Sistemas Operativos- 40381

# 2º Trabalho
# (Simulação jogo de futebol)

Tomás Hilário - 119896

Diogo Duarte - 120482

deti universidade de aveiro
departamento de eletrónica,
telecomunicações e informática

# Introdução

Neste trabalho foi-nos pedido a realização de um conjunto de funções/scripts em C que simulassem um jogo de futebol constituído por três entidades diferentes: player, *goalie* e referee. Cada uma destas entidades representa um processo e cada um dos seus estados são controlados por semáforos. Para que um jogo se possa realizar são necessários pelo menos 10 jogadores, 8 players (4 por equipa) mais 2 *goalies* (1 por equipa) e um árbitro. Cada uma das entidades tem os seguintes estados:

### ***Player/ Goalie*:**

- ARRIVING (a chegar)
- WAITING_TEAM (à espera para formar equipa)
- FORMING_TEAM (a formar equipa)
- WAITING_START_1/WAITING_START_2 (à espera de começar o jogo numa determinada equipa)
- PLAYING_1/PLAYING_2 (a jogar numa determinada equipa)
- LATE (chegou atrasado)

### ***Referee*:**

- ARRIVINGR (a chegar)
- WAITING_TEAMS (à espera de equipas)
- STARTING_GAME (a começar o jogo)
- REFEREEING (a arbitrar)
- ENDING_GAME (a terminar o jogo)

# Explicação dos semáforos

- ***Mutex*:** É decrementado à entrada da região crítica, bloqueando quaisquer outros processos que queiram aceder a esta região. O incrementar do semáforo marca a saída da região crítica.
- ***playersWaitTeam***: Utilizado para bloquear *player* até que haja uma equipa a que o mesmo se possa juntar.
- ***goaliesWaitTeam***: Utilizado para bloquear *goalie* até que haja uma equipa a que o mesmo se possa juntar.
- ***playersWaitReferee***: Utilizado para bloquear *players* e *goalies* até que o *referee* dê o jogo como começado.
- ***playing***: Utilizado para bloquear *referee* até que os *players* e *goalies* estejam prontos para jogar.
- ***playersWaitEnd***: Bloqueia o processo dos players e goalies até que o *referee* termine o jogo.
- ***refereeWaitTeams***: Utilizado para bloquear processo do *referee* até que ambas as equipas estejam completamente formadas.

- ***playerRegistered***: Utilizado para bloquear processos dos *players* e *goalies* até que os mesmo estejam registados numa equipa.

# Desenvolvimento de cada módulo

## Goalie (SemSharedMemGoalie.c)

### 1.arrive()

Nesta função apenas foi adicionado código que altera e guarda o estado do **"goalie"** *para* ***"ARRIVING"*** *da seguinte maneira:*

```
1  sh->fSt.st.goalieStat[id] = ARRIVING;
2  saveState(nFic, &sh->fSt);
```

### 2.goalieConstituteTeam()

Ao chegar, o "goalie" é contabilizado como presente e disponível para formar equipas. Caso ainda haja vagas para "goalies" no jogo (goaliesArrived <= 2), o sistema verifica se há "players" e "goalies" suficientes para formar uma equipa. Se não houver recursos suficientes, o **"goalie"** entra no estado **"WAITING_TEAM"** e aguarda a oportunidade de formar uma equipa.

```
1  sh->fSt.goaliesFree++;          // Incrementar gaoliesFree
2  sh->fSt.goaliesArrived++;       // Incrementar golaiesArrived
3
4  if (sh->fSt.goaliesArrived <= 2) {
5
6      if ((sh->fSt.playersFree < NUMTEAMPLAYERS) || (sh->fSt.goaliesFree < NUMTEAMGOALIES)) {
7
8          sh->fSt.st.goalieStat[id] = WAITING_TEAM;
9          saveState(nFic, &sh->fSt);
10     }
```

Quando há "players" e "goalies" suficientes para formar uma equipa, o **"goalie"** atualiza seu estado para **"FORMING_TEAM"**. O sistema decrementa os contadores de "players" e "goalies" disponíveis, liberta os jogadores para formar a equipa e espera que eles se registem antes de sinalizar ao "referee" que a equipa está pronta.

```
else {   // Define o estado do player como Waiting_Team
    sh->fSt.st.goalieStat[id] = FORMING_TEAM;
    saveState(nFic, &sh->fSt);

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

    for (int i = 0; i < NUMTEAMPLAYERS; i++) {
        if (semUp(semgid, sh->playersWaitTeam) == -1) {
            perror("Error on the up operation for semaphore access (GL)");
            exit(EXIT_FAILURE);
        }
        // Espera que os jogadores estejam registrados na equipa
        if (semDown(semgid, sh->playerRegistered) == -1) {
            perror("error on the up operation for semaphore access (GL)");
            exit(EXIT_FAILURE);
        }

    }

    ret = sh->fSt.teamId++;

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

Caso o “goalie” tenha de aguardar, o mesmo entra no estado de espera por uma equipa formada. Após ser notificado de que a equipe foi completa, o “goalie” é registado como parte da equipa, e o identificador da equipa é retornado.

```
if (sh->fSt.st.goalieStat[id] == WAITING_TEAM) {
    if (semDown(semgid, sh->goaliesWaitTeam) == -1) {
        perror("error on the up operation for semaphore access (PL)");
        exit(EXIT_FAILURE);
    }

    ret = sh->fSt.teamId;       // atribuir id de equipa

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

### 3.WaitReferee()

Primeiro o estado do **“goalie”** é atualizado para indicar que ele está à espera do início do jogo (**“WAITING_START_1”** ou **“WAITING_START_2”**, dependendo da equipa).

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

Em seguida, o **“goalie”** aguarda a autorização do “referee” através do semáforo **“playersWaitReferee”**. Após ser desbloqueado, o “goalie” sinaliza que está pronto para jogar incrementando o semáforo **“playing”**, indicando ao *referee* que ele está pronto para o início do jogo.

```
// Faz o guarda redes esperar pelo arbitro
if (semDown(semgid, sh->playersWaitReferee) == -1) {
    perror("error on the up operation for semaphore access(GL)");
    exit(EXIT_FAILURE);
}

// Sinal ao arbitro que o redes está pronto
if (semUp(semgid, sh->playing) == -1) {
    perror("Error signaling referee (waitReferee)");
    exit(EXIT_FAILURE);
}
```

### 4.playUntilEnd()

Primeiramente o estado do **“goalie”** é atualizado para indicar que ele está a jogar (**“PLAYING_1”** ou **“PLAYING_2”**, dependendo da equipa), e essa alteração é registada no log.

```
sh->fSt.st.goalieStat[id] = (team == 1) ? PLAYING_1 : PLAYING_2;
saveState(nFic, &sh->fSt);
```

Em seguida, o **“goalie”** aguarda o sinal do “referee” para o fim do jogo, bloqueando o semáforo **“playersWaitEnd”**, garantindo que ele só prossiga quando o jogo for finalizado.

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

## Player (SemSharedMemPlayer.c)

### 1.arrive()

Nesta função apenas foi adicionado código que altera e guarda o estado do *player* para "**ARRIVING**" da seguinte maneira:

```
1  sh->fSt.st.playerStat[id] = ARRIVING;
2  saveState(nFic, &sh->fSt);
```

## 2.playerConstituteTeam()

Após a chegada do jogador irá ser-lhe atribuída uma equipa e para tal, é necessário a chamada da função **playerConstituteTeam()**. Em primeiro lugar, dada a chegada do jogador, o número de jogadores livres (***playersFree***) e jogadores que chegaram (***playersArrived***) é incrementado uma unidade.

```
1  sh->fSt.playersFree++;
2  sh->fSt.playersArrived++;
```

De seguida, é verificado se o *player* em questão se encontra ou não atrasado (***"LATE"***) e é lhe atribuído esse estado se tal se verificou. Um player está atrasado quando o número de jogadores que já se encontram no local é igual a 8.

```
1  if (sh->fSt.playersArrived <= 8) {...}
2  else {
3      ret = 0;
4      sh->fSt.st.playerStat[id] = LATE;
5      saveState(nFic, &sh->fSt);
6  }
```

Se o *player* não estiver atrasado precisamos de verificar se existem condições de se formar uma equipa, ou seja, se existem pelo menos quatro *players* e um *goalie* livres. Nos casos em que isto se verifica, o estado do *player* é alterado para **"FORMING_TEAM"** e são decrementadas a variáveis ***playersFree*** e ***goaliesFree*** quatro e uma unidades, respetivamente, uma vez que essas cinco entidades já não se encontram livres. Também é necessário passar a informação ao *goalie* e aos restantes *players* que os mesmos se podem juntar à equipa desbloqueando o semáforo ***goaliesWaitTeam*** uma vez e o semáforo ***playersWaitTeam*** 3 vezes. Além disso, é decrementado o semáforo ***playerRegistered*** um total de quatro vezes (três para os *players* e uma para o *goalie*) para que os mesmos sejam registados na equipa. Por fim, é guardado e incrementado o id da equipa para que a próxima equipa tenha um id diferente e é desbloqueado o semáforo ***refereeWaitTeams*** para que o *referee* se aperceba da formação das equipas.

```
1  if ((sh->fSt.playersFree < NUMTEAMPLAYERS) || (sh->fSt.goaliesFree < NUMTEAMGOALIES)) {...}
2  else {
3      sh->fSt.st.playerStat[id] = FORMING_TEAM;                                    // D
4      saveState(nFic, &sh->fSt);
5
6      sh->fSt.playersFree -= NUMTEAMPLAYERS;          // Decrementar nº de jogadores que não se er
7      sh->fSt.goaliesFree -= NUMTEAMGOALIES;          // Decrementar nº de goalies que não se encc
8
9      if (semUp(semgid, sh->goaliesWaitTeam) == -1) {
10         perror("Error on the up operation for semaphore access (PL)");      // Desbloquear goali
11         exit(EXIT_FAILURE);
12     }
13
14     if (semDown(semgid, sh->playerRegistered) == -1) {
15         perror("error on the up operation for semaphore access (PL)");      // Espera que goalie
16         exit(EXIT_FAILURE);
17     }
18
19     for (int i = 0; i < NUMTEAMPLAYERS - 1; i++) {
20         if (semUp(semgid, sh->playersWaitTeam) == -1) {
21             perror("Error on the up operation for semaphore access (PL)");  // Desbloquear playe
22             exit(EXIT_FAILURE);
23         }
24
25         if (semDown(semgid, sh->playerRegistered) == -1) {                  // Espera que os joc
26             perror("error on the up operation for semaphore access (PL)");
27             exit(EXIT_FAILURE);
28         }
29
30     }
31
32     ret = sh->fSt.teamId++;     // retirar e incrementar id da equipa
33
34     if (semUp(semgid, sh->refereeWaitTeams) == -1) {                        // Sinalizar referee
35         perror("error on the up operation for semaphore access (PL)");
36         exit(EXIT_FAILURE);
37     }
38 }
```

Se as condições de formação de equipa não se verificarem (menos de quatro *players* e um *goalie* livres), o estado do *player* passa **“WAITING_TEAM”** e o mesmo é bloqueado até que existam condições para se juntar a uma equipa. Além disso, também é sinalizado que o jogador se encontra pronto a registar numa equipa, desbloqueando o semáforo **playerRegistered**.

```
    if ((sh->fSt.playersFree < NUMTEAMPLAYERS) || (sh->fSt.goaliesFree < NUMTEAMGOALIES)) {

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

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

/* TODO: insert your code here ------------------------------------------------------------*/
// Caso em que o jogador se encontra à espera para formar equipa
if (sh->fSt.st.playerStat[id] == WAITING_TEAM) {
    if (semDown(semgid, sh->playersWaitTeam) == -1) {
        perror("error on the up operation for semaphore access (PL)");      // bloquear jogado
        exit(EXIT_FAILURE);
    }

    ret = sh->fSt.teamId;       // atribuir id de equipa

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

**3.**

## WaitReferee()

É pretendido que aquando da chamada desta função, o player se encontra numa equipa e esteja pronto a jogar e, dessa maneira, é lhe atribuído o estado **"WAITING_START_1"** ou **"WAITING_START_2"** consoante a equipa a que o mesmo pertence. Posto isto, o *player* é então bloqueado até que o árbitro se encontre pronto para o desafio e é incrementado o semáforo ***playing*** para sinalizar ao referee de que o *player* se encontra pronto a ir a jogo.

```
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(GL)");
    exit(EXIT_FAILURE);
}

if (semUp(semgid, sh->playing) == -1) {
    perror("Error signaling referee (waitReferee)");
    exit(EXIT_FAILURE);
}
```

## 4. playUntilEnd()

Esta função é responsável por alterar o estado do *player* para **"PLAYING_1"** ou **"PLAYING_2"** consoante a equipa do mesmo e esperar pela conclusão do jogo por parte do árbitro. Nesse sentido, foi alterado e guardado o novo estado do *player* e bloqueado os processos alusivos ao mesmo até que o *referee* dê o jogo como terminado.

**Referee**

```c
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(GL)");
    exit(EXIT_FAILURE);
}
```

## (SemSharedMemReferee.c)

## 1.arrive()

Nesta função apenas foi adicionado código que altera e guarda o estado do *referee* para "**ARRIVINGR**" da seguinte maneira:

```c
sh->fSt.st.refereeStat = ARRIVINGR;
saveState(nFic, &sh->fSt);
```

## 2.waitForTeams ()

Neste ponto da execução, o árbitro já se encontra em campo, encontrando-se à espera da chegada de ambas as equipas, daí termos definido e guardado o seu estado como **"WAITING_TEAMS"**. Além disso, foi decrementado o semáforo ***refereeWaitTeams***, bloqueando o *referee* até que ambas as equipas estejam constituídas. O semáforo foi decrementado duas vezes dada a existência de duas equipas e necessidade de esperar por cada uma delas.

```
sh->fSt.st.refereeStat = WAITING_TEAMS;
saveState(nFic, &sh->fSt);

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);
}
```

## 3.startGame()

Esta função é responsável pela inicialização do jogo a partir do momento em que ambas as equipas estão completas, alterando e guardando o estado do *referee* para **"STARTING_GAME"**. Faz também parte da função passar a informação aos dez jogadores de campo que o jogo pode começar incrementando o semáforo ***refereeWaitTeams***, desbloqueando cada um dos *players* e *goalies* previamente bloqueados. Visto que existem dez jogadores em campo o semáforo é incrementado um total de dez vezes para que todos os intervenientes recebam a informação. Além disso, é decrementado o semáforo ***playing*** uma vez por cada jogador em campo, para que o *referee* espere que todos os jogadores comecem a jogar.

```
sh->fSt.st.refereeStat = STARTING_GAME;
saveState(nFic, &sh->fSt);

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

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

## 4.play()

Pretende-se que durante a chamada desta função o jogo se encontre em andamento, pelo que é apenas necessário alterar o estado do *referee* para **"REFEREEING"**.

```
1  sh->fSt.st.refereeStat = REFEREEING;
2  saveState(nFic, &sh->fSt);
```

### 5.endGame()

Depois de esgotado o tempo útil de jogo é necessário que o *referee* dê o jogo como terminado, passando o seu estado a **"ENDING_GAME"**. Além disso é necessário desbloquear o semáforo ***playersWaitEnd*** para cada jogador em campo que esperava que o *referee* terminasse o jogo. Da mesma maneira que anteriormente, este processo é realizado um total de dez vezes.

```
1  sh->fSt.st.refereeStat = ENDING_GAME;
2  saveState(nFic, &sh->fSt);
3
4  for(int i = 0; i < 10; i++){
5      if(semUp(semgid, sh->playersWaitEnd) == -1){
6          perror ("error on the up operation for semaphore access (RF)");
7          exit (EXIT_FAILURE);
8      }
9  }
```

## Resultado obtido:

| P00 | P01 | P02 | P03 | P04 | P05 | P06 | P07 | P08 | P09 | G00 | G01 | G02 | R01 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | A | A | A | A | A | A | A | A |
| A | A | A | A | A | A | W | A | A | A | A | A | A | A |
| A | A | A | A | A | A | W | A | A | A | A | A | A | A |
| A | A | A | A | A | A | W | A | W | A | A | A | A | A |
| A | A | A | A | A | A | W | A | W | A | A | A | A | A |
| A | A | A | A | A | A | W | W | W | A | A | A | A | A |
| A | A | A | A | A | A | W | W | W | A | A | W | A | A |
| A | A | A | F | A | A | W | W | W | A | A | W | A | A |
| A | W | A | F | A | A | W | W | W | A | A | W | A | A |
| A | W | W | F | A | A | W | W | W | A | A | W | A | A |
| W | W | W | F | A | A | W | W | W | A | A | W | A | A |
| W | W | W | F | A | A | W | W | W | W | A | W | A | A |
| W | W | W | F | L | A | W | W | W | W | A | W | A | A |
| W | W | W | F | L | L | W | W | W | W | A | W | A | A |
| W | W | W | F | L | L | W | W | W | W | A | W | A | W |
| W | W | W | F | L | L | W | W | W | W | A | s | A | W |
| W | W | W | F | L | L | s | W | W | W | A | s | A | W |
| W | W | W | F | L | L | s | W | s | W | A | s | A | W |
| W | W | W | F | L | L | s | s | s | W | A | s | A | W |
| W | W | W | s | L | L | s | s | s | W | A | s | A | W |
| W | W | W | s | L | L | s | s | s | W | A | s | A | W |
| W | W | W | s | L | L | s | s | s | W | A | s | F | W |
| W | S | W | s | L | L | s | s | s | W | A | s | F | W |
| W | S | S | s | L | L | s | s | s | W | A | s | F | W |
| S | S | S | s | L | L | s | s | s | W | A | s | F | W |
| S | S | S | s | L | L | s | s | s | S | A | s | F | W |
| S | S | S | s | L | L | s | s | s | S | A | s | S | W |
| S | S | S | s | L | L | s | s | s | S | A | s | S | S |
| S | S | S | s | L | L | s | s | s | S | L | s | S | S |
| S | S | S | s | L | L | s | s | s | S | L | p | S | S |
| S | S | S | s | L | L | s | s | p | S | L | p | S | S |
| S | S | S | s | L | L | p | s | p | S | L | p | S | S |
| S | S | S | s | L | L | p | p | p | S | L | p | S | S |
| S | S | S | p | L | L | p | p | p | S | L | p | S | S |
| S | P | S | p | L | L | p | p | p | S | L | p | S | S |
| S | P | S | p | L | L | p | p | p | P | L | p | S | S |
| S | P | S | p | L | L | p | p | p | P | L | p | P | S |
| S | P | P | p | L | L | p | p | p | P | L | p | P | S |
| P | P | P | p | L | L | p | p | p | P | L | p | P | S |
| P | P | P | p | L | L | p | p | p | P | L | p | P | R |
| P | P | P | p | L | L | p | p | p | P | L | p | P | E |

## Conclusão

Este projeto ampliou e consolidou o nosso conhecimento teórico e prático sobre semáforos e mostrou a sua importância na sincronização de processos. Aprendemos como utilizar os semáforos para bloquear e desbloquear processos, neste caso, processos relativos a entidades de um jogo de futebol, garantindo o bom funcionamento do jogo.