RELAZIONE PROGETTO DI SISTEMI OPERATIVI ANNO 2016/2017

Nome: Valentino Di Cianni
Matricola: 800665
Email: valentino.dicianni@edu.unito.it

Introduzione:
La soluzione scelta per l'implementazione del progetto è schematicamente illustrata nello diagramma UML allegato alla relazione.

Ho utilizzato CMake per la build del progetto, facilmente eseguibile grazie allo script "setup" presente all'interno della cartella con i file sorgente.
Una volta eseguita la build del progetto, verrà prodotto un eseguibile "launcher" nella directory /cmake/bin/ che, se mandato in esecuzione proporrà all'utente un'interfaccia da linea di comando per la scelta dell'azione da intraprendere.

Come da specifica del progetto, è presente un file "config" nel quale sono presenti i dati di configurazione della partita. È possibile modificare i tassi di probabilità accedendo al file, mentre si può modificare la lunghezza del match anche da terminale nella schermata del manager di partita.
Inoltre è presente un file "header.h" nel quale sono specificate tutte le informazioni comuni come chiavi, struct, signature delle funzioni, che viene incluso da ogni classe.

Implementazione ed esecuzione:
Per quanto riguarda l'implementazione, tutto parte da un processo arbitro il quale imposta il punteggio iniziale, rappresentato da una struct Score, a 0-0, genera il semaforo pallone e i semafori delle squadre, li inizializza, e solo a quel punto inizia la creazione dei processi figli quali il fato e le squadre.
La creazione di questi ultimi, è affidata a due funzioni: createFato() e createSquadre() le quali eseguono una execve() per creare e mandare in esecuzione i rispettivi processi. Durante la creazione delle squadre viene passato come parametro ad ogni nuovo processo il numero di squadra che le identifica, rispettivamente 0 e 1.
Dopo aver creato i processi figli, l'arbitro imposta un timer alarm con il valore specificato nel file di configurazione, e si mette in attesa di ricevere un signal o nel caso in cui venga segnato un goal oppure nel caso in cui finisca il tempo a disposizione della partita.

Una volta creato, il fato genera la coda di messaggi con la chiave specificata nell' header file, crea un nuovo file di log, situato nella directory /log , e si mette in attesa di ricevere messaggi da parte dei giocatori. Quando il fato riceve un messaggio, a seconda dell'azione (1,2,3) ottiene la probabilità relativa dal file "config" e, tramite la funzione askFate(), restituisce 0 in caso di esito negativo, oppure 1 in caso di esito positivo. Successivamente, aggiunge al file di log un testo che descrive l'azione appena eseguita, e manda sulla coda di messaggi la risposta al giocatore, specificando il pid di questo affinchè il messaggio sia letto solo dal giocatore selezionato.

Nello stesso tempo, anche i processi squadra vengono creati. Questi, dopo aver recuperato il pool di semafori relativi alle squadre, creano 5 processi giocatore ciascuna e si mettono in lista per riservare i semafori nel caso un giocatore termini per infortunio.
I processi giocatore così creati, a turno provano a riservare il semaforo "pallone", invocano il metodo play(), e a seconda del numero che ricevono come valore di ritorno eseguono un'azione specifica. Se l'azione risulta essere dribbling e va a buon fine, il processo giocatore selezionato ripete la giocata play(). Nel caso sia segnato un goal, a seconda della squadra di appartenenza, viene generato un segnale SIGUSR1 o SIGUSR2, intercettato poi dall'arbitro. Se invece il giocatore subisce un infortunio, rilascia il semaforo del team di appartenenza, e termina. Il nuovo giocatore creato dalla squadra a seguito di ciò rilascia il pallone ed ha inizio una nuova fase di gioco.
Alla fine di ogni fase di gioco, il giocatore che deteneva il possesso del pallone rilascia il semaforo e si mette in sleep per evitare che il pallone venga riservato dallo stesso giocatore più di una volta consecutivamente, scongiurando così la starvation.

Il metodo play(), dopo aver determinato un'azione casuale da intraprendere, manda una richiesta di responso al fato tramite la coda di messaggi specificando IDFATE nel campo mtype del messaggio, e si mette in attesa che il fato risponda. A questo punto restituisce il risultato del responso: rispettivamente 0 o 1.
Tutto ciò viene eseguito finche i processi giocatore non ricevono un segnale di terminazione.

Terminazione:
La terminazione del programma parte dall'arbitro, allo scadere del timer: l'arbitro invia un segnale SIGALRM a tutto il gruppo di processi tramite la kill e si mette in attesa di ricevere la conferma da tutti i processi figli di avvenuta terminazione tramite waipid().

Quello che succede nei processi squadra, fato e giocatore è molto simile: il sigHandler() di ognuno imposta flagSignal = 1. Nei processi giocatore questo genera il completamento dell'azione in corso, se in fase avanzata, e la terminazione di tutti i processi in attesa sul semaforo pallone. Le squadre attendono la terminazione di tutti i processi giocatore, per poi terminare a loro volta, mentre il fato finisce eventualmente di dare l'ultimo responso, attende che sia letto, dealloca la coda di messaggi e termina.
L'arbitro, terminati tutti i suoi processi figli, comunica il risultato finale della partita, rimuove i semafori e termina l'esecuzione.