Relazione Progetto di Sistemi Operativi 2019-20

Vincenzo Gargano
Alessandro Bongiovanni

November 2020

Contents

1 Introduzione

In questa relazione parleremo della realizzazione del progetto di SO

2 Inizializzazione e preparazione del gioco

2.1 Processo Principale

Il processo principale **Main** si occupa di inizializzare le strutture dati, e chiamare la funzione ***master_handler()*** del processo **Master**, quando sarà terminata la funzione ***master_handler()***, che alla fine fa una wait sui processi **Players**, il processo principale si occupa di distruggere tutte le strutture persisenti dedicate alla IPC che è stata utilizzata per la sincronizzazione.

2.2 Processo Master

Il processo master è incaricato di gestire lo svolgimento del gioco ed è invocato tramite la chiamata alla funzione ***master_handler()***. Inizialmente il master inizializza un semaforo di controllo sullo svolgimento del round a 1. Subito dopo resetta la scacchiera prima di cominciare a mandare il messaggio **MSG_PAWN_PLACEMENT** a tutti i processi player che procedono al piazzamento delle loro pedine in memoria condivisa. Successivamente viene invocata la funzione ***restart_game()*** che ha il compito di far ricominciare il round. Nel caso sia la prima esecuzione la funzione fa iniziare effettivamente il gioco: per prima cosa si occupa di creare tanti semafori quanti sono i giocatori presenti (2 in easy e 4 in hard); poi si occupa di piazzare le bandierine sulla scacchiera in memoria condivisa. Durante questa procedura viene chiamata una funzione che secondo i parametri di configurazione iniziale setta il numero di bandierine e il loro punteggio. Viene poi randomizzata la loro posizione e vengono posizionate sulla scacchiera. A questo punto viene inviato il messaggio **MSG_FLAG_PLACEMENT** a tutti i processi player che vengono appunto notificati del fatto che tutte le bandierine sono state piazzate. Viene quindi invocata la funzione ***goal_assignment()*** che fornisce le informazioni sulla posizione delle bandierine ai player poplando il loro campo **goal_t* goals**.

A questo punto viene stampato lo stato della scacchiera, con pedine e bandierine in posizione e viene invocata la funzione ***start_round()***: questa setta inizialmente il timer allo scadere del cui viene inviato il messaggio **SIGALARM** che interromperà il gioco. Viene quindi inviato il messaggio **MSG_START_GAME** che notifica ai giocatori l'inizio del gioco.

Durante lo svolgimento del gioco, la funzione ***all_flag_captured()***, resta in attesa di del messaggio **MSG_FLAG_CAPTURED** che notifica il master che una bandierina è stata catturata. Quando il numero delle bandierine catturate è uguale al numero totale delle bandierine che erano inizialmente presenti sulla scacchiera viene interrotto il round. Vengono stampati i risultati del round appena concluso e con la funzione ***restart_game()*** viene fatto cominciare il nuovo round.

3 Schacchiera

3.1 Realizzazione

La **scacchiera** è stata realizzata come una matrice m*n. Ogni campo della scacchiera ha un tipo definito con una enum, questo può essere:

- **FLAG**
- **PAWN**
- **EMPTY**

Per far si che la scacchiera sia in memoria condivisa abbiamo creato un array che gestisce la matrice. Abbiamo definito una variabile (cells_t** cells), puntatore a puntatore, ed usiamo (cell_t *) per riferirci agli indirizzi di memoria per le righe di lunghezza m.

4 Sincronizzazione

4.1 Semaforo di inizio gioco

Questo semaforo binario è stato aggiunto quando ci siamo accorti che serviva sincronizzare la fine di un round con l'inizio di un altro. In quanto in ogni round i processi che entrano in sezione critica devono accorgersi che il gioco è sospeso. Questo semaforo è controllato da due funzioni: **pause_game()** e **resume_game()** e viene testato da una funzione **is_running**, la funzione per controllare lo svolgimento del gioco viene testata prima di ogni movimento delle pedine, cosi da impedire movimenti dopo la fine di un round.

4.2 Sincronizzazione sulla schacchiera

Per la sincronizzazione della scacchiera, abbiamo creato un set di semafori binari di dimensione n*m . Le due funzioni che usiamo per gestire questi semafori sono **lock_cell** e **unlock_cell**.

4.3 Sezione Critica: movimento

Per quanto riguardo il movimento della pedina, abbiamo usato i semafori della schacchiera, in modo che, se trovano una bandiera o una cella vuota continuino, mentre nel caso in cui incontrarno un altra pedina, cambiano direzione. Per impedire al segnale SIGARLM, catturato dal nostro alarm_handler avevamo pensato di usare una maschera di segnali che bloccasse usando **sigprocmask()**, prima di una sezione critica, tutti i segnali di tipo SIGARLM, e nel caso ci fosse un segnale in pending viene chiamata la funzione **pause_game()**, alla fine di quella sezione critica la maschera di segnali viene reimpostata

4.4 Coda di messaggi

Per far comunicare i vari processi abbiamo usato una coda di messaggi, usiamo due funzioni **send_message()** e **receive_message()** in cui la receive aspetta che ci sia abbastanza spazio per i messaggi nella coda. I messaggi sono visti come dei caratteri, e vengono inviati inizialmente per coordinare la creazione di giocatori e segnalare l'inizio del gioco.

5 Conclusioni

Durante lo svolgimento del progetto ci siamo trovati di fronte a molte difficoltà di implementazione che non sempre siamo riusciti a risolvere. Per questo il progetto è incompleto in alcune parti: il movimento delle pedine non è regolato da una logica o un algoritmo anche se inizialmente avevamo pensato a un modo semplice per far si che le pedine, utilizzando il loro campo **goal_t* goals**, contenente le coordinate delle bandierine sulla scacchiera, raggiungessero le bandierine più vicine, ordinando la lista di goals secondo la distanza dalla pedina stessa. Anche per questo motivo, le pedine spesso non riescono a prendere tutte le bandierine prima di **SO_MAX_TIME**.