella scacchiera minore di *max_dist*) e passare il messaggio ad uno di essi. Ciascun device che riceve il messaggio deve cercare un altro device nel proprio raggio di comunicazione che non lo abbia ancora ricevuto e passarglielo. La comunicazione client-device e device-device avviene tramite *fifo*.

Il tracciamento dei passaggi del messaggio tra devices avviene tramite la scrittura, da parte dei device, di **acknowledgement** su un segmento di memoria condivisa. Ciascun acknowledgement contiene *pid_sender, pid_receiver, message_id, date_time* di un passaggio del messaggio (*date_time* è relativo all'istante in cui il ricevente ha ricevuto il messaggio). L'acknowledgement viene scritto dal device ricevente al momento della ricezione del messaggio.

Il processo **ack_manager** gestisce la lista condivisa di acknowledgement. In particolare, scandisce ad intervalli regolari di 5 secondi la lista per controllare se tutti i dispositivi hanno ricevuto il messaggio. In caso positivo invia subito la lista di acknowledgements al client per mezzo di un unico messaggio depositato nella coda di messaggi **msg_queue** e rimuove (marcandoli come liberi) gli acknowledgement relativi al messaggio stesso dalla memoria condivisa.

Quando il client riceve il messaggio nella coda di messaggi genera un file **out_message_id.txt** (dove *message_id* è l'identificativo del messaggio) e vi scrive la lista di 5 acknowledgement che identificano i passaggi fatti dal messaggio con i relativi istanti di tempo. Una volta generato il file di output il client termina.

I movimenti dei device nella scacchiera avvengono ogni 2 secondi e sono sincronizzati tra device in modo che prima si muova il primo processo, poi il secondo e cosi via fino al quinto. La sincronizzazione deve avvenire tramite semafori (SEM_IDX_BOARD in figura).

L'ordine delle azioni che deve eseguire ciascun device è il seguente: invio dei propri messaggi (se disponibili) a device vicini, ricezione di messaggi da device vicini, movimento.

Più client possono inviare messaggi contemporaneamente ai dispositivi, quindi i dispositivi devono poter memorizzare e gestire più messaggi contemporaneamente.

I processi ack_manager e devices sono figli del processo **server** che genera anche i segmenti di memoria condivisa relativi alla scacchiera ed alla lista di acknowledge, ed i semafori per gestire l'accesso ai segmenti di memoria condivisa ed il movimento dei device.

Il processo server può essere terminato solamente per mezzo di un segnale SIGTERM. La ricezione di tale segnale da parte del server gestisce la terminazione di tutti i processi device, del processo

ack_manager e la chiusura di tutti i meccanismi di comunicazione/sincronizzazione tra processi (memoria condivisa, semafori, fifo, etc.). Ogni altro segnale non strettamente necessario per l'esecuzione del programma deve essere bloccato.

Il tempo dei movimenti è scandito dal server. Uno step è eseguito ogni 2 secondi. Ad ogni step si esegue la lista di 5 movimenti relativi ad una riga del file di posizioni. I movimenti sono eseguiti nell'ordine dei device D1, D2, D3, D4, D5 (quando D1 ha fatto il movimento si muove D2, ecc.).

Riassunto elementi principali del progetto:

I processi attivi durante l'esecuzione del progetto sono *7+c*, dove *c* è il numero di client eseguiti. Tali processi sono:

- server
- ack_manager
- 5 devices (D1, … D5)
- *c>=1* processi client

Le aree di memoria condivisa sono 2, una per la scacchiera ed una per la gestione degli acknowledgment.

I semafori sono 7: uno per gestire l'accesso alla scacchiera, uno per gestire l'accesso alla lista di acknowledgments, e 5 per gestire la sincronizzazione dei movimenti dei device.

Esecuzione programmi e relativi input

Inizialmente si esegue il programma server per mezzo del comando:

./server msg_queue_key file_posizioni

dove

- *msg_queue_key è la chiave della coda di messaggi*
- *file_posizioni* è il percorso del file delle posizioni

Tale comando produce l'avvio del server, di ack_manager e dei 5 device che iniziano a muoversi nella scacchiera. Le chiavi dei segmenti di memoria condivisa e dei semafori sono generate dal server.

Successivamente si esegue un client per mezzo del comando:

./client msg_queue_key

dove *msg_queue_key* è la chiave della coda di messaggi da cui il client attende la lista di acknowledgment. Tale comando richiede le seguemti informazioni all'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)

poi prepara il messaggio (inserendo anche il proprio pid) e lo invia alla fifo del relativo device /tmp/def_fifo.pid_device. Il client poi resta in attesa del messaggio contenente gli acknowledgment dalla coda di messaggi. *R*icevuto il messaggio il client salva su file la lista di acknowledgment e termina.

Infine si termina il server, ack_manager e devices inviando un segnale SIGTERM al processo server il quale gestisce la chiusura dei processi figlio e la rimozione/chiusura dei meccanismi di comunicazione tra processi (i.e., fifo, memoria condivisa, code messaggi, semafori, etc).

Dettagli Server:

- Ad ogni step (cioè ogni 2 secondi, e quindi per ogni riga del file di posizioni) il server deve stampare a video le posizioni dei devices e gli id dei messaggi da essi contenuti nel formato:
  # 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
  ##
- La gestione del tempo per per l'esecuzione dei movimenti dei device deve essere eseguita dal server. Quindi il server, ogni 2 secondi, deve attivare il movimento del primo device, che genera poi in cascata l'attivazione dei movimenti di tutti i device (i.e., quando il primo device ha terminato il proprio movimento attiva il movimento del secondo device, e via di seguito fino al quinto device). Poi si attende due secondi per eseguire i prossimi movimenti.
- Il server deve generare i processi figlio *ack_manager* e 5 *device*
- Il server deve generare i semafori usati per sincronizzare l'accesso alle due aree di memoria condivisa. Gli stessi devono poi essere rimossi prima della terminazione del server.
- Il server deve allocare i due segmenti di memoria condivisa per la scacchiera e la lista di acknowledgment e poi rimuoverli prima di terminare.

Dettagli Device:

- Ciascun device genera una fifo */tmp/dev_fifo.pid*, dove *pid* e' il pid del device stesso, e la apre in lettura. **NB: tale fifo viene utilizzata per ricevere messaggi dal client o da altri devices, quindi deve rimanere attiva anche se nessun processo ha aperto la fifo in scrittura.** Ciascuna fifo deve poi essere rimossa dal device che l'ha generata.
- Ciascun device deve poter gestire simultaneamente più messaggi poiché più client possono immettere nel sistema messaggi che verranno passati tra i device in modo indipendente.
- Ciascun device scrive nella lista di acknowledgment (segmento di memoria condivisa) un acknowledgment (pid_sender, pid_receiver, message_id, timestamp) ogni volta che riceve un messaggio.
- Al momento dell'invio dei propri messaggi ciascun device accede, bloccando opportunamente i relativi semafori, alla scacchiera per cercare i device nel suo raggio di comunicazione, ed alla lista di acknowledgment per identificare i device che non hanno ancora ricevuto i propri messaggi.
- Il device, una volta identificati i device a cui inviare i propri messaggi, apre le relative fifo in scrittura e vi scrive all'interno i relativi messaggi.
- L'ultimo device che riceve un messaggio (e che quindi non può più inviarlo a nessun'altro device) lo deve cancellare (modifica 11/05/2020).

Dettagli gestore acknowledgment (ack_manager):

- Il gestore degli acknowledgement deve generare la coda di messaggi con cui invia i messaggi (lista acknowledgmemnt) ai relativi client che hanno immesso i messaggi nel sistema. La coda di messaggi deve essere rimossa prima della terminazione del programma.

- Il gestore degli acknowledgment deve utilizzare opportuni metodi per fare in modo che ciascun client riceva solo la lista di acknowledgment relativi al messaggio che ha immesso nel sistema.
- La lista di acknowledgement ha dimensione finita, ad esempio può contenere al più 100 acknowledgement, quindi ogni volta che un messaggio è passato da tutti i device ed i relativi acknowledgment sono stati inviati al client, il gestore degli acknowledgment "libera" i relativi acknowledgment marcandoli in modo opportuno.
- ~~Il gestore degli acknowledgement deve rimuovere il messaggio dall'ultimo device che lo ha ricevuto, prima di inviare la relativa lista di acknowledgment al client.~~ (Modifica 11/05/2020).

Dettagli client

- Il client deve chedere all'utente le informazioni necessarie per inviare un messaggio ad un device (vedi sezione S*equenza esecuzione programmi e relativi input*), aprire la relativa fifo in scrittura, inviare il messaggio e poi attendere dalla coda di messaggi la relativa lista di acknowledgment. Una volta ricevuta, deve stamparla su file e terminare.
- Moltepici client possono essere eseguiti in modo concorrente per inviare più messaggi (con diverso id) a più device.
- Il message_id passato dall'utente al client deve essere univoco

Strutture dati principali

- I messaggi passati da client a device o tra due device devono avere la seguente struttura dati:

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

- Gli acknowledgment salvati nella memoria condivisa devono avere la seguente struttura dati:

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

Formato file input

Il file delle posizioni dei device deve avere il seguente formato (vedere file d'esempio)

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

in cui 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. Nell'esempio sopra:

- all'istante 0 il device D1 si trova in posizione (1,7), il device D2 in posizione (7,3), il device D3 in posizione (2,6), il device D4 in posizione (1,2), il device D5 in posizione (7,2)
- all'istante 1 il device D1 si trova in posizione (0,7), il device D2 in posizione (7,4), il device D3 in posizione (3,6), il device D4 in posizione (1,1), il device D5 in posizione (7,1)

Il file delle posizioni deve essere generato dallo studente. I devices possono anche rimanere fermi, quindi in due righe successive potremmo avere le stesse coordinate x,y per alcuni device.

Formato file "out_message_id.txt"
Il file di output generato dal client deve avere il seguente 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

Esempio:
Messaggio 5: Ciao come stai?
Lista acknowledgment:
134, 156, 2020-05-02 18:10:23
156, 123, 2020-05-02 18:10:28
123, 166, 2020-05-02 18:10:31
166, 198, 2020-05-02 18:10:46
198,111, *2020-05-02 18:11:02*

Altre informazioni
Tutto ciò non espressamente specificato nel testo del progetto è a scelta dello studente.

È vietato l'uso di funzioni C per la gestione del file system (e.g. fopen). Il progetto deve funzionare su sistema operativo Ubuntu 18 e Repl, rispettare il template fornito, ed essere compilabile con il comando *make.*

L'ammissione all'esame orale è possibile solo se rispettata la data di consegna dell'elaborato.

Il presente elaborato permette un voto massimo di 25 trentesimi. La consegna dell'esercitazione su MentOS/process-management, e del presente elaborato permette un voto massimo di 27 trentesimi. La consegna dell'esercitazione su MentOS/process-management, MentOS/memory-management, e del presente elaborato permette un voto massimo di 30 e lode.

Consegna elaborato e-learning
Si richiede di rispettare la seguente struttura di cartelle per la consegna dell'elaborato:
/sistemi_operativi
    /system-call/ (elaborato system call)
    /MentOS/scheduler_algorithm.c
    /MentOS/buddysystem.c
La directory sistemi_operativi deve essere compressa in un archivio di nome *<matricola>_sistemi_operativi.tar.gz*. L'archivio deve essere creato con il comando tar (no

programmi esterni). L’archivio tar.gz deve essere caricato nell’apposita sezione sul sito di e-learning.