Sistemas Operativos

Simulação de jogo de futebol

Trabalho Prático 2 - Turma P1

Prof. José Nuno Panelas Nunes Lau

2020/2021

João Reis (98474)

Ricardo Rodriguez (98388)

Índice

1. Introdução

O segundo trabalho prático de Sistemas Operativos consiste na simulação de um jogo de futebol com três entidades diferentes: o *player*, o *goalie* e o *referee*.

O projeto, desenvolvido na linguagem de programação C, tem como principal objetivo perceber como é que os semáforos conseguem manipular e sincronizar múltiplos processos simultaneamente de forma a obter a solução desejada.

Para visualizar o resultado da execução de todos os processos corremos o programa *./probSemSharedMemSoccerGame* já compilado.

2. Variáveis gerais e de estados

No programa *probConst.h* temos definidas uma série de variáveis que precisam de ser utilizadas para o desenvolvimento do programa.

As variáveis gerais ditam o número máximo de *players*, *goalies* e de *referees*, bem como o número máximo de *players* e *goalies* por equipa. Os *players*/*goalies* podem ter vários estados possíveis: a chegar (ARRIVING), à espera de equipa (WAITING_TEAM), a formar equipa (FORMING_TEAM), à espera de começar o jogo numa determinada equipa (WAITING_START_1/WAITING_START_2), a jogar numa determinada equipa (PLAYING_1/PLAYING_2) e chegar atrasado (LATE). O mesmo acontece com o árbitro: a chegar (ARRIVING), à espera que as equipas sejam formadas (WAITING_TEAMS), a começar o jogo (STARTING_GAME), a arbitrar (REFEREEING) e, por último, a acabar o jogo (ENDING_GAME).

Desta forma, agrupámos os diferentes grupos destas variáveis em tabelas para uma melhor compreensão das mesmas.

Variáveis gerais

Variável	**Valor**
NUMPLAYERS	10
NUMGOALIES	3
NUMREFEREES	1
NUMTEAMPLAYERS	4
NUMTEAMGOALIES	1

Estado do player/goalie

Estado	**Valor**
ARRIVING	0
WAITING_TEAM	1
FORMING_TEAM	2
WAITING_START_1	3
WAITING_START_2	4
PLAYING_1	5
PLAYING_2	6
LATE	7

Estado do refere

Estado	**Valor**
ARRIVING	0
WAITING_TEAMS	1
STARTING_GAME	2
REFEREEING	3
ENDING_GAME	4

3. Estruturas de dados internos

Outro ficheiro igualmente importante é o *probDataStruct*, uma vez que contém as estruturas de dados essenciais para o programa.

O tipo *STAT* vai ser uma estrutura que vai armazenar os estados de todas as entidades do problema. Assim, este possui três parâmetros: um *array* de inteiros *unsigned playerStat* para os *players* com tamanho igual ao máximo de jogadores possível, outro *array* de inteiros *unsigned goalieStat* para os *goalies* com tamanho igual ao número máximo de guarda-redes possível e um inteiro para guardar o estado do único árbitro presente.

```
typedef struct {
    /** \brief players state */
    unsigned int playerStat[NUMPLAYERS];
    /** \brief goalies state */
    unsigned int goalieStat[NUMGOALIES];
    /** \brief referees state */
    unsigned int refereeStat;

} STAT;
```

O tipo de dados *FULL_STAT* vai guardar informação sobre praticamente todas as variáveis intrínsecas ao problema. Este guarda a estrutura de dados *STAT* apresentada anteriormente numa variável st, tendo informação sobre todos os estados atuais, guarda o número total de *players*, *goalies* e *referees*, o número de *players* e *goalies* que já chegaram, bem como o número de *players* e *goalies* disponíveis para formar uma equipa, e, por último, armazena o ID da próxima equipa a ser formada (inicializado com valor 1).

```
typedef struct
{   /** \brief state of all intervening entities */
    STAT st;

    /** \brief total number of players */
    int nPlayers;

    /** \brief total number of goalies */
    int nGoalies;

    /** \brief total number of referees */
    int nReferees;

    /** \brief number of players that already arrived */
    int playersArrived;
    /** \brief number of goalies that already arrived */
    int goaliesArrived;
    /** \brief number of players that arrived and are free (no team) */
    int playersFree;
    /** \brief number of goalies that arrived and are free (no team) */
    int goaliesFree;

    /** \brief id of team that will be formed next - initial value=1 */
    int teamId;

} FULL_STAT;
```

4. Definição de semáforos

Uma vez que, por vezes, precisamos que um processo ou condição seja executada primeiro antes que outra, é necessário o uso de semáforos. Estes permitem controlar os processos sequencialmente, bloqueando processos caso seja necessário.

A estrutura de dados *SHARED_DATA* contém uma variável *fSt* do tipo FULL_STAT e uma série de variáveis de inteiros que representam os semáforos:

Semáforo	Propósito do semáforo
mutex ***(valor inicial = 1)***	Quando entramos na região crítica este é decrementado através do *semDown*(), bloqueando qualquer outro processo que tente entrar numa região e protegendo a execução do processo atual. A saída desta região é marcada pelo incremento do semáforo com o *semUp*()
playersWaitTeam ***(valor inicial = 0)***	Utilizado para bloquear o processo do *player* enquanto este não tiver equipa formada, altura em que o semáforo é incrementado e o processo é libertado
goaliesWaitTeam ***(valor inicial = 0)***	Utilizado para bloquear o processo do *goalie* enquanto este não tiver equipa formada, altura em que o semáforo é incrementado e o processo é libertado
playersWaitReferee ***(valor inicial = 0)***	Utilizado para bloquear o processo dos *players* e *goalies* enquanto um *referee* não começar o jogo, altura em que o semáforo é incrementado e o processo é libertado
playersWaitEnd ***(valor inicial = 0)***	Utilizado para bloquear o processo dos *players* e *goalies* enquanto um *referee* não acabar o jogo, altura em que o semáforo é incrementado e o processo é libertado
refereeWaitTeams ***(valor inicial = 0)***	Utilizado para bloquear o processo do *referee* enquanto ambas as equipas não estiverem formadas, altura em que o semáforo é incrementado e o processo é libertado
playerRegistered ***(valor inicial = 0)***	Utilizado para bloquear o processo dos *players* e *goalies* enquanto estes não estiverem efetivamente inscritos numa equipa, altura em que o semáforo é incrementado e o processo é libertado

Antes de começar o projeto propriamente dito, tentamos completar a seguinte tabela para nos guiar e ajudar durante a implementação dos semáforos. Ao longo do projeto, fomos atualizando pequenas informações que só as percebemos quando enfrentados com certos problemas, por exemplo, no semáforo playersWaitTeam, a entidade que o desbloqueia depende de quem for o capitão de equipa, ou o player ou o goalie.

Semáforo	Entidade down	Função down	#down	Entidade up	Função up	#up
playersWaitTeam	Player	playerConstituteTeam	1	Player (se é capitão de equipa)	playerConstituteTeam	3
				Goalie (se é capitão de equipa)	goalieConstituteTeam	4
goaliesWaitTeam	Goalie	goalieConstituteTeam	1	Player (se é capitão de equipa)	playerConstituteTeam	1
				Goalie (se é capitão de equipa)	goalieConstituteTeam	0
playersWaitReferee	Player & Goalie	waitReferee	1	Referee	startGame	10
PlayersWaitEnd	Player & Goalie	playUntilEnd	1	Referee	endGame	10
RefereeWaitTeams	Referee	waitForTeams	2	Player (se é capitão de equipa)	playerConstituteTeam	1
				Goalie (se é capitão de equipa)	goalieConstituteTeam	1
PlayerRegistered	Player (se é capitão de equipa)	playerConstituteTeam	4	Player & Goalie	playerConstituteTeam & goalieConstituteTeam	1
	Goalie (se é capitão de equipa)	goalieConstituteTeam	4			

5. Entidades

5.1. Referee (SemSharedMemReferee.c)

Função *arrive*()

Nesta função, definimos o estado inicial do árbitro (ARRIVING). Ou seja, o arbitro está no estado 0, o que significa o arbitro está a chegar ao encontro. E guardados esse estado com o *saveState*.

```
static void arrive ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = ARRIVING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    usleep((100.0*random())/(RAND_MAX+1.0)+10.0);

}
```

Função waitForTeams()

Quando o árbitro chega ao encontro, este espera que ambas as equipas estejam prontas para jogar, ou seja, constituídas. Portanto, atualizamos o estado do árbitro para WAITING_TEAMS (1).

Em relação aos semáforos, o semáforo *refereeWaitTeams* decrementa uma unidade, bloqueando o próprio até uma equipa estar formada e este é desbloqueado. É importante referir que este semáforo é repetido duas vezes uma vez que há duas equipas e o processo do árbitro só pode ser desbloqueado quando ambas as equipas estejam formadas.

```c
static void waitForTeams ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = WAITING_TEAMS;
    saveState(nFic, &sh->fSt);


    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    // colocamos 2 vezes o semáforo down porque tem que esperar por 2 equipas

    if (semDown (semgid, sh->refereeWaitTeams) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    if (semDown (semgid, sh->refereeWaitTeams) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }
}
```

Função *startGame*()

Quando as equipas estiverem feitas, o árbitro tem autorização para começar o jogo. Primeiramente, muda o seu estado para STARTING_GAME (2) dentro da região crítica.

De seguida, incrementa-se o semáforo *playersWaitReferee* para cada elemento de cada equipa, ou seja, 10 vezes, libertando todos os *players* e *goalies* que estavam bloqueados e indicando a cada *player* e *goalie* de cada equipa que o jogo pode começar.

```c
static void startGame ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = STARTING_GAME;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    for (int p = 0; p < 10; p++) {   // por cada player colocamos o semaforo Up
        if (semUp (semgid, sh->playersWaitReferee) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

Função *play*()

Apenas atualizamos o estado do árbitro para REFEREEING (3), ou seja, nesse momento, o jogo já está a decorrer.

```c
static void play ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = REFEREEING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    usleep((100.0*random())/(RAND_MAX+1.0)+900.0);
}
```

Função *endGame*()

O tempo de jogo chega ao fim, e o árbitro tem que terminá-lo. Portanto, atualizamos o seu estado para ENDING_GAME (4).

Em relação a semáforos, desbloqueia-se o semáforo *playersWaitEnd* para cada elemento de cada equipa, ou seja, 10 vezes. Estes estavam à espera que o árbitro

acabasse o jogo, estando os seus processos bloqueados pelo semáforo enquanto o jogo decorria, como poderemos ver posteriormente neste relatório.

```
static void endGame ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.refereeStat = ENDING_GAME;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    for (int p = 0; p < 10; p++) {   // por cada player colocamos o semaforo Up
        if (semUp (semgid, sh->playersWaitEnd) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }

}
```

5.2. Player (SemSharedMemPlayer.c) e Goalie (SemSharedMemGoalie.c)

Decidimos juntar a explicação de cada um destes scripts nesta secção, uma vez que são bastante semelhantes.

Função *arrive*()

Esta função define o estado inicial dos *players* e dos *goalies* como ARRIVING. O estado deles estará então com o valor 0, o que significa que ainda estão a chegar.

Para os *players*:

```
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.playerStat[id] = ARRIVING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    usleep((200.0*random())/(RAND_MAX+1.0)+50.0);
}
```

Para os *goalies*:

```
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.goalieStat[id] = ARRIVING;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    usleep((200.0*random())/(RAND_MAX+1.0)+60.0);
}
```

Função *playerConstituteTeam*()

Esta função tem como objetivo associar uma equipa a cada *player*, seja ou à equipa de ID 1 ou à equipa de ID 2.

Quando um *player* chega, tem que lhe ser associado uma equipa pela qual vai jogar. Primeiramente, o número de *players* que chegaram e o número de *players* livres, isto é, ainda sem equipa, aumentam. Por isso, é incrementada uma unidade às variáveis da estrutura FULL_STAT *playerArrived* e *playersFree*, uma vez que chegou mais um jogador, jogador este que não tem equipa e está livre.

Se já tiverem chegado 8 *players*, significa que o jogador que chegou está atrasado, portanto ser-lhe-á atribuído o estado LATE (7). Caso contrário, verificamos se o *player* que acabou de chegar é ou não o capitão de equipa.

O capitão de equipa é definido como o quinto elemento de uma equipa a chegar, por exemplo, se já chegaram 3 *players* e 1 *goalie* então o próximo *player* a chegar será o capitão da equipa. Este terá como objetivo criar a equipa e bloquear e desbloquear alguns semáforos.

Se o número de *playersFree* e o número de *goaliesFree* ao nosso dispor for maior ou igual ao número necessário para formar equipa, 4 e 1 respetivamente, então o *player* em questão é o capitão de equipa. Caso contrário, o *player* ficará à espera de equipa, atualizando o seu estado para WAITING_TEAM (3).

Para um *player* capitão, são realizados os seguintes passos:

O estado desse *player* passará a ser FORMING_TEAM (2), pois será o capitão que vai formar a equipa. Decrementamos quatro unidades à variável *playersFree* e uma unidade à variável *goaliesFree*, porque, ao formar uma equipa, quatro *players* e um *goalie* deixaram de estar livres.

Sendo constituída a equipa, dentro do *mutex*, o semáforo *playersWaitTeam* irá desbloquear 3 vezes, uma para cada jogador excluíndo o capitão, pois o seu semáforo não está bloqueado. O semáforo *goaliesWaitTeam* irá, também, ser desbloqueado para o *goalie* pertencer a esta equipa.

Para registar os jogares na equipa, bloqueamos quatro vezes o semáforo *playerRegistered*, um para cada elemento, exceto o capitão dado que é ele que forma a equipa.

Com a equipa já constituída, atribui-se o valor da variável *teamID* da estrutura *FULL_STAT* à variável de retorno *ret*, e incrementa-se o *teamID* para uma futura criação da segunda equipa. A variável *ret* será, assim, o ID da equipa.

No final, depois de sair da região crítica, vamos bloquear e desbloquear semáforos dependendo do *player* em questão.

Se o *player* estiver no estado WAITING_TEAM, o semáforo *playersWaitTeam* será decrementado. Este só será elevado quando o *player* capitão chegar e realizar os passos

acima descritos. O valor de retorno guarda o *teamID* ao qual esse *player* pertecerá. O semáforo *playerRegistered* vai ser incrementado.

Caso o *player* estiver no estado FORMING_TEAM, ou seja, se for o capitão, o semáforo *refereeWaitTeams* irá ser desbloqueado, porque a equipa já está formada pelo capitão, então “dão” essa informação ao árbitro.

```
static int playerConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1)  {                                                    /* enter
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.playersArrived++;
    sh->fSt.playersFree++;

    if (sh->fSt.playersArrived <= 8){ //
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS && sh->fSt.goaliesFree >= NUMTEAMGOALIES){

            // código para o player capitão

            sh->fSt.st.playerStat[id] = FORMING_TEAM;

            sh->fSt.playersFree -= NUMTEAMPLAYERS;
            sh->fSt.goaliesFree -= NUMTEAMGOALIES;

            // SEMÁFOROS
            // colocamos o semaforo Up para os restantes jogadores(3) (sem contar com o capitão)

            for (int p = 0; p < 3; p++){
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (PL)");
                    exit (EXIT_FAILURE);
                }
            }

            // colocamos o semaforo Up porque o guarda-redes junta-se a esta equipa

            if (semUp (semgid, sh->goaliesWaitTeam) == -1) {
                perror ("error on the up operation for semaphore access (GL)");
                exit (EXIT_FAILURE);
            }

            // colocamos o semaforo down para registar os jogadores na equipa
            // (sem contar com capitão pois ele forma a equipa)
```

```
            for (int p = 0; p < 4; p++){
                if (semDown (semgid, sh->playerRegistered) == -1) {
                    perror ("error on the down operation for semaphore access (PL)");
                    exit (EXIT_FAILURE);
                }
            }
            // FIM SEMÁFOROS

            ret = sh->fSt.teamId;
            sh->fSt.teamId++;
            saveState(nFic, &sh->fSt); // guardamos o estado do capitão como forming team

        } else {

            // código para um player não capitão

            sh->fSt.st.playerStat[id] = WAITING_TEAM;
            saveState(nFic, &sh->fSt);

        }
    } else {

        // código para um player que chega atrasado

        sh->fSt.st.playerStat[id] = LATE;
        saveState(nFic, &sh->fSt);
        sh->fSt.playersFree -= 1;

    }

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
```

```
    if (sh->fSt.st.playerStat[id] == WAITING_TEAM){

        // semaféros para um player não capitão

        if (semDown (semgid, sh->playersWaitTeam) == -1) {
            perror ("error on the down operation for semaphore access (PL)");
            exit (EXIT_FAILURE);
        }

        ret = sh->fSt.teamId; // guardar o ID da equipa como valor de retorno

        if (semUp (semgid, sh->playerRegistered) == -1) {
            perror ("error on the up operation for semaphore access (PL)");
            exit (EXIT_FAILURE);
        }

    } else if (sh->fSt.st.playerStat[id] == FORMING_TEAM){

        // semáforo para o player capitão

        if (semUp (semgid, sh->refereeWaitTeams) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }

    }
    return ret;
}
```

Função *goalieConstituteTeam*()

Esta função usa a mesma filosofia que a função *playerConstituteTeam*(), tendo algumas exceções por se tratar de um *goalie*. Nesta parte trataremos só de explicar as exceções, a explicação do resto do código está na função acima.

Se o *goalie* for o último elemento da equipa a chegar, então será o capitão. Ao contrário do *playerConstituteTeam* os semáforos que funcionam dentro do *mutex* serão

o *playersWaitTeam* e o *playersRegistered* repetidos quatro vezes, uma para cada *player*, ficando o primeiro desbloqueado, uma vez que os *players* ao formarem uma equipa deixam de esperar por ela, e o segundo bloqueado.

```
static int goalieConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }
    sh->fSt.goaliesArrived++;
    sh->fSt.goaliesFree++;

    if (sh->fSt.goaliesArrived <= 2){
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS){

            // código se o goalie for capitão

            sh->fSt.st.goalieStat[id] = FORMING_TEAM;

            sh->fSt.playersFree -= NUMTEAMPLAYERS;
            sh->fSt.goaliesFree -= NUMTEAMGOALIES;

            // SEMÁFOROS
            // colocamos 4 vezes o semaфóro down para todos os jogadores da equipa (4)

            for (int p = 0; p < 4; p++){
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }

            // colocamos o semaforo down para registar os jogadores na equipa
            // (sem contar com capitão pois ele forma a equipa)

            for (int p = 0; p < 4; p++){
                if (semDown (semgid, sh->playerRegistered) == -1) {
                    perror ("error on the down operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }

            // FIM SEMÁFOROS

            ret = sh->fSt.teamId;
            sh->fSt.teamId++;
            saveState(nFic, &sh->fSt);

        } else {

            // código caso o golie não é capitão

            sh->fSt.st.goalieStat[id] = WAITING_TEAM;
            saveState(nFic, &sh->fSt);

        }
    } else {
```

```
        // código caso o goalie chegar atrasado

        sh->fSt.st.goalieStat[id] = LATE;
        saveState(nFic, &sh->fSt);
        sh->fSt.goaliesFree -= 1;

    }

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (sh->fSt.st.goalieStat[id] == WAITING_TEAM){

        // semaforos se o goalie não for capitão

        if (semDown (semgid, sh->goaliesWaitTeam) == -1) {
            perror ("error on the down operation for semaphore access (GL)");
            exit (EXIT_FAILURE);
        }

        ret = sh->fSt.teamId; // guardar o ID da equipa como valor de retorno

        if (semUp (semgid, sh->playerRegistered) == -1) {
            perror ("error on the up operation for semaphore access (GL)");
            exit (EXIT_FAILURE);
        }

    } else if (sh->fSt.st.goalieStat[id] == FORMING_TEAM){

        // semaforos caso for capitão

        if (semUp (semgid, sh->refereeWaitTeams) == -1) {
            perror ("error on the up operation for semaphore access (GL)");
            exit (EXIT_FAILURE);
        }

    }
    return ret;
}
```

Função *waitReferee*()

Nesta função, as equipas estão à espera que o árbitro comece o jogo.

Portanto, atualizamos o estado de cada *player* e *goalie* dependendo da sua equipa. Se pertencer á equipa do ID 1, então passará para o estado WAITING_START_1 (3). No caso contrário, ou seja, se o ID for 2, passará para o estado WAITING_START_2 (4).

O semáforo *playersWaitReferee* será decrementado, uma vez que os *players* ficarão à espera que o árbitro dê início ao jogo. Este semáforo só irá ser incrementado e desbloquear o processo na função *startGame*() no script do árbitro.

Para o *player*:

```c
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    sh->fSt.st.playerStat[id] = (team == 1) ? WAITING_START_1 : WAITING_START_2;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */

    if (semDown (semgid, sh->playersWaitReferee) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

}
```

Para o *goalie*:

```c
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.playerStat[id] = (team == 1) ? WAITING_START_1 : WAITING_START_2;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitReferee) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

}
```

Função *playUntilEnd*()

Nesta função, os *players* já se encontram durante o jogo. Portanto, atualizamos o estado do *player* para PLAYING_1, caso pertencer à equipa com ID 1, e para PLAYING_2, caso pertencer à outra.

Em relação aos semáforos, os *players* estão á espera que o árbitro termine o jogo, então o semáforo *playerWaitEnd* vai ser decrementado e o processo ficará bloqueado até que o próprio árbitro o desbloqueie na função *endGame*().

Para os *players*:

```
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.playerStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitEnd) == -1) {
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

}
```

Para os *goalies*:

```
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    sh->fSt.st.goalieStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;
    saveState(nFic, &sh->fSt);

    if (semUp (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

    /* TODO: insert your code here */
    if (semDown (semgid, sh->playersWaitEnd) == -1) {
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

}
```

6. Resultados obtidos

```
                    SoccerGame - Description of the internal state

P00 P01 P02 P03 P04 P05 P06 P07 P08 P09  G00 G01 G02  R01
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  0   0   0   0   0   0   0   0   0   0    0   0   0    0
  1   0   0   0   0   0   0   0   0   0    0   0   0    0
  1   0   0   0   1   0   0   0   0   0    0   0   0    0
  1   0   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   0   0   0   0   0    0   0   0    0
  1   1   0   1   1   1   0   0   0   0    0   0   0    0
  1   1   0   1   1   1   0   0   0   0    0   0   0    0
  1   1   1   1   1   1   0   0   0   0    0   0   0    0
  1   1   1   1   1   1   0   0   0   0    2   0   0    0
  1   1   1   1   1   1   0   0   0   0    2   0   0    0
  1   1   1   1   1   1   0   0   0   0    2   1   0    0
  1   1   1   1   1   1   0   0   0   0    2   1   7    0
  1   1   1   1   1   1   0   1   0   0    2   1   7    0
  1   1   1   1   1   1   0   1   0   2    2   1   7    0
  1   1   1   1   1   1   7   1   0   2    2   1   7    0
  3   1   1   1   1   1   7   1   0   2    2   1   7    0
  3   1   1   3   1   1   7   1   0   2    2   1   7    0
  3   3   1   3   1   1   7   1   0   2    2   1   7    0
  3   3   1   3   3   1   7   1   0   2    2   1   7    0
  3   3   1   3   3   1   7   1   0   2    3   1   7    0
  3   3   1   3   3   1   7   1   7   2    3   1   7    0
  3   3   1   3   3   1   7   1   7   2    3   1   7    1
  3   3   1   3   3   4   7   1   7   2    3   1   7    1
  3   3   4   3   3   4   7   1   7   2    3   1   7    1
  3   3   4   3   3   4   7   4   7   2    3   1   7    1
  3   3   4   3   3   4   7   4   7   2    3   4   7    1
  3   3   4   3   3   4   7   4   7   4    3   4   7    1
  3   3   4   3   3   4   7   4   7   4    3   4   7    2
  5   3   4   3   3   4   7   4   7   4    3   4   7    2
  5   3   4   5   3   4   7   4   7   4    3   4   7    2
  5   5   4   5   3   4   7   4   7   4    3   4   7    2
  5   5   4   5   5   4   7   4   7   4    3   4   7    2
  5   5   4   5   5   4   7   4   7   4    5   4   7    2
  5   5   4   5   5   6   7   4   7   4    5   4   7    2
  5   5   6   5   5   6   7   4   7   4    5   4   7    2
  5   5   6   5   5   6   7   4   7   4    5   6   7    2
  5   5   6   5   5   6   7   4   7   4    5   6   7    3
  5   5   6   5   5   6   7   4   7   6    5   6   7    3
  5   5   6   5   5   6   7   6   7   6    5   6   7    3
  5   5   6   5   5   6   7   6   7   6    5   6   7    4
```

7. Conclusão

A realização deste projeto fomentou o nosso conhecimento teórico e prático sobre os semáforos e permitiu-nos perceber a sua importância na sincronização de vários processos, uma vez que este bloqueia e desbloqueia processos para garantir de que o programa corre conforme o esperado e para assegurar que cada processo tenha todos os recursos que necessita.