Sistemi Operativi Avanzati

Progetto A.A. 2020/2021

Paolo Melissari 0287567

Build e Test del sistema

I comandi per la compilazione del sottosistema sono presenti nel *Makefile.*

In particolare, tramite i comandi:

make; make load

è possibile compilare e caricare il modulo kernel implementato.

Nella cartella *user/test* è presente un codice per far girare una demo che:

1. Crea un certo numero di TAG services
2. Lancia dei thread in ascolto su alcuni TAG services
3. Lancia dei thread in scrittura su alcuni TAG services
4. Chiama il comando AWAKE_ALL per risvegliare tutti i thread che non hanno letto
5. Elimina tutti i TAG services creati

Nella cartella *user/commands* sono disponibili degli eseguibili per utilizzare le funzioni implementate direttamente da linea di comando, ovvero:

- **set.out**: richiama la funzione *tag_get(int key, int command, int permission)*
- **send.out**: richiama la funzione *tag_send(int tag, int level, char* buffer, size_t size)*
- **receive.out**: richiama la funzione *tag_receive(int tag, int level, char* buffer, size_t size)*
- **ctl.out**: richiama la funzione *tag_ctl(int tag, int command)*

Breve descrizione del sistema

Il sistema è composto da 3 parti:

SysCall Discovery

Ha la funzione di cercare la syscall table in memoria, e riprende le funzionalità del codice visto a lezione (https://github.com/FrancescoQuaglia/Linux-sys_call_table-discoverer). Permette di aggiungere delle nuove syscall nella syscall table.

TAG Subsystem

Implementa le funzioni richieste dal progetto. I TAG Service sono definiti come in Figura 1.

```
typedef struct TAG_Service{
  int key;              // chiave del tag service
  unsigned int tag;     // valore del tag
  int uid_Creator;   // PID del creatore
  int perm;    // permessi
  list levels; // lista contenente i 32 livelli, ognuno implementato come elemento di una lista RCU
} TAG_Service;
```

Figura 1. TAG Service

In particolare, ogni TAG Service contiene una lista RCU. Ogni elemento della lista RCU rappresenta un livello di comunicazione del TAG Service ed è definito come in Figura 2.

```
typedef struct _element{
    struct _element * next;
    int level;
    char* message;
    size_t size;
} element;
```

Figura 2. Elemento lista RCU

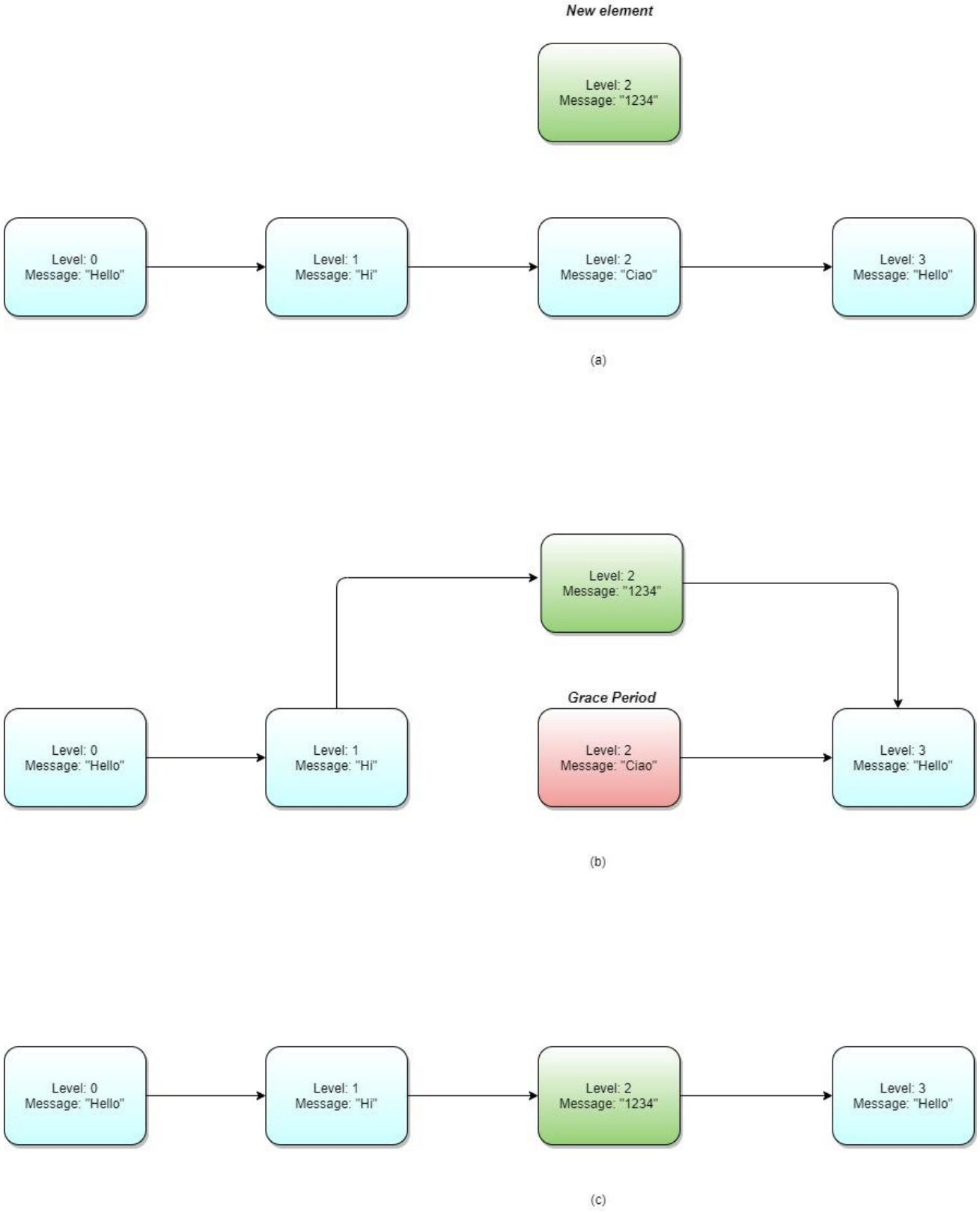
Quando un thread richiede la scrittura su livello di un certo TAG Service, cerca l'elemento che contiene i dati del livello richiesto ed effettua la sostituzione dell'elemento.

La sostituzione avviene come segue:

1. Viene creato il nuovo elemento e_i' per il livello i
2. Viene individuato l'elemento e_i da sostituire (ovvero l'attuale elemento presente nella lista che contiene i dati del livello i)
3. Viene fatto puntare l'elemento e_i' allo stesso elemento puntato da e_i
4. Viene considerato l'elemento e_{i-1} (ovvero l'elemento che precede l'elemento e_i da sostituire) e ne viene cambiato il puntatore verso l'elemento successivo, che diventerà e_i'.

5. Al termine del grace period, viene eliminato l'elemento e_i.

La Figura 3 mostra graficamente il processo:



È importante notare come nel punto (b), ovvero durante l'attesa del grace period, ci siano due versioni della lista:

- Quella contenente il "vecchio" livello 2, accessibile solo ai lettori che, durante la sostituzione, stavano ancora processando i dati del livello 2.
- Quella contenente il "nuovo" livello 2, accessibile a tutti i nuovi lettori.

CharDev

Device driver per ottenere uno snapshot dello stato del sistema. Implementa le seguenti funzioni:

- Open: viene salvato lo stato nel sistema nel campo *private_data* della *struc file*
- Read: viene letto il contenuto del campo *private_data*
- Release: cancella l'area di memoria allocata per la memorizzazione dello stato del sistema