Sistemas Operativos

Relatório Trabalho 02– Turma P02

Prof. José Nuno Panelas Nunes Lau

Janeiro 2021

Paulo Pereira (98430) (50 %)

Alexandre Serras (97505) (50 %)

Índice

Introdução

O objetivo deste trabalho era o desenvolvimento de um programa em C capaz de simular um encontro de amigos para uma futebolada.

Para isso o programa considera processos independentes, o árbitro, os jogadores de campo e os guarda-redes. É também necessária a sincronização e comunicação entre os processos, isto será alcançado através do uso de semáforos e de memória partilhada, além disso cada processo deverá estar ativo apenas quando necessário, estando bloqueado enquanto espera por um determinado evento.

Existem duas equipas de 4 jogadores, um guarda-redes e ainda um árbitro. O árbitro terá a função de iniciar e concluir o jogo, após receber a confirmação que ambas as equipas se encontram prontas para tal. Os jogadores e guarda-redes terão que após a sua chegada tentar formar equipa caso isso não seja possível deverão esperar ser chamados para uma das equipas. Temos de considerar ainda o caso de os jogadores (ou guarda-redes) chegarem atrasados, caso em que não participam no jogo. Ao longo do documento quando referirmos o jogador no estado FORMING_TEAM iremos utilizar capitão.

Ao longo do relatório é apresentada a nossa metodologia para solucionar o problema apresentado.

Os semáforos

Neste capítulo iremos procurar explicar a função de cada semáforo, além disso teremos também uma tabela onde é exposto o seu comportamento ao longo do programa.

Semáforo	Entidade Down	Função Down	# Downs	Entidade UP	Função Up	#Ups
playersWaitTeam	Player	playerConstituteTeam	1	Caso1 goalie Caso2 player	Caso 1 goalieConstituteTeam Caso 2 playerConstituteTeam	Caso 1: 4 Caso 2: 3
goaliesWaitTeam	Goalie	goalieConstituteTeam	1	player	playerconstitueteam	Caso 1: 1
playersWaitReferee	Player/goalie	waitReferee	1	referee	startGame	10
playersWaitEnd	Player/goalie	playUntilEnd	1	referee	endGame	10
refereeWaitTeams	Referee	waitForTeams	2	Caso 1 goalie Caso 2 player	Caso 1 goalieConstituteTeam Caso 2 playerConstituteTeam	1
playerRegistered	Player/goalie	Caso 1 GoalieConstituteTeam Caso 2 PlayerConstituteteam	Caso 1: 4 Caso 2: 4	Caso 1: goalie Caso 2: player	Caso 1 goalieConstituteTeam Caso 2 playerconstitueteam	Caso 1: 1 Caso 2: 1

playersWaitTeam- Semáforo utilizado pelos jogadores de campo no estado WAITING_TEAM enquanto esperam pelo capitão (jogador que estará no estado FORMING_TEAM).

goaliesWaitTeam- Semáforo utilizado pelos guarda-redes no estado WAITING_TEAM enquanto esperam pelo capitão (jogador que estará no estado FORMING_TEAM).

playersWaitReferee- Semáforo utilizado pelos jogadores enquanto esperam que o árbitro inicie a partida.
playersWaitEnd- Semáforo utilizado pelos jogadores enquanto esperam que o árbitro conclua a partida.

refereeWaitTeams- Semáforo utilizado pelo árbitro enquanto espera que as equipas estejam formadas.

playerRegistered- Semáforo utilizado pelos jogadores para comunicarem ao capitão que reconhecem que pertencem à sua equipa.

Descrição das funções

Nesta seção do relatório vamos descrever cada função de cada entidade que intervém na futebolada.

Referee

```
/**
 *  \brief referee takes some time to arrive
 *
 *  Referee updates state and takes some time to arrive
 *  The internal state should be saved.
 *
 */
static void arrive ()
{
    if (semDown (semgid, sh->mutex) == -1) {                    /* enter critical region */
        perror ("error on the down 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) {                      /* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

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

}
```

A função de arrive do árbitro, é uma função muito simples e apenas se alterou-se o estado do árbitro para arrive e guardou-se o estado, nota que a função saveState, imprime o estado no terminal.

```c
/**
 *  \brief referee waits for teams to be formed
 *
 *  Referee updates state and waits for the 2 teams to be completely formed
 *  The internal state should be saved.
 *
 */
static void waitForTeams ()
{
    if (semDown (semgid, sh->mutex) == -1) {                                    /* enter critical region */
        perror ("error on the down 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 */
    for (int i=0;i<2 ;i++){
    if (semDown(semgid,sh->refereeWaitTeams) == -1) {
        perror ("error on the down operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }
    }
}
```

A função waitTeams, primeiro o árbitro entra no mutex, altera o seu estado para Waiting_Teams, imprime no terminal e sai do mutex.

Depois fica bloqueado até que as 2 equipas não lhe digam que já estão prontas.

```
/**
 *  \brief referee starts game
 *
 *  Referee updates state and notifies players and goalies to start match
 *  The internal state should be saved.
 *
 */
static void startGame ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

    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 i=0;i<10 ;i++){
        if (semUp(semgid,sh->playersWaitReferee) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

Nesta função o árbitro , altera o seu estado para Starting_Game e imprime no terminal o seu novo estado.

Depois de sair do mutex, o árbitro informa os 10 jogadores que está pronto para começar o jogo.

```
/**
 *  \brief referee takes some time to allow game to finish
 *
 *  Referee updates state and takes some time to finish the game
 *  The internal state should be saved.
 *
 */
static void play ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }

      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);
}
```

Na função play(), o referee entra no mutex, altera o seu estado REFEREEING e sai do mutex.

```
/**
 *  \brief referee ends game
 *
 *  Referee updates state and notifies players and goalies to end match
 *  The internal state should be saved.
 *
 */
static void endGame ()
{
    if (semDown (semgid, sh->mutex) == -1) {
        perror ("error on the down 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);
    }

    for (int i=0;i<10 ;i++){
        if (semUp(semgid,sh->playersWaitEnd) == -1) {
            perror ("error on the up operation for semaphore access (RF)");
            exit (EXIT_FAILURE);
        }
    }

}
```

Quando o referee chega a esta função está na hora de acabar a futebolada, ele entra no mutex, altera o seu estado para o ENDING_GAME e sai do mutex.

Antes de sair da função o árbitro, todos os jogadores,incluindo os goalies, que acabou a partida.

Goalies

```c
/**
 *  \brief goalie takes some time to arrive
 *
 *  Goalie updates state and takes some time to arrive
 *  The internal state should be saved.
 *
 */
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                   /* enter critical region */
        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) {                                      /* exit critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

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

Função arrive do goalies faz exatamente o mesmo que a função do referee, ou seja, apenas serve para alterar o estado do goalie para arrived e imprimir no terminal.

Nota a ter em conta, para o goalie como existem mais que um, vamos ter que atualizar na sua posição logo temos que fazer que é o sh->fSt.st.goalieStat[id], e assim atualizamos o estado do goalie correto, ou seja, do goalie que está no mutex naquele momento.

```
/**
 *  \brief goalie constitutes team
 *
 *  If goalie is late, it updates state and leaves.
 *  If there are enough free players to form a team, goalie forms team allowing team members to
 *  proceed and waiting for them to acknowledge registration.
 *  Otherwise it updates state, waits for the forming teammate to "call" him, saves its team
 *  and acknowledges registration.
 *  The internal state should be saved.
 *
 *  \param id goalie id
 *
 *  \return id of goalie team (0 for late goalies; 1 for team 1; 2 for team 2)
 *
 */
static int goalieConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }
    sh->fSt.goaliesArrived++;
    /* TODO: insert your code here */
        if (sh->fSt.playersFree > 3){
            sh->fSt.playersFree =  sh->fSt.playersFree -4;
            sh->fSt.st.goalieStat[id]=FORMING_TEAM;
            saveState(nFic,&sh->fSt);
            for (int i=0;i<4; i++){
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }
            for (int i=0;i<4; i++){
                if (semDown (semgid, sh->playerRegistered) == -1) {
                    perror ("error on the down operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }
            ret=sh->fSt.teamId;
            if (sh->fSt.teamId == 1){
                sh->fSt.teamId=sh->fSt.teamId+1;
            }
        }
         else if ( sh->fSt.goaliesArrived <= 2  && sh->fSt.playersFree <= 3){
            sh->fSt.goaliesFree= sh->fSt.goaliesFree+1;
            sh->fSt.st.goalieStat[id]=WAITING_TEAM;
            saveState(nFic,&sh->fSt);
        }
         else{
        sh->fSt.goaliesFree= sh->fSt.goaliesFree+1;
        sh->fSt.st.goalieStat[id]=LATE;
         saveState(nFic,&sh->fSt);
    }

    if (semUp (semgid, sh->mutex) == -1) {                                             /* exit critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }
    if (sh->fSt.st.goalieStat[id] == WAITING_TEAM){

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

    }
     if (sh->fSt.st.goalieStat[id] == FORMING_TEAM){
     if (semUp (semgid, sh->refereeWaitTeams) == -1) {                                                 /* exit critical region
                perror ("error on the up operation for semaphore access (GL)");
                exit (EXIT_FAILURE);
            }
    }
    return ret;
}
```

Esta função é onde existem mais pormenores a ter em conta, logo, vamos analisá-la caso a caso, 1º caso: goalie tem que formar a equipa, 2º caso: goalie fica à espera de equipa e o 3º caso : o goalie chegou atrasado e já não pode jogar por nenhuma equipa.

Analisando a função para o 1º caso:

Entrar no mutex

O goalie verifica que existem pelo 4 jogadores livres e aí o goalie sabe que tem que formar a equipa e decrementa em 4, a variável playersFree e altera o seu estado para FORMING_TEAMS, dá up aos players que vão constituir a sua equipa e fica bloqueado, dentro do mutex, no semáforo playerRegistered ficando à espera que os jogadores confirmem que sabem que vão pertencer à equipa formada por ele, quando os 4 jogadores confirmarem, ele verifica a que equipa pertence, e caso ele pertença à equipa 1, ele atualiza a variável teamId para 2.

Sair do mutex.

E informa o árbitro que a sua equipa já está formada.

Analisando a função para o 2º caso:

Entrar no mutex

Para o goalie executar este caso, significa que não chegaram 2 goalies primeiro que ele e que não estão 4 jogadores livres.

Nesta situação o goalie precisa apenas de informar que ele está livre e alterar o seu estado para WAITING_TEAM.

Sair do mutex

Dar um down no semáforo do goaliesWaitTeam, depois quando for levantado este semáforo, o goalie/goalie verifica a que equipa pertence guardando na variável ret e informa o jogador que está a formar a equipa que tomou conhecimento que pertence a essa equipa dando up do semáforo playerRegistered.

Analisando a função para o 3º caso:

O goalie verifica que já chegaram 2 goalies, ou seja, ele já não tem espaço em nenhuma das equipas e altera o seu estado para LATE e imprime no terminal.

```
/**
 *  \brief goalie waits for referee to start match
 *
 *  The goalie updates its state and waits for referee to start match.
 *  The internal state should be saved.
 *
 *  \param id   goalie id
 *  \param team goalie team
 */
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }    /* TODO: insert your code here */
    if (team ==1 ){
        sh->fSt.st.goalieStat[id]= WAITING_START_1;
         saveState(nFic,&sh->fSt);
    }
    else{
        sh->fSt.st.goalieStat[id]= WAITING_START_2;
         saveState(nFic,&sh->fSt);
    }
    if (semUp (semgid, sh->mutex) == -1) {                                          /* exit critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

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

Nesta função o goalie entra no mutex, verifica através da variável team, de que equipa é e consoante a equipa que ele pertença coloca-se no estado Waiting_Start_1 ou 2, imprime no terminal e sai do mutex.

E fica preso no semáforo playersWaitReferee à espera que o árbitro comece o jogo.

```
/**
 *  \brief goalie waits for referee to end match
 *
 *  The goalie updates its state and waits for referee to end match.
 *  The internal state should be saved.
 *
 *  \param id   goalie id
 *  \param team goalie team
 */
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

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

    }
    else{
        sh->fSt.st.goalieStat[id]= PLAYING_2;
            saveState(nFic,&sh->fSt);

    }

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

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

O goalie entra no mutex, verifica a que equipa pertence e consoante a equipa a que ele pertence e coloca-se a jogar por essa mesma equipa, PLAYING_1 ou 2 e sai do mutex.

Depois fica preso no semáforo, playersWaitEnd, à espera que o árbitro dê o jogo por terminado, após isto, o goalie termina.

Player

```
/**
 *  \brief player takes some time to arrive
 *
 *  Player updates state and takes some time to arrive
 *  The internal state should be saved.
 *
 */
static void arrive(int id)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                   /* enter critical region */
        perror ("error on the down 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) {                                      /* exit critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }

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

Função arrive do player faz exatamente o mesmo que a função do referee e do goalie, ou seja, apenas serve para alterar o estado do player para arrived e imprimir no terminal.

Nota a ter em conta, para o player como existem mais que um, vamos ter que atualizar na sua posição logo temos que fazer que é o sh->fSt.st.playerStat[id], e assim atualizamos o estado do player correto, ou seja, do player que está no mutex naquele momento.

```
/**
 *  \brief player constitutes team
 *
 *  If player is late, it updates state and leaves.
 *  If there are enough free players and free goalies to form a team, player forms team allowing
 *  team members to proceed and waiting for them to acknowledge registration.
 *  Otherwise it updates state, waits for the forming teammate to "call" him, saves its team
 *  and acknowledges registration.
 *  The internal state should be saved.
 *
 *  \param id player id
 *
 *  \return id of player team (0 for late goalies; 1 for team 1; 2 for team 2)
 *
 */
static int playerConstituteTeam (int id)
{
    int ret = 0;

    if (semDown (semgid, sh->mutex) == -1)  {                                                   /* enter critical region */
        perror ("error on the down operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }
    sh->fSt.playersArrived++;
         if (sh->fSt.goaliesFree >=1  && sh->fSt.playersFree >=3 ){
            sh->fSt.playersFree =  sh->fSt.playersFree -3;
            sh->fSt.goaliesFree =  sh->fSt.goaliesFree -1;
            sh->fSt.st.playerStat[id]=FORMING_TEAM;
            saveState(nFic,&sh->fSt);
            for (int i=0;i<3; i++){
                if (semUp (semgid, sh->playersWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }
            if (semUp (semgid, sh->goaliesWaitTeam) == -1) {
                    perror ("error on the up operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            for (int i=0;i<4; i++){
                if (semDown (semgid, sh->playerRegistered) == -1) {
                    perror ("error on the down operation for semaphore access (GL)");
                    exit (EXIT_FAILURE);
                }
            }
            ret=sh->fSt.teamId;
            if (sh->fSt.teamId == 1){
                sh->fSt.teamId=sh->fSt.teamId+1;
            }
        }
         else if (sh->fSt.playersArrived <= 8){
            sh->fSt.playersFree= sh->fSt.playersFree+1;
            sh->fSt.st.playerStat[id]=WAITING_TEAM;
            saveState(nFic,&sh->fSt);
        }
         else{
        sh->fSt.st.playerStat[id]=LATE;
         saveState(nFic,&sh->fSt);
    }

    /* TODO: insert your code here */
    if (semUp (semgid, sh->mutex) == -1) {                                                      /* exit critical region */
        perror ("error on the up operation for semaphore access (PL)");
        exit (EXIT_FAILURE);
    }
       if (sh->fSt.st.playerStat[id] == WAITING_TEAM){
        if (semDown (semgid, sh->playersWaitTeam) == -1) {
            perror ("error on the down operation for semaphore access (GL)");
            exit (EXIT_FAILURE);
        }
        ret=sh->fSt.teamId;
        if (semUp (semgid, sh->playerRegistered) == -1) {                                                   /* exit critical region */
            perror ("error on the up operation for semaphore access (GL)");
            exit (EXIT_FAILURE);
        }
    }

    /* TODO: insert your code here */
    if (sh->fSt.st.playerStat[id] == FORMING_TEAM){
     if (semUp (semgid, sh->refereeWaitTeams) == -1) {                                                  /* exit critical region */
                perror ("error on the up operation for semaphore access (GL)");
                exit (EXIT_FAILURE);
            }
    }
    return ret;
}
```

Esta função é onde existem mais pormenores a ter em conta, logo, vamos analisá-la caso a caso, 1º caso: player tem que formar a equipa, 2º caso: player fica à espera de equipa ,e o 3º caso : o player chegou atrasado e já não pode jogar por nenhuma equipa.

Analisando a função para o 1º caso:

Entrar no mutex

O player verifica que existem pelo 3 jogadores livres e 1 goalie, e aí o player sabe que tem que formar a equipa e decrementa em 3, a variável playersFree e em 1 o goaliesFree, e altera o seu estado para FORMING_TEAMS.

O player dá up dos semáforos, playerWaitTeam e goalieWaitTeam, dos 3 jogadores que pertence à equipa dele e ao goalie que pertence

Fica bloqueado, dentro do mutex, no semáforo playerRegistered ficando à espera que os jogadores da sua equipa confirmem que sabem que vão pertencer à equipa formada por ele, quando os 4 jogadores confirmarem, ele verifica a que equipa pertence, e caso ele pertença à equipa 1, ele atualiza a variável teamId para 2.

Sair do mutex.

E informa o árbitro que a sua equipa já está formada.

Analisando a função para o 2º caso:

Entrar no mutex

Para o player executar este caso, significa que não chegaram 8 players primeiro que ele logo ele vai jogar mas não sabe porque equipa.

Nesta situação o player precisa apenas de informar que ele está livre e alterar o seu estado para WAITING_TEAM.

Sair do mutex

Dar um down no semáforo do playerWaitTeam, depois quando for levantado este semáforo, o player verifica a que equipa pertence guardando na variável ret e informa o jogador/goalie que está a formar a equipa que tomou conhecimento que pertence a essa equipa dando up do semáforo playerRegistered.

Analisando a função para o 3º caso:

O player verifica que já chegaram 8 players, ou seja, ele já não tem espaço em nenhuma das equipas e altera o seu estado para LATE e imprime no terminal.

```c
/**
 *  \brief player waits for referee to start match
 *
 *  The player updates its state and waits for referee to end match.
 *  The internal state should be saved.
 *
 *  \param id   player id
 *  \param team player team
 */
static void waitReferee (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                          /* enter critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }    /* TODO: insert your code here */
    if (team ==1 ){
        sh->fSt.st.playerStat[id]= WAITING_START_1;
         saveState(nFic,&sh->fSt);
    }
    else{
        sh->fSt.st.playerStat[id]= WAITING_START_2;
         saveState(nFic,&sh->fSt);
    }
    if (semUp (semgid, sh->mutex) == -1) {                                          /* exit critical region */
        perror ("error on the up operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

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

Nesta função o player entra no mutex, verifica através da variável team, de que equipa é e consoante a equipa que ele pertença coloca-se no estado Waiting_Start_1 ou 2, imprime no terminal e sai do mutex.

E fica preso no semáforo playersWaitReferee à espera que o árbitro comece o jogo.

```
/**
 *  \brief player waits for referee to end match
 *
 *  The player updates its state and waits for referee to end match.
 *  The internal state should be saved.
 *
 *  \param id   player id
 *  \param team player team
 */
static void playUntilEnd (int id, int team)
{
    if (semDown (semgid, sh->mutex) == -1)  {                                                    /* enter critical region */
        perror ("error on the down operation for semaphore access (GL)");
        exit (EXIT_FAILURE);
    }

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

    }
    else{
        sh->fSt.st.playerStat[id]= PLAYING_2;
            saveState(nFic,&sh->fSt);
    }

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

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

}
```

O player entra no mutex, verifica a que equipa pertence e consoante a equipa a que ele pertence e coloca-se a jogar por essa mesma equipa, PLAYING_1 ou 2 e sai do mutex.

Depois fica preso no semáforo, playersWaitEnd, à espera que o árbitro dê o jogo por terminado, após isto, o player termina.

Testes

Para verificar a coerência dos nossos resultados, testamos o nosso código entidade por entidade com os binários dados pelo professor,isto é o nosso referee com os binários dos jogador e do goalie e por aí em diante, e apenas quando todos corriam 1000 vezes seguidas sem nenhum deadlock, passamos para a próxima entidade.

No final quando já funcionavam as 3 entidades em separado, fomos tentar juntá-las, através do make all, e como era esperado corremos 1000 vezes e como esperado não houve nenhum deadlock.

As figuras seguintes que vamos inserir no relatório, são os casos que achamos mais pertinentes e que necessitavam de ser estudados ao pormenor, são eles, 2 goalies a formar a equipa, 2 players a formar a equipa e 1 player e 1 goalie a formar a equipa.

Seguidamente, verificamos se os resultados obtidos tinham lógica face ao objetivo do projeto, se de facto as entidades funcionam como é pretendido. Este teste foi realizado manualmente, verificando linha a linha se as atualizações e mudanças de estado fazem sentido, isto foi feito para vários casos individuais, sendo que aqui demonstramos apenas 3 desses casos, aqueles que achamos mais benéficos.

Caso em que 2 goalies a formam as equipas

```
Run n.º 999
                    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
  1   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   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   1   1   0   0   0   0   0   0   0    0   0   0    0
  1   1   1   0   0   0   0   0   0   0    0   0   0    0
  1   1   1   0   0   0   1   0   0   0    0   0   0    0
  1   1   1   0   0   0   1   0   0   0    0   0   0    0
  1   1   1   0   0   1   1   0   0   0    0   0   0    0
  1   1   1   0   0   1   1   0   0   0    0   0   0    0
  1   1   1   1   0   1   1   0   0   0    0   0   0    0
  1   1   1   1   0   1   1   1   0   0    0   0   0    0
  1   1   1   1   0   1   1   1   0   0    0   0   0    0
  1   1   1   1   0   1   1   1   0   1    0   0   0    0
  1   1   1   1   0   1   1   1   0   1    0   0   0    0
  1   1   1   1   0   1   1   1   0   1    0   0   0    0
  1   1   1   1   0   1   1   1   0   1    0   0   0    1
  3   1   1   1   0   1   1   1   0   1    2   0   0    1
  3   1   3   1   0   1   1   1   0   1    2   0   0    1
  3   3   3   1   0   1   1   1   0   1    2   0   0    1
  3   3   3   1   0   1   3   1   0   1    2   0   0    1
  3   3   3   1   0   1   3   1   0   1    3   0   0    1
  3   3   3   1   0   1   3   1   0   1    3   2   0    1
  3   3   3   1   0   4   3   1   0   1    3   2   0    1
  3   3   3   4   0   4   3   1   0   1    3   2   0    1
  3   3   3   4   0   4   3   4   0   1    3   2   0    1
  3   3   3   4   0   4   3   4   0   4    3   2   0    1
  3   3   3   4   0   4   3   4   0   4    3   4   0    1
  3   3   3   4   0   4   3   4   0   4    3   4   0    2
  5   3   3   4   0   4   3   4   0   4    3   4   0    2
  5   3   5   4   0   4   3   4   0   4    3   4   0    2
  5   3   5   4   0   4   5   4   0   4    3   4   0    2
  5   5   5   4   0   4   5   4   0   4    3   4   0    2
  5   5   5   4   0   6   5   4   0   4    3   4   0    2
  5   5   5   6   0   6   5   4   0   4    3   4   0    2
  5   5   5   6   0   6   5   6   0   4    3   4   0    2
  5   5   5   6   0   6   5   6   0   4    3   6   0    2
  5   5   5   6   0   6   5   6   0   4    3   6   7    2
  5   5   5   6   0   6   5   6   0   4    3   6   7    3
  5   5   5   6   0   6   5   6   0   4    5   6   7    3
  5   5   5   6   0   6   5   6   0   6    5   6   7    3
  5   5   5   6   7   6   5   6   0   6    5   6   7    3
  5   5   5   6   7   6   5   6   0   6    5   6   7    4
  5   5   5   6   7   6   5   6   0   6    5   6   7    4
  5   5   5   6   7   6   5   6   7   6    5   6   7    4
```

universidade de aveiro

Caso em que 2 players formam a equipa

Run n.º 980

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	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	0	2	0	1	0	0
1	1	1	1	0	0	0	0	0	2	0	1	0	0
1	1	1	1	0	0	0	1	0	2	0	1	0	0
1	1	1	1	0	0	0	1	0	2	0	1	1	0
1	1	1	1	0	1	0	1	0	2	0	1	1	0
1	1	1	1	0	1	0	1	0	2	0	1	1	0
1	1	1	1	0	1	0	1	0	2	0	1	1	0
1	1	1	3	0	1	0	1	0	2	0	1	1	0
1	1	1	3	0	1	0	1	0	2	0	1	1	0
3	1	1	3	0	1	0	1	0	2	0	1	1	0
3	1	3	3	0	1	0	1	0	2	0	1	1	0
3	1	3	3	0	1	0	1	0	2	0	3	1	0
3	1	3	3	0	1	0	1	0	3	0	3	1	0
3	1	3	3	0	1	0	1	0	3	7	3	1	0
3	1	3	3	0	1	0	1	0	3	7	3	1	0
3	1	3	3	0	1	0	1	2	3	7	3	1	0
3	1	3	3	0	1	0	1	2	3	7	3	1	1
3	4	3	3	0	1	0	1	2	3	7	3	1	1
3	4	3	3	0	4	0	1	2	3	7	3	1	1
3	4	3	3	0	4	0	1	2	3	7	3	4	1
3	4	3	3	0	4	0	4	2	3	7	3	4	1
3	4	3	3	0	4	0	4	4	3	7	3	4	1
3	4	3	3	0	4	0	4	4	3	7	3	4	2
3	4	3	3	0	4	7	4	4	3	7	3	4	2
3	4	3	3	7	4	7	4	4	3	7	3	4	2
3	4	3	5	7	4	7	4	4	3	7	3	4	2
5	4	3	5	7	4	7	4	4	3	7	3	4	2
5	4	5	5	7	4	7	4	4	3	7	3	4	2
5	4	5	5	7	4	7	4	4	5	7	3	4	2
5	4	5	5	7	6	7	4	4	5	7	3	4	2
5	4	5	5	7	6	7	4	4	5	7	3	4	3
5	4	5	5	7	6	7	4	6	5	7	3	4	3
5	4	5	5	7	6	7	4	6	5	7	3	6	3
5	6	5	5	7	6	7	4	6	5	7	3	6	3
5	6	5	5	7	6	7	4	6	5	7	5	6	3
5	6	5	5	7	6	7	6	6	5	7	5	6	3
5	6	5	5	7	6	7	6	6	5	7	5	6	4

Sistemas Operativos

Caso em que 1 goalie e 1 player formam equipas

```
Run n.º 1000
                    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   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   0   0   0   0   0   0   0    0   0   0    0
  1   0   1   1   0   0   0   0   0   0    0   0   0    0
  1   0   1   1   0   0   0   0   0   0    0   0   0    0
  1   0   1   1   0   0   0   0   0   0    0   0   0    0
  1   0   1   1   0   0   0   0   0   1    0   0   0    0
  1   0   1   1   0   0   0   0   0   1    0   0   0    0
  1   0   1   1   0   0   0   0   0   1    0   0   0    0
  1   0   1   1   0   0   0   0   1   1    0   0   0    0
  1   1   1   1   0   0   0   0   1   1    0   0   0    0
  1   1   1   1   0   0   0   0   1   1    2   0   0    0
  1   1   1   1   0   0   0   1   1   1    2   0   0    0
  1   1   1   1   0   0   0   1   1   1    2   1   0    0
  1   1   1   1   0   0   0   1   1   1    2   1   0    1
  1   1   1   1   0   0   2   1   1   1    2   1   0    1
  1   1   1   1   0   7   2   1   1   1    2   1   0    1
  1   1   3   1   0   7   2   1   1   1    2   1   0    1
  3   1   3   1   0   7   2   1   1   1    2   1   0    1
  3   1   3   1   0   7   2   1   1   3    2   1   0    1
  3   1   3   3   0   7   2   1   1   3    2   1   0    1
  3   1   3   3   0   7   2   1   1   3    3   1   0    1
  3   4   3   3   0   7   2   1   1   3    3   1   0    1
  3   4   3   3   0   7   2   4   1   3    3   1   0    1
  3   4   3   3   0   7   2   4   1   3    3   4   0    1
  3   4   3   3   0   7   2   4   4   3    3   4   0    1
  3   4   3   3   0   7   4   4   4   3    3   4   0    1
  3   4   3   3   7   7   4   4   4   3    3   4   0    1
  3   4   3   3   7   7   4   4   4   3    3   4   0    2
  3   4   3   3   7   7   4   4   4   3    3   4   7    2
  3   4   5   3   7   7   4   4   4   3    3   4   7    2
  5   4   5   3   7   7   4   4   4   3    3   4   7    2
  5   4   5   5   7   7   4   4   4   3    3   4   7    2
  5   4   5   5   7   7   4   4   4   3    5   4   7    2
  5   6   5   5   7   7   4   4   4   3    5   4   7    2
  5   6   5   5   7   7   4   4   4   3    5   4   7    3
  5   6   5   5   7   7   4   6   4   3    5   4   7    3
  5   6   5   5   7   7   6   6   4   3    5   4   7    3
  5   6   5   5   7   7   6   6   4   5    5   4   7    3
  5   6   5   5   7   7   6   6   6   5    5   4   7    3
  5   6   5   5   7   7   6   6   6   5    5   6   7    3
  5   6   5   5   7   7   6   6   6   5    5   6   7    4
```

Conclusão

A implementação do programa probSemSharedMemSoccerGame foi um êxito, pois todos os testes apresentaram os resultados esperados. A base deste trabalho foi uma pesquisa intensa tendo grande parte da informação origem nas aulas teóricas e práticas.