universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Licenciatura em Engenharia Informática
Sistemas Operativos
2020/2021

Trabalho 2 - Simulação de Jogo de Futebol

José Trigo **Nº 98597**
Pedro Monteiro **Nº 97484**

Índice

Introdução

No segundo trabalho prático da unidade curricular de Sistemas Operativos foi proposta a realização da simulação de um encontro de futebol, entre duas equipas, com jogadores de campo e guarda-redes, e um árbitro, abordando temas como sincronização de processos, semáforos e memória partilhada.

Abordagem do problema

Começámos por ler e interpretar o enunciado disponibilizado na plataforma do e-learning, bem como todo o código base apresentado no ficheiro semaphore_soccergame.tgz.

Percebemos então que iríamos fazer uma simulação de um jogo de futebol em que existe um árbitro (referee) e duas equipas, sendo estas constituídas especificamente por 4 jogadores de campo (players) e 1 guarda-redes (goalies). Caso existam jogadores a mais para jogar devem ser avisados de que chegaram atrasados e serão descartados, pelo que não entram no jogo.

O árbitro desempenha um papel fundamental, pois é ele que decide quando o jogo começa e quando o jogo termina.

Todos os intervenientes neste jogo são processos independentes, pelo que é necessário que seja realizada a sua sincronização e comunicação através de semáforos e memória partilhada, sendo este o principal tema deste trabalho.

Em todo o relatório irão ser apenas analisadas as zonas em que foi inserido código, sinalizadas pelo docente através de TODO.

Parâmetros iniciais

```
/* Generic parameters */

/** \brief total number of players */
#define  NUMPLAYERS       10
/** \brief total number of goalies */
#define  NUMGOALIES        3
/** \brief total number of referees */
#define  NUMREFEREES       1

/** \brief number of players in each team */
#define  NUMTEAMPLAYERS     4
/** \brief number of goalies in teach team */
#define  NUMTEAMGOALIES     1
```

Foram definidos alguns parâmetros genéricos, que podemos encontrar no ficheiro *probConst.h*, que nos indicam o número total de intervenientes no jogo, tais como, o número de jogadores (10), o número de guarda-redes (3) e o número de árbitros (1).

Foram definidos também 2 parâmetros que mostram como é constituída cada equipa, 4 jogadores de campo e 1 guarda-redes.

```
/* Player/Goalie state constants */

/** \brief player/goalie initial state, arriving */
#define  ARRIVING          0
/** \brief player/goalie waiting to constitute team */
#define  WAITING_TEAM      1
/** \brief player/goalie waiting to constitute team */
#define  FORMING_TEAM      2
/** \brief player/goalie waiting for referee to start game in team 1 */
#define  WAITING_START_1   3
/** \brief player/goalie waiting for referee to start game in team 2 */
#define  WAITING_START_2   4
/** \brief player/goalie playing in team 1 */
#define  PLAYING_1         5
/** \brief player/goalie playing in team 2 */
#define  PLAYING_2         6
/** \brief player/goalie playing */
#define  LATE              7

/* Referee state constants */

/** \brief referee initial state, arriving  */
#define  ARRIVING          0
/** \brief referee waiting for both teams  */
#define  WAITING_TEAMS     1
/** \brief referee starting game  */
#define  STARTING_GAME     2
/** \brief referee refereeing  */
#define  REFEREEING        3
/** \brief referee ending game  */
#define  ENDING_GAME       4
```

Para além dos parâmetros genéricos definidos no início, foram também definidas constantes que indicam o que significa o estado em que se encontram os jogadores, o guarda-redes e o árbitro.

Exemplificando, o valor 2, para o caso dos jogadores e do guarda-redes, indica que estes estão a formar equipa, enquanto que, no caso do árbitro, o valor 2 significa o início do jogo.

Estas constantes irão ser usadas ao longo de todo o código, facilitando assim a implementação, pelo que se torna mais visível os estados para os quais as entidades transitam.

	Pequena descrição	SemUp	SemDown
mutex	Permite saber se estamos dentro da zona crítica	Usado para sair da zona crítica	Usado para entrar na zona crítica
playersWaitTeam	Usado pelos jogadores para esperarem que a equipa seja formada	Usado para desbloquear os jogadores para se atribuirem a si próprios a uma equipa	Usado para bloquear os jogadores para que possam esperar que o elemento que vai fazer as equipas lhes diga qual a equipa à qual se devem atribuir
goaliesWaitTeam	Usado pelos guarda-redes para esperarem que a equipa seja formada	Usado para desbloquear os guarda-redes para se atribuirem a si próprios a uma equipa	Usado para bloquear os guarda-redes para que possam esperar que o elemento que vai fazer as equipas lhes diga qual a equipa à qual se devem atribuir
playersWaitReferee	Usado pelos jogadores e pelos guarda-redes para esperarem que o árbitro sinalize o começo do jogo	Usado pelo árbitro para desbloquear os jogadores e os guarda-redes para que possam jogar	Usado pelos jogadores e pelos guarda-redes para se bloquearem, até que o jogo comece
playersWaitEnd	Usado pelos jogadores e pelos guarda-redes para esperarem que o árbitro sinalize o fim do jogo	Usado pelo árbitro para desbloquear os jogadores e os guarda-redes após o fim do jogo	Usado pelos jogadores e pelos guarda-redes para se bloquearem, de modo a jogarem o jogo até ao final
refereeWaitTeams	Usado pelo árbitro para esperar que as equipas sejam formadas	Usado pelos jogadores e pelos guarda-redes para desbloquear	Usado pelo árbitro para se bloquear para que possa esperar que ambas as equipas se formem
playerRegistered	Usado pelos jogadores e pelos guarda-redes para confirmarem que estão numa equipa	Usado pelos jogadores e pelos guarda-redes para se desbloquearem para que possam responder a quem está a fazer as equipas	Usado pelos jogadores e pelos guarda-redes para se bloquearem para que possam esperar pela resposta dos outros jogadores

Tabela 1 - Semáforos usados ao longo do código

Código utilizado

semSharedMemReferee.c

O primeiro interveniente no jogo que optámos por implementar foi o árbitro, responsável pelo início e pelo fim do jogo.

Interage com todas as outras entidades, jogadores e guarda-redes, pelo que a solução teve como base mecanismos relacionados com sincronização e comunicação através de semáforos e memória partilhada, como referido

anteriormente.

Função arrive()

```
static void arrive ()
{
    if (semDown (semgid, sh->mutex) == -1) {
/* enter critical region */
        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) {
/* 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);

}
```

Na função *arrive()*, após entrarmos na zona crítica, alteramos o estado do árbitro para ARRIVING, indicando que o árbitro acabou de chegar, e guardamos as alterações imediatamente para o ficheiro log.

Função waitForTeams()

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

    /* TODO: insert your code here */
    //referee waits for team 1
    if (semDown (semgid, sh->refereeWaitTeams) == -1) {
/* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }
    //referee waits for team 2
    if (semDown (semgid, sh->refereeWaitTeams) == -1) {
/* leave critical region */
        perror ("error on the up operation for semaphore access (RF)");
        exit (EXIT_FAILURE);
    }
}
```

Na função waitForTeams(), alteramos o estado do árbitro para WAITING_TEAMS dentro da zona crítica, uma vez que a escrita de ficheiros deve ser sempre feita exclusivamente para evitar potenciais erros e guardamos as alterações imediatamente para o ficheiro log.

Fora da zona crítica, bloqueamos o árbitro duas vezes (uma vez para cada equipa) com o semáforo *refereeWaitTeams* para ele aguardar que as equipas estejam formadas antes de começar o jogo.

Função startGame()

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

    /* TODO: insert your code here */
    for (int i = 0; i < NUMPLAYERS; i++) {
        if (semUp(semgid, sh->playersWaitReferee) == -1)
        { /* leave critical region */
            perror("error on the up operation for semaphore access
(RF)");
            exit(EXIT_FAILURE);
        }
    }
}
```

Na função *startGame()*, alteramos o estado do árbitro para STARTING_GAME dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, desbloqueamos o semáforo *playersWaitReferee* dez vezes (uma vez por cada jogador e guarda-redes que pertencem a uma equipa) que é usado pelos jogadores e pelos guarda-redes para esperarem que o árbitro sinalize o começo do jogo, dando assim início à partida.

Função play()

```
static void play ()
{
    if (semDown (semgid, sh->mutex) == -1) {
/* enter critical region */
```

```
        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) {
/* leave critical region */
        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()*, alteramos o estado do árbitro para REFEREEING, dentro da zona crítica, para que o jogo seja iniciado, e guardamos as alterações.

Função endGame()

```
static void endGame ()
{
```

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

    /* TODO: insert your code here */
    for (int i = 0; i < NUMPLAYERS; i++) {
        if (semUp (semgid, sh->playersWaitEnd) == -1) {
/* leave critical region */
            perror ("error on the up operation for semaphore access
(RF)");
            exit (EXIT_FAILURE);
        }
    }
}
```

Na função *endGame()*, alteramos o estado do árbitro para ENDING_GAME dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, desbloqueamos o semáforo *playersWaitEnd* dez vezes (uma vez por cada jogador e guarda-redes que estão a jogar) que é usado pelos jogadores e pelos guarda-redes para esperarem que o árbitro sinalize o fim do jogo, dando assim a partida por concluída.

semSharedMemGoalie.c

Em segundo lugar, decidimos tratar do goalie (guarda-redes), implementando o código necessário para que esta entidade ficasse funcional e pudesse interagir com o resto dos jogadores, para formar equipa, e com o árbitro, para que o jogo pudesse começar.

Função arrive()

```
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);
}
```

Na função *arrive()*, alteramos o estado do goalie para ARRIVING dentro da zona crítica, uma vez que a escrita de ficheiros deve ser sempre feita exclusivamente para evitar potenciais erros, e guardamos as alterações imediatamente para o ficheiro log.

Função goalieConstituteTeam()

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

    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.goaliesArrived++;

    //se chegou atrasado
    if (sh->fSt.goaliesArrived > 2*NUMTEAMGOALIES) {
        sh->fSt.st.goalieStat[id] = LATE;
        saveState(nFic, &sh->fSt);
    }
    // se nao chegou atrasado
    else {
        sh->fSt.goaliesFree++;
        // Verificar se tem recursos para formar equipa
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS) {
            sh->fSt.st.goalieStat[id] = FORMING_TEAM;
            saveState(nFic, &sh->fSt);

            //enviar sinal para os players se desbloquearem e atribuirem
equipa
            for (int i = 0; i < NUMTEAMPLAYERS; i++) {
                if (semUp(semgid, sh->playersWaitTeam) == -1) {
                    perror("error on the down operation for semaphore
access (GL)");
                    exit(EXIT_FAILURE);
                }
                sh->fSt.playersFree--; // 4 players que deixam de estar
free
            }
            sh->fSt.goaliesFree--; // o goalie deixa de estar free

            //ficar bloqueado à espera dos jogadores responderem que já
têm equipa
            for (int i = 0; i < NUMTEAMPLAYERS; i++) {
                if (semDown (semgid, sh->playerRegistered) == -1) {
/* exit critical region */
```

```
                    perror ("error on the down operation for semaphore
access (GL)");
                    exit (EXIT_FAILURE);
                }
            }

            //alterar o team id depois da equipa estar formada
            ret = sh->fSt.teamId;
            sh->fSt.teamId = 2;
        }
        else {
            //Não tem recursos para formar equipa então vai ficar à
espera
            sh->fSt.st.goalieStat[id] = WAITING_TEAM;
            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);
    }
    /* TODO: insert your code here */
    //Baixar o semáforo goaliesWaitTeam se não chegou atrasado e não
forma equipa
    if (sh->fSt.goaliesFree <= 2*NUMTEAMGOALIES &&
sh->fSt.st.goalieStat[id] != LATE) {
        if (sh->fSt.st.goalieStat[id] == WAITING_TEAM) {
            //fica bloqueado à espera do constitute team do player
            if (semDown (semgid, sh->goaliesWaitTeam) == -1) {
/* exit critical region */
                perror ("error on the down operation for semaphore
access (GL)");
                exit (EXIT_FAILURE);
            }
            //aqui já está desbloqueado e já sabemos o teamID
            ret = sh->fSt.teamId;

            //Responder que já tem equipa ao player que está a formar as
equipas
            if (semUp (semgid, sh->playerRegistered) == -1) {
/* exit critical region */
                perror ("error on the down operation for semaphore
access (GL)");
```

```
                exit (EXIT_FAILURE);
            }
        }
        else { //se não está waiting_team, quer dizer que está
waiting_start (está pronto)
            if (semUp (semgid, sh->refereeWaitTeams) == -1) {
/* exit critical region */
                perror ("error on the down operation for semaphore
access (GL)");
                exit (EXIT_FAILURE);
            }
        }
    }

    return ret;
}
```

Na função *goalieConstituteTeam()*, dentro da região crítica, começamos por incrementar a variável *goaliesArrived* que regista o número de guarda-redes que já chegaram ao campo de futebol.

Depois, verificamos se o guarda-redes em questão chegou atrasado com a condição *if (sh->fSt.goaliesArrived >= 2*NUMTEAMGOALIES)* e em caso afirmativo, alteramos o seu estado para LATE e guardamos a alteração no ficheiro de logs. Caso contrário, incrementamos a variável *goaliesFree* e averiguamos se se verificam as condições para o guarda-redes formar a equipa.

Se a condição não se verificar, altera-se o estado do goalie para WAITING_TEAM e guarda-se a alteração de estado para o ficheiro de logs. Por outro lado, se o número de jogadores disponíveis for maior ou igual a 4, então as condições verificam-se e altera-se o estado do guarda-redes para FORMING_TEAM, guarda-se a alteração de estado no ficheiro log e desbloqueia-se o semáforo *playersWaitTeam,* que serve para sinalizar aos jogadores que já se podem atribuir a uma equipa (identificada pelo teamId), quatro vezes com o ciclo *for*. Dentro desse mesmo ciclo, também é decrementada a variável *playersFree* a cada uma das quatro iterações e, por fim, é decrementada uma vez a variável *goaliesFree.*

Após este processo, é a vez do goalie ficar bloqueado à espera da resposta dos jogadores que necessitam confirmar que se conseguiram atribuir

à equipa bem sucedidamente, com o semáforo *playersRegistered*, que é baixado quatro vezes, uma por cada jogador.

Finalmente, guardamos o teamId para a variável de retorno da função "*ret*" e alteramos o teamId para 2.

Fora da zona crítica, se o goalie não chegou atrasado e está à espera de se juntar a uma equipa, baixa-se o semáforo *goaliesWaitTeam*, que é usado para bloquear o guarda-redes para que possa esperar que o elemento que vai fazer as equipas lhe diga a qual equipa se deve juntar.

Seguidamente guardamos o teamId (que foi atualizado pelo jogador que forma as equipas) na variável de retorno *"ret"* e sinalizamos ao jogador que está a formar as equipas que o goalie se juntou com sucesso a uma equipa.

Finalmente, se o goalie não chegou atrasado, mas não está no estado WAITING_TEAM, então é porque foi ele que formou as equipas e encontra-se agora no estado WAITING_START tendo o dever de avisar o árbitro que pode começar a partida levantando o semáforo *refereeWaitTeams*.

Função waitReferee()

```
static void waitReferee (int id, int team)
{
    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 */
    //atribuir equipa ao goalie que forma a equipa
    if (team == 1) {
        sh->fSt.st.goalieStat[id] = WAITING_START_1;
    }
    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 down operation for semaphore access
(GL)");
        exit (EXIT_FAILURE);
    }

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

Na função *waitReferee()*, alteramos o estado do goalie para WAITING_START_1 se pertencer à team 1 ou para WAITING_START_2 se pertencer à team 2 dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, bloqueamos o semáforo *playersWaitReferee* que é usado pelo goalie para esperar que o árbitro sinalize o começo do jogo.

Função playUntilEnd()

```
static void playUntilEnd (int id, int team)
{
    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 */
    if (team == 1) {
        sh->fSt.st.goalieStat[id] = PLAYING_1;
    } 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 down operation for semaphore access
(GL)");
        exit (EXIT_FAILURE);
    }

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

}
```

Na função *playUntilEnd()*, alteramos o estado do goalie para PLAYING_1 se pertencer à team 1 ou para PLAYING_2 se pertencer à team 2 dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, bloqueamos o semáforo *playersWaitEnd* que é usado pelos goalie para esperar que o árbitro sinalize o fim do jogo, dando assim a partida por concluída.

semSharedMemPlayer.c

Por fim, implementámos todo o código necessário para que também o

player pudesse participar no jogo, comunicando, através de memória partilhada, com todos os outros intervenientes.

Função arrive()

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

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

Na função *arrive()*, alteramos o estado do jogador para ARRIVING dentro da zona crítica, uma vez que a escrita de ficheiros deve ser sempre feita exclusivamente para evitar potenciais erros, e guardamos as alterações imediatamente para o ficheiro log.

Função playerConstituteTeam()

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

    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 */

    //se chegou atrasado
    sh->fSt.playersArrived++;

    if (sh->fSt.playersArrived > 2*NUMTEAMPLAYERS) {
        sh->fSt.st.playerStat[id] = LATE;
        saveState(nFic, &sh->fSt);
    }
    // se nao chegou atrasado
    else {
        // Verificar se tem recursos para formar equipa
        sh->fSt.playersFree++;
        if (sh->fSt.playersFree >= NUMTEAMPLAYERS && sh->fSt.goaliesFree
>= 1) {
            sh->fSt.st.playerStat[id] = FORMING_TEAM;
            saveState(nFic, &sh->fSt);
            //enviar sinal para os players se desbloquearem e atribuirem
equipa
            for (int i = 0; i < NUMTEAMPLAYERS-1; i++) {
                if (semUp(semgid, sh->playersWaitTeam) == -1) {
                    perror("error on the down operation for semaphore
access (GL)");
                    exit(EXIT_FAILURE);
                }
                sh->fSt.playersFree--;
            }
            if (semUp(semgid, sh->goaliesWaitTeam) == -1) {
                perror("error on the down operation for semaphore access
(GL)");
                exit(EXIT_FAILURE);
            }
            sh->fSt.goaliesFree--;
            sh->fSt.playersFree--; // ele proprio, o que faz a team
```

```
            //ficar bloqueado à espera dos jogadores responderem que já
têm equipa
            for (int i = 0; i < NUMTEAMPLAYERS; i++) {
                if (semDown (semgid, sh->playerRegistered) == -1) {
/* exit critical region */
                    perror ("error on the down operation for semaphore
access (GL)");
                    exit (EXIT_FAILURE);
                }
            }

            //alterar o team id depois da equipa estar formada
            ret = sh->fSt.teamId;
            sh->fSt.teamId = 2;
        }
        else {
            //Não tem recursos para formar equipa então vai ficar à
espera
            sh->fSt.st.playerStat[id] = WAITING_TEAM;
            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);
    }
    /* TODO: insert your code here */
    if ((sh->fSt.playersFree < NUMTEAMPLAYERS || sh->fSt.goaliesFree ==
0) || sh->fSt.st.playerStat[id] != LATE) {
        if (sh->fSt.st.playerStat[id] == WAITING_TEAM)
        {
            if (semDown (semgid, sh->playersWaitTeam) == -1) {
/* exit critical region */
                perror ("error on the down operation for semaphore
access (GL)");
                exit (EXIT_FAILURE);
            }

            ret = sh->fSt.teamId;

            //Responder que já tem equipa ao player que está a formar as
equipas
```

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

    return ret;
}
```

Na função *playerConstituteTeam()*, dentro da região crítica, começamos por incrementar a variável *playersArrived* que regista o número de jogadores que já chegaram ao campo de futebol.

Depois, verificamos se o jogador em questão chegou atrasado com a condição *if (sh->fSt.playersArrived >= 2*NUMTEAMPLAYERS)* e em caso afirmativo, alteramos o seu estado para LATE e guardamos a alteração no ficheiro de logs. Caso contrário, incrementamos a variável *playersFree* e averiguamos se se verificam as condições para o jogador formar a equipa.

Se a condição não se verificar, altera-se o estado do jogador para WAITING_TEAM e guarda-se a alteração de estado para o ficheiro de logs. Por outro lado, se existirem mais outros três jogadores disponíveis e mais um guarda-redes disponível, então as condições verificam-se e altera-se o estado do jogador para FORMING_TEAM, guarda-se a alteração de estado no ficheiro log e desbloqueia-se o semáforo *playersWaitTeam,* que serve para sinalizar aos jogadores que já se podem atribuir a uma equipa (identificada pelo teamId), três vezes com o ciclo *for*. Dentro desse mesmo ciclo, também é decrementada a variável *playersFree* a cada uma das três iterações e, por fim, é também desbloqueado o semáforo *goaliesWaitTeam* pelo mesmo motivo e são

decrementadas uma vez as variáveis *goaliesFree* e *PlayersFree* (esta última é decrementada uma quarta vez pois temos de contar com o jogador que está a formar a equipa, que também deixa de estar disponível).

Após este procedimento, o jogador que forma as equipas fica bloqueado à espera da resposta dos restantes jogadores e guarda-redes que necessitam confirmar que se conseguiram atribuir à equipa com sucesso, através do semáforo *playersRegistered*, que é baixado quatro vezes, uma vez por cada processo.

Finalmente, guardamos o teamId para a variável de retorno da função "*ret*" e alteramos o teamId para 2.

Fora da zona crítica, se o jogador não chegou atrasado e está à espera de se juntar a uma equipa, baixa-se o semáforo *playersWaitTeam*, que é usado para bloquear o jogador para que este possa esperar que o elemento que vai formar as equipas lhe diga a qual equipa se deve juntar.

Seguidamente guardamos o teamId (que foi atualizado pelo jogador ou guarda-redes que forma as equipas) na variável de retorno *"ret"* e sinalizamos ao elemento que está a formar as equipas que o jogador se conseguiu juntar com sucesso a uma equipa.

Finalmente, se o jogador não está no estado WAITING_TEAM, significa que foi ele que formou as equipas e então, é necessário avisar o árbitro que pode começar a partida levantando o semáforo *refereeWaitTeams*.

Função waitReferee()

```
static void waitReferee (int id, int team)
{
```

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

    /* TODO: insert your code here */
    //atribuir equipa ao goalie que forma a equipa
    if (team == 1) {
        sh->fSt.st.playerStat[id] = WAITING_START_1;
    }
    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 down operation for semaphore access
(PL)");
        exit (EXIT_FAILURE);
    }

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

}
```

Na função *waitReferee()*, alteramos o estado do player para WAITING_START_1 se pertencer à team 1 ou para WAITING_START_2 se pertencer à team 2 dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, bloqueamos o semáforo *playersWaitReferee* que é usado pelo jogador para esperar que o árbitro sinalize o começo do jogo.

Função playUntilEnd()

```
static void playUntilEnd (int id, int team)
```

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

    /* TODO: insert your code here */
    if (team == 1) {
        sh->fSt.st.playerStat[id] = PLAYING_1;
    } 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 down operation for semaphore access
(PL)");
        exit (EXIT_FAILURE);
    }

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

Na função *playUntilEnd()*, alteramos o estado do jogador para PLAYING_1 se pertencer à team 1 ou para PLAYING_2 se pertencer à team 2 dentro da zona crítica e guardamos as alterações.

Fora da zona crítica, bloqueamos o semáforo *playersWaitEnd* que é usado pelos jogadores para esperar que o árbitro sinalize o fim do jogo, dando assim a partida por concluída.

Resultados obtidos

De maneira a visualizar a solução, e com o objetivo de verificarmos que a solução implementada está correta, o programa foi testado múltiplas vezes, sendo para isso corrido várias vezes, através da makefile fornecida pelo docente.

Sempre que o programa é corrido (./probSemSharedMemSoccerGame) os resultados que são obtidos no terminal são diferentes, pelo que é necessário verificar se esses resultados estão, efetivamente, corretos, por exemplo verificar se os jogadores estão a formar bem as equipas, e se esperam (no estado WAITING_START_1, para o caso da equipa 1), pelo árbitro.

Ao longo da execução foram encontrados vários deadlocks, sendo que, para percebermos melhor onde estava o erro, fizemos várias tentativas de debugging por injeção ao processo de forma a visualizar a sua stack, através de "*sudo gdb <executável> <ID do processo>*".

Realizamos ainda um teste adicional que consistiu em correr o programa um elevado número de vezes, neste caso 1000 (./run.sh 1000), onde não ocorreram deadlocks e todas as vezes os estados eram atribuídos e alterados como o suposto.

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

Figura 1 - Resultados Obtidos

Em relação à figura 1 é de notar que:

➜ As colunas 1 a 10 correspondem aos jogadores

➜ As colunas 11 a 13 correspondem aos guarda-redes

➜ A última coluna corresponde ao árbitro

➜ Ao longo das linhas podemos observar a variação dos estados de cada entidade

Analisando o resultado do output verificamos que, inicialmente, todas as entidades se encontram com o valor 0, indicando que estão a chegar (ARRIVING).

Após um período de tempo aleatório, os valores dos estados para os jogadores são alterados para 1, ou seja, estão prontos a formar equipa, pelo que os últimos dois jogadores a chegar passam automaticamente para o estado 7 (LATE), não podendo participar no jogo (no exemplo da figura verificamos que os jogadores que chegaram atrasados foram o P04 e o P09).

Olhando para os goalies podemos ver que se encontram no estado 2, o que indica que vão ser estes os responsáveis por formar a equipa, sendo o goalie G01 descartado por ter chegado atrasado (fica com o valor 7). Cada entidade é responsável pelo seu estado, não podendo alterar o estado de outras.

As equipas começam a formar-se, com a alteração do estado dos jogadores para 3, correspondente à equipa 1 (WAITING_START_1) e para 4, correspondente à equipa 2(WAITING_START_2).

Depois de estarem formadas as equipas o árbitro pode começar o jogo, e ocorrendo uma mudança do seu estado para 2 (STARTING_GAME). Após esta mudança há um período de jogo, onde o árbitro se encontra em REFEREEING (estado 3), e os jogadores e os guarda-redes em PLAYING (estados 5 e 6 consoante a equipa).

Quando o árbitro decidir pode terminar o jogo, e o valor do seu estado é alterado para 4.

Conclusão

De referir que com a execução deste trabalho prático temas como semáforos, memória partilhada e sincronização foram estudados, sendo esse o nosso principal objetivo.

Após compararmos os resultados obtidos com os resultados que o docente mostrou durante as aulas práticas podemos concluir que se encontram bastante semelhantes, pelo que se pode dizer que toda a comunicação entre entidades está bem executada, não havendo deadlocks, sendo esta uma possibilidade de código para que o jogo seja realizado com sucesso.