UNIVERSIDADE da MADEIRA

Sistemas Operativos

2023/2024, 1º Semestre

Projeto Prático - 3.ª Fase

Simulação de um Parque Aquático

Docentes

Eduardo Marques | Fernando Martins

Luís Franco n.º 2083121

André Vieira n.º 2035617

Maria Catarina Freitas n.º 2081621

1. Introdução

Na disciplina de Sistemas Operativos, foi-nos proposto o desenvolvimento de um simulador de Parque Aquático como projeto prático. Este projeto visa aplicar os conceitos aprendidos na disciplina, utilizando a linguagem de programação C para criar uma simulação realista de um aquaparque. A ideia é proporcionar aos alunos a oportunidade de explorar e implementar dinâmicas semelhantes às encontradas em parques aquáticos reais.

A simulação abrange uma variedade de recursos no parque aquático, permitindo aos utilizadores movimentarem-se entre eles. Além disso, a simulação incorpora desafios como possíveis limitações de acesso e lotação em diferentes áreas do parque. O objetivo final é avaliar o funcionamento do sistema, incluindo tempos de espera, gestão de utilizadores e dinâmicas operacionais.

O projeto é flexível, permitindo diferentes abordagens na implementação, refletindo a compreensão e aplicação prática dos conceitos adquiridos na disciplina de Sistemas Operativos. Desta forma, os alunos terão a oportunidade de demonstrar as suas habilidades na criação de um sistema operativo que simula de maneira eficaz as complexidades encontradas em parques aquáticos reais.

2. Descrição do Aquaparque

O Aquaparque abre e a dada altura vão entrando utilizadores até atingir a lotação máxima de 20 pessoas. Caso o aquaparque esteja cheio, os utilizadores ficarão na fila de espera. Existirão 2 filas na entrada do aquaparque, uma para os prioritários e outra para os não prioritários. Cada fila só pode ter no máximo 10 utilizadores. Um utilizador de uma das filas de espera só entrará quando houver pelo menos uma vaga no aquaparque. Os utilizadores prioritários, ou seja, pessoas com deficiência/incapacidade, pessoas idosas, grávidas ou acompanhadas de crianças de colo, irão ser os primeiros a entrar no aquaparque. A prioridade do utilizador será definida aleatoriamente, se é prioritário ou não.

Após a sua entrada no aquaparque, o utilizador tem a possibilidade de escolher entrar em um dos quatro recursos ou então sair do aquaparque, definido aleatoriamente.

Para o 1.º recurso, sendo este uma piscina, apenas poderá haver 10 pessoas no máximo e uma fila de 10 pessoas. O seu tempo de duração será definido aleatoriamente.

Para o 2.º recurso, sendo este um tobogã de crianças e com uma idade máxima de 12 anos, temos 10 boias de 3 lugares cada, não sendo necessário a boia estar completa para ser utilizada (a boia pode ser usada por 1, 2 ou 3 crianças). A quantidade de crianças que vão numa boia é declarada a partir da fila em que estas se encontram. Para este recurso vão existir 3 filas diferentes: uma fila para crianças que querem ir sozinhas; uma fila para crianças que querem ir a pares; e uma fila de crianças que querem ir a trios. A entrada no tobogã é feita, dando oportunidade a uma fila de cada vez partindo da fila de grupos, ou seja, entram 3 crianças da fila de trios, depois entram 2 crianças da fila de pares e seguidamente entra uma criança da fila dos que querem ir sozinhos. As boias são numeradas de 1 a 10 e são usadas sequencialmente, ou seja, a boia que entrou primeiro tem de sair primeiro. Quando a boia retorna esta é utilizada depois das que ainda não foram usadas, por exemplo: quando a boia 1 volta, esta é usada após a boia 10. Esta atração tem como tempo de duração 5 segundos.

Para o 3.º recurso, sendo este um escorrega, apenas poderão estar 3 pessoas e haverá uma fila para 10 pessoas. O seu tempo de duração é de 10 segundos.

Para o 4.º recurso, sendo este um snack-bar, os clientes do aquaparque vão buscar comida. Os clientes podem consumir enquanto a prateleira tiver stock, esta reposta pelos funcionários. Os funcionários irão repor na prateleira enquanto houver espaço nesta.

Quando o utilizador sai do recurso tem, novamente, a possibilidade de escolher entrar em um dos quatro recursos ou sair do aquaparque.

3. Reflexão

- Pergunta: Qual a influência de uma alteração do padrão de chegadas na solução apresentada?
    - Resposta: O padrão de chegadas atual no aquaparque permite que os utilizadores entrem individualmente, escolhendo o recurso desejado imediatamente após a entrada. No entanto, uma modificação no processo, por exemplo, os utilizadores só escolherão o recurso após a entrada de 20 utilizadores. Esta mudança no padrão de chegadas pode impactar significativamente a dinâmica do fluxo de utilizadores e implicar uma maior probabilidade de haver utilizadores nas filas.

- Pergunta: A solução apresentada apresenta uma maior preocupação no uso justo/equilibrado dos recursos ou na eficiência geral do sistema?
    - Resposta: A nossa solução apresenta uma maior preocupação na eficiência geral do sistema pois tivemos o cuidado de limitar, de diferentes maneiras, a quantidade de utilizadores e o tempo de duração dos recursos. Como nos focamos no tempo de duração de cada recurso, por exemplo no recurso 4 o tempo de duração é de 5 segundos, logo por ser tão curto vai haver uma maior afluência de utilizadores no recurso 4. Isto pode criar um mau equilíbrio em relação ao fluxo de utilizadores em cada recurso, mas melhora a eficiência geral do sistema.

- Pergunta: Descrevam, pelo menos, duas limitações da solução apresentada.
    - Resposta: As limitações da solução podem ser, por exemplo, haver uma limitação de utilizadores nos recursos e nas filas, fazendo com que quando isso acontecer o próximo utilizador a ir para o recurso, que tiver a sua lotação e fila cheias, desista do recurso e mude de opção, ir para outro recurso ou sair do aquaparque. Uma outra limitação é quando a lotação máxima das filas do aquaparque é atingida, o utilizador que quer entrar no aquaparque desiste de entrar.

5. Conclusão

Na terceira fase do projeto, conseguimos ter uma visão mais clara do que deveria ser implementado nesta etapa do desenvolvimento. Durante esse período, dedicamos-nos ao planeamento detalhado das operações dos recursos no parque aquático. Definimos políticas de acesso para cada fila, estabelecemos protocolos de comunicação e atribuímos números e parâmetros específicos para cada evento relevante no sistema simulado.

Apesar do progresso feito, encontramos desafios na implementação de algumas funcionalidades essenciais, como a movimentação de usuários entre diferentes recursos, a gestão de prioridades, restrições específicas e o tratamento de desistências por parte dos usuários, entre outras complexidades. Essas dificuldades ofereceram uma oportunidade valiosa para explorarmos os conceitos aprendidos em sala de aula e iniciarmos uma análise mais aprofundada sobre como esses conceitos se traduzem na prática, especialmente no contexto da simulação de um parque aquático.

Essa fase do projeto não apenas consolidou nossos conhecimentos teóricos, mas também nos desafiou a compreender a aplicação prática desses conceitos em um ambiente simulado. O planejamento minucioso e a identificação das lacunas na implementação permitiram-nos ajustar nossa abordagem para as fases subsequentes, visando uma simulação mais completa e fiel aos desafios encontrados em parques aquáticos reais.

A. Código fonte

Ficheiro "configbase.txt":

```
numMaxUtilizadores : 20
numMaxUtilizadoresCriar : 50
numMaxUsersRec1 : 10
numMaxFilaRec1 : 10
numMaxUsersRec2 : 5
numMaxFilaRec2 : 10
numMaxUsersRec3 : 3
numMaxFilaRec3 : 10
numMaxUsersRec4 : 8
numMaxFilaRec4 : 10
numRecursos : 4
maxFilaAquaparque : 10
```

Ficheiro "makefile":

```
all: simulador monitor

monitor: monitor.o
	gcc monitor.c -lpthread -o monitor

simulador: simulador.o
	gcc simulador.c -lpthread -o simulador

clean:
	rm *.o monitor
	rm *.o simulador
```

Ficheiro “util.h”:

```
// Bibliotecas
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdbool.h>  // Para resolver o erro "unknown type name 'bool'"

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <semaphore.h>
#include <pthread.h>
#include <ctype.h>
#include <time.h>

// Tempos
#define _TIMESPEC_DEFINED

// Cores
#define RED "\x1B[31m"
#define GRN "\x1B[32m"
#define YEL "\x1B[33m"
#define BLU "\x1B[34m"
#define MAG "\x1B[35m"
#define CYN "\x1B[36m"
#define WHT "\x1B[37m"
#define RESET "\x1B[0m"

// Defines
#define UNIXSTR_PATH "/tmp/s.2083121"
#define LINHAMAX 512 // Tamanho máximo da linha

#define MAX_USERS_AQUAPARQUE 20
#define MAX_FILA_AQUAPARQUE 10
#define MAX_RECURSO1 10
#define MAX_RECURSO2 5
#define MAX_RECURSO3 3
```

```
#define MAX_RECURSO4 8
#define MAX_FILA_RECURSO1 10
#define MAX_FILA_RECURSO2 10
#define MAX_FILA_RECURSO3 10
#define MAX_FILA_RECURSO4 10

// Estruturas de Dados
struct aquaparque
{
	int qtdRecursos;
	int qtdUtilizadores;
	int maxUtilizadores;
	struct recurso *recursos;
};

struct recurso
{
	int numRecurso;
	int numUtilizadoresDentro;
	int numUtilizadoresFila;
	int numMaxUtilizadoresDoRecurso;
};

struct utilizador
{
	int idUtilizador;
	int prioridade;
	int numRecursoEncontra;
	bool entrou;
	int tempoR1;
};

// Protótipos de Funções

void readFile();
void criarRecursos(int qtdRecursosCriar);
int numRecurso(int tiraRecurso1, int tiraRecurso2);
```

```
int valorPrioridade();
int probSair();
void userSai(struct utilizador user, int num);
struct utilizador criaUser();
void * recursos(void * pointer);
int tempoRecurso();
void infoSend(int socketfd, int state, int numUtilizadores, int qtdUtilizadores1, int
qtdUtilizadores2, int qtdUtilizadores3, int qtdUtilizadores4);
int criaSocket();
void semInit();
void mutexesInit();
void valueInit();
void mudaOuSai(struct utilizador user, int lugar);
void entrarRec1(struct utilizador user);
void rec1(struct utilizador user);
void entrarRec2(struct utilizador user);
void rec2(struct utilizador user);
void entrarRec3(struct utilizador user);
void rec3(struct utilizador user);
void entrarRec4(struct utilizador user);
void rec4(struct utilizador user);
void filaPrio_Aquaparque(struct utilizador user);
void filaNaoPrio_Aquaparque(struct utilizador user);
int getValue(sem_t *sem);
void * users(void * pointer);
void sim(int socketfd);
```

Ficheiro “monitor.c”:

```
#include "util.h"

// Variável
bool fimSimulacao = false;
bool monitorAtivo = false;

// Recebe mensagens do cliente através do socket, decodifica e escreve informações
formatadas no ficheiro de output e na consola
void decodMessage (int newsocket)
```

```
{
	int messageArrived = 0;
	int state = 0;
	int numUtilizadores = 0;
	int tmp = 0;
	int qtdUtilizadores1 = 0;
	int qtdUtilizadores2 = 0;
	int qtdUtilizadores3 = 0;
	int qtdUtilizadores4 = 0;

	FILE* ficheiroSaida = fopen("saida.txt", "a");

	if(ficheiroSaida == NULL)
	{
		printf("Ocorreu um erro ao criar um ficheiro.");
	}
	else
	{
		while(state != 20)
		{
			char lineReceive[LINHAMAX + 1];
			messageArrived = recv(newsocket, lineReceive, LINHAMAX, 0);

			sscanf(lineReceive, "%d %d %d %d %d %d", &state,
&numUtilizadores, &qtdUtilizadores1, &qtdUtilizadores2, &qtdUtilizadores3,
&qtdUtilizadores4);

			switch(state)
			{
				case 1:
				{
					fprintf(ficheiroSaida, "Início de simulação\n");
					printf("Início de simulação\n");
					break;
				}
				case 2:
				{
```

```
                    fprintf(ficheiroSaida, "Quantidade de utilizadores:
%d\n", numUtilizadores);
                    printf("Quantidade de utilizadores: %d\n",
numUtilizadores);
                    break;
                }
                case 3:
                {
                    fprintf(ficheiroSaida, "Recurso 1: %d utilizador(es)\n",
qtdUtilizadores1);
                    printf("Recurso 1: %d utilizador(es)\n",
qtdUtilizadores1);
                    break;
                }
                case 4:
                {
                    fprintf(ficheiroSaida, "Recurso 2: %d utilizador(es)\n",
qtdUtilizadores2);
                    printf("Recurso 2: %d utilizador(es)\n",
qtdUtilizadores2);
                    break;
                }
                case 5:
                {
                    fprintf(ficheiroSaida, "Recurso 3: %d utilizador(es)\n",
qtdUtilizadores3);
                    printf("Recurso 3: %d utilizador(es)\n",
qtdUtilizadores3);
                    break;
                }
                case 6:
                {
                    fprintf(ficheiroSaida, "Recurso 4: %d utilizador(es)\n",
qtdUtilizadores4);
                    printf("Recurso 4: %d utilizador(es)\n",
qtdUtilizadores4);
                    break;
```

```
                                }
                        }
                }
        }
        if(state == 20)
        {
                fprintf(ficheiroSaida, "Término da simulação.\n");
                printf("Término da simulação.\n");
                fclose(ficheiroSaida);
        }
}

//Inicia o servidor, aguardando conexões do cliente e criando uma thread para decodificar
mensagens do cliente
void simInit()
{

        int socketfd, newsocket, lenClient, childpid, lenServer;
        struct sockaddr_un addrClient, addrServer;

        if ((socketfd = socket(AF_UNIX,SOCK_STREAM, 0)) < 0)
        {
                perror ("Erro ao abrir o socket do server.");
        }

        bzero((char*)&addrServer, sizeof(addrServer));
        addrServer.sun_family = AF_UNIX;
        strcpy(addrServer.sun_path, UNIXSTR_PATH);
        lenServer = strlen(addrServer.sun_path) + sizeof(addrServer.sun_family);
        unlink(UNIXSTR_PATH);

        if(bind(socketfd, (struct sockaddr *) &addrServer, lenServer) < 0)
        {
                perror("Não é possível a conexão do socket ao endereço específico.");
        }

        if(monitorAtivo == false)
```

```
	{
			printf(WHT "Esperando pelo monitor\n" RESET);
			monitorAtivo = true;
	}

	printf("Esperando pelo simulador\n");
	listen (socketfd, 1);

	lenClient = sizeof(addrClient);
	newsocket = accept (socketfd, (struct sockaddr *) &addrClient, &lenClient);
	if(newsocket < 0)
	{
		perror ("Erro ao criar o socket.");
	}

	if((childpid = fork()) < 0)
	{
		perror("Erro ao criar o processo filho.");
	}
	else if(childpid == 0)
	{
		close(socketfd);
		decodMessage(newsocket);
		exit(0);
	}
	close(newsocket);
}

//Apresenta um menu para iniciar a simulação e chama simInit para iniciar o servidor
int main(int argc, char const * argv[])
{
	printf ("Selecione a seguinte opção:\n");
	printf ("1. Começar a simulação.\n");

	int opt = 0;
	while(!fimSimulacao)
	{
```

```
                if(opt == 1)
                {
                        simInit();
                }
                while (opt != 1)
                {
                        printf("Seleciona a opção anterior!\n");
                        scanf("%d", &opt);
                }
        }
        return 0;
}
```

Ficheiro “simulador.c”:

```
#include "util.h"

// Socket
int socketfd = 0;

// Inicialização da struct global
struct aquaparque meuAquaparque;

// Declaração de semáforos para controlar o acesso a recursos compartilhados.
sem_t infoSendT;                    // Controlar o número de mensagens enviadas por vez

sem_t semFilaNP_Aquaparque;         // Controlar o número de pessoas não prioritárias na fila do aquaparque
sem_t semFilaP_Aquaparque;          // Controlar o número de pessoas prioritárias na fila do aquaparque
sem_t semEsperaNP_Aquaparque;       // Bloquear os processos não prioritários quando houver processos prioritários na fila prioritária

sem_t semNumUsersAquaparque;        // Controlar o número de pessoas que entram no aquaparque

sem_t semNumUsersRec1;              // Controlar o número de pessoas que entram no Recurso 1
```

```
sem_t semNumUsersRec2;              // Controlar o número de pessoas que entram no
Recurso 2
sem_t semNumUsersRec3;              // Controlar o número de pessoas que entram no
Recurso 3
sem_t semNumUsersRec4;              // Controlar o número de pessoas que entram no
Recurso 4

sem_t semFilaRec1;                  // Controlar o número de pessoas na fila do Recurso 1
sem_t semEsperaFilaRec1;            // Bloquear os utilizadores quando houver utilizadores
na fila do Recurso 1

sem_t semFilaRec2;                  // Controlar o número de pessoas na fila do Recurso 1
sem_t semEsperaFilaRec2;            // Bloquear os utilizadores quando houver utilizadores
na fila do Recurso 2

sem_t semFilaRec3;                  // Controlar o número de pessoas na fila do Recurso 3
sem_t semEsperaFilaRec3;            // Bloquear os utilizadores quando houver utilizadores
na fila do Recurso 3

sem_t semFilaRec4;                  // Controlar o número de pessoas na fila do Recurso 4
sem_t semEsperaFilaRec4;            // Bloquear os utilizadores quando houver utilizadores
na fila do Recurso 4

// Declaração de trincos (mutexes) para garantir acesso exclusivo a seções críticas do
código.
pthread_mutex_t trincoUser;         // Controlar a criação da pessoa
pthread_mutex_t order;              // Controlar a entrada de cada pessoa no aquaparque
pthread_mutex_t orderP;             // Controlar a entrada de cada pessoa na fila prioritária
do aquaparque
pthread_mutex_t orderNP;            // Controlar a entrada de cada pessoa na fila NÃO
prioritária do aquaparque
pthread_mutex_t trincoNumUsers;     // Controlar o número de utilizadores no aquaparque
pthread_mutex_t trincoNumUsersEntrada;  // Controlar a entrada de utilizadores no
aquaparque

pthread_mutex_t trincoUserRec1;     // Controlar a colocação da pessoa no Recurso 1
```

```
pthread_mutex_t trincoUserFilaRec1;     // Controlar a colocação da pessoa na Fila do Recurso 1

pthread_mutex_t trincoUserRec2;         // Controlar a colocação da pessoa no Recurso 1
pthread_mutex_t trincoUserFilaRec2;     // Controlar a colocação da pessoa na Fila do Recurso 1

pthread_mutex_t trincoUserRec3;         // Controlar a colocação da pessoa no Recurso 3
pthread_mutex_t trincoUserFilaRec3;     // Controlar a colocação da pessoa na Fila do Recurso 3

pthread_mutex_t trincoUserRec4;         // Controlar a colocação da pessoa no Recurso 4
pthread_mutex_t trincoUserFilaRec4;     // Controlar a colocação da pessoa na Fila do Recurso 4

pthread_mutex_t trincoDuracaoRec1;      // Controlar a duração da pessoa no Recurso 1
pthread_mutex_t trincoDuracaoRec2;      // Controlar a duração da pessoa no Recurso 2
pthread_mutex_t trincoDuracaoRec3;      // Controlar a duração da pessoa no Recurso 3
pthread_mutex_t trincoDuracaoRec4;      // Controlar a duração da pessoa no Recurso 4

// Declaração de identificadores de thread para recursos e utilizadores.
pthread_t tIdRecursos[4];
pthread_t tIdUtilizadores[200];

// Declaração de variáveis que serão lidas a partir do arquivo de configuração.
int numMaxUtilizadoresCriar = 0;        // 50
int numMaxUtilizadoresRec1 = 0;         // 10
int numMaxFilaRecurso1 = 0;             // 10
int numMaxUtilizadoresRec2 = 0;         // 5
int numMaxFilaRecurso2 = 0;             // 10
int numMaxUtilizadoresRec3 = 0;         // 3
int numMaxFilaRecurso3 = 0;             // 10
int numMaxUtilizadoresRec4 = 0;         // 8
int numMaxFilaRecurso4 = 0;             // 10
int quantidadeRecursos = 0;             // 4
int maximoFila = 0;                     // 10
```

```
// Declaração de variáveis globais utilizadas na simulação.
int idUser = 0;                    // Inicia a 0
int tempoR1 = 0;                   // Vai ser aleatório para cada utilizador
int tempoR2 = 5;
int tempoR3 = 10;
int tempoR4 = 5;

int vagasAquaparque = 0;           // = numMaxUtilizadores = 20

int vagasRec1 = 0;                 // = numMaxUtilizadoresRec1 = 10
int vagasFilaRec1 = 0;             // = numMaxFilaRecurso1 = 10

int vagasRec2 = 0;                 // = numMaxUsersRec2 = 5
int vagasFilaRec2 = 0;             // = numMaxFilaRec2 = 10

int vagasRec3 = 0;                 // = numMaxUtilizadoresRec3 = 3
int vagasFilaRec3 = 0;             // = numMaxFilaRecurso3 = 10

int vagasRec4 = 0;                 // = numMaxUtilizadoresRec4 = 8
int vagasFilaRec4 = 0;             // = numMaxFilaRecurso4 = 10

int filaP_Aquaparque = 0;          // = maximoFila = 10
int numUsersP = 0;

int filaNP_Aquaparque = 0;         // = maximoFila = 10
int numUsersNP = 0;

int j = 0;

// Variáveis que serão enviadas para o monitor
int numUsers = 0;                  // Inicia a 0
int numUsers1 = 0;                 // Inicia a 0
int numUsers2 = 0;                 // Inicia a 0
int numUsers3 = 0;                 // Inicia a 0
int numUsers4 = 0;                 // Inicia a 0
```

```
// Função para ler as configurações do arquivo "configbase.txt" e inicializar as variáveis globais.
void readFile()
{
    FILE* configuracao = fopen("configbase.txt", "r");

    if(configuracao != NULL)
    {
        char line[50];
        int value;
        char param[50];

        while(fgets(line, sizeof(line), configuracao) != NULL)
        {
            sscanf(line, "%s : %d", param , &value);

            if(strcmp(param, "numMaxUtilizadores") == 0)
            {
                vagasAquaparque = value;
            }
            if(strcmp(param, "numMaxUtilizadoresCriar") == 0)
            {
                numMaxUtilizadoresCriar = value;
            }
            if(strcmp(param, "numMaxUsersRec1") == 0)
            {
                numMaxUtilizadoresRec1 = value;
                vagasRec1 = numMaxUtilizadoresRec1;
            }
            if(strcmp(param, "numMaxFilaRec1") == 0)
            {
                numMaxFilaRecurso1 = value;
                vagasFilaRec1 = numMaxFilaRecurso1;
            }
            if(strcmp(param, "numMaxUsersRec2") == 0)
            {
                numMaxUtilizadoresRec2 = value;
```

```
    vagasRec2 = numMaxUtilizadoresRec2;
}
if(strcmp(param, "numMaxFilaRec2") == 0)
{
    numMaxFilaRecurso2 = value;
    vagasFilaRec2 = numMaxFilaRecurso2;
}
if(strcmp(param, "numMaxUsersRec3") == 0)
{
    numMaxUtilizadoresRec3 = value;
    vagasRec3 = numMaxUtilizadoresRec3;
}
if(strcmp(param, "numMaxFilaRec3") == 0)
{
    numMaxFilaRecurso3 = value;
    vagasFilaRec3 = numMaxFilaRecurso3;
}
if(strcmp(param, "numMaxUsersRec4") == 0)
{
    numMaxUtilizadoresRec4 = value;
    vagasRec4 = numMaxUtilizadoresRec4;
}
if(strcmp(param, "numMaxFilaRec4") == 0)
{
    numMaxFilaRecurso4 = value;
    vagasFilaRec4 = numMaxFilaRecurso4;
}
if(strcmp(param, "numRecursos") == 0)
{
    quantidadeRecursos = value;
}
if(strcmp(param, "maxFilaAquaparque") == 0)
{
    maximoFila = value;
    filaP_Aquaparque = maximoFila;
    filaNP_Aquaparque = maximoFila;
}
```

```
        }
        fclose(configuracao);
    }
    else
    {
        perror("Erro ao abrir o ficheiro de configuração");
        exit(EXIT_FAILURE);
    }
}

// Função para inicializar a estrutura do aquaparque e seus recursos com base na
quantidade fornecida.
void criarRecursos(int qtdRecursosCriar)
{
    meuAquaparque.qtdRecursos = qtdRecursosCriar;
    meuAquaparque.recursos = malloc(meuAquaparque.qtdRecursos * sizeof(struct
recurso));

    for(int i = 0; i < qtdRecursosCriar; i++)
    {
        meuAquaparque.recursos[i].numRecurso = i + 1;
        meuAquaparque.recursos[i].numUtilizadoresDentro = 0;

        if(i == 0)
        {
            meuAquaparque.recursos[i].numMaxUtilizadoresDoRecurso =
numMaxUtilizadoresRec1;
            meuAquaparque.recursos[i].numUtilizadoresFila = numMaxFilaRecurso1;
        }
        else if(i == 1)
        {
            meuAquaparque.recursos[i].numMaxUtilizadoresDoRecurso =
numMaxUtilizadoresRec2;
            meuAquaparque.recursos[i].numUtilizadoresFila = numMaxFilaRecurso2;
        }
        else if(i == 2)
        {
```

```
            meuAquaparque.recursos[i].numMaxUtilizadoresDoRecurso =
numMaxUtilizadoresRec3;
            meuAquaparque.recursos[i].numUtilizadoresFila = numMaxFilaRecurso3;
        }
        else if(i == 3)
        {
            meuAquaparque.recursos[i].numMaxUtilizadoresDoRecurso =
numMaxUtilizadoresRec4;
            meuAquaparque.recursos[i].numUtilizadoresFila = numMaxFilaRecurso4;
        }
    }
}

// Função para determinar um número de recurso aleatório, removendo os valores
especificados por tiraRecurso1 e tiraRecurso2.
int numRecurso(int tiraRecurso1, int tiraRecurso2)
{
    int valRecurso[5] = {1, 2, 3, 4, 5};

    // Retira o valor de tiraRecurso1 do array valRecurso
    for (int i = 0; i < 5; ++i)
    {
        if (valRecurso[i] == tiraRecurso1)
        {
            for (int j = i; j < 4; ++j)
            {
                valRecurso[j] = valRecurso[j + 1];
            }
            break;
        }
    }

    // Retira o valor de tiraRecurso2 do array valRecurso
    for (int i = 0; i < 4; ++i)
    {
        if (valRecurso[i] == tiraRecurso2)
        {
```

```
            for (int j = i; j < 3; ++j)
            {
                valRecurso[j] = valRecurso[j + 1];
            }
            break;
        }
    }

    int numRemocoes = (tiraRecurso1 != 0) + (tiraRecurso2 != 0); // Conta o número de retiradas
    int valRecRandom = valRecurso[rand() % (5 - numRemocoes)];
    return valRecRandom;
}

// Função para determinar o valor de prioridade de um recurso com base no número do recurso.
int valorPrioridade()
{
    int valorPrioridade[2] = {0, 1};
    int valorPrioRandom = valorPrioridade[rand() % 2];
    return valorPrioRandom;
}

// Função para simular a probabilidade de um utilizador sair.
int probSair()
{
    return rand() % 3;
}

// Função para que um utilizador saia do aquaparque ou escolha um novo recurso.
void userSai(struct utilizador user, int num)
{
    int numProbSair = probSair();

    if(numProbSair == 0)
    {
        user.numRecursoEncontra = 5;
```

```
        mudaOuSai(user, 5);
    }
    else
    {
        user.numRecursoEncontra = numRecurso(num, 5);
        int valor = user.numRecursoEncontra;
        mudaOuSai(user, valor);
    }
}

// Cria um novo utilizador com id, prioridade, número de recurso, estado de se já entrou no
recurso ou não.
struct utilizador criaUser()
{
    pthread_mutex_lock(&trincoUser);

    struct utilizador user;
    user.idUtilizador = idUser;

    user.prioridade = valorPrioridade();

    user.entrou = false;
    user.tempoR1 = tempoRecurso();

    user.numRecursoEncontra = numRecurso(0, 5);

    idUser++;

    pthread_mutex_unlock(&trincoUser);
    return user;
}

// Imprime informações sobre os recursos na consola.
void * recursos(void * pointer)
{
    int *numeroRecurso = (int*) pointer;
    struct recurso *recursosNovos = &(meuAquaparque.recursos[*numeroRecurso]);
```

```
    printf(WHT "Recurso %d: Pode conter, no máximo, %2d pessoas dentro e %2d pessoas
na fila.\n" RESET, recursosNovos->numRecurso,
recursosNovos->numMaxUtilizadoresDoRecurso, recursosNovos->numUtilizadoresFila);
}

// Retorna um valor de tempo aleatório para simular o tempo de uso do recurso.
int tempoRecurso()
{
    return rand() % 16 + 5;
}

// Envia informações formatadas sobre o estado da simulação, o número de utilizadores
totais, dos recursos 1, 2, 3 e 4 para o servidor através de sockets
void infoSend(int socketfd, int state, int numUtilizadores, int qtdUtilizadores1, int
qtdUtilizadores2, int qtdUtilizadores3, int qtdUtilizadores4)
{
    sem_wait(&infoSendT);

    char buffer[LINHAMAX];
    int lenMsg = 0;

    sprintf(buffer, "%d %d %d %d %d %d", state, numUtilizadores, qtdUtilizadores1,
qtdUtilizadores2, qtdUtilizadores3, qtdUtilizadores4);
    lenMsg = strlen(buffer) + 1;

    if(send(socketfd, buffer, lenMsg, 0) != lenMsg)
    {
        perror("Ocorreu um erro ao enviar os dados\n");
    }
    sleep(1);
    sem_post(&infoSendT);
}

// Cria um socket Unix e conecta ao servidor, retornando o descritor do socket.
int criaSocket()
{
```

```
    struct sockaddr_un addrServer;
    int lenServer;

    if ((socketfd = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)
    {
        perror(" Ocorreu um erro ao criar um socket\n");
    }

    bzero((char*)&addrServer, sizeof(addrServer));

    addrServer.sun_family = AF_UNIX;
    strcpy(addrServer.sun_path, UNIXSTR_PATH);
    lenServer = strlen(addrServer.sun_path) + sizeof(addrServer.sun_family);

    bool simWorking = false;
    while (connect(socketfd, (struct sockaddr *) &addrServer, lenServer) < 0)
    {
        if(simWorking == false)
        {
            printf(WHT "Esperando pelo monitor\n" RESET);
            simWorking = true;
        }
    }

    if(socketfd < 0)
    {
        perror ("Erro aceite");
    }
    return socketfd;
}

// Inicializa os semáforos.
void semInit()
{
    sem_init(&infoSendT, 0, 1);
    sem_init(&semFilaNP_Aquaparque, 0, MAX_FILA_AQUAPARQUE);
    sem_init(&semFilaP_Aquaparque, 0, MAX_FILA_AQUAPARQUE);
```

```
    sem_init(&semEsperaNP_Aquaparque, 0, 1);

    sem_init(&semNumUsersAquaparque, 0, MAX_USERS_AQUAPARQUE);

    sem_init(&semNumUsersRec1, 0, MAX_RECURSO1);
    sem_init(&semNumUsersRec2, 0, MAX_RECURSO2);
    sem_init(&semNumUsersRec3, 0, MAX_RECURSO3);
    sem_init(&semNumUsersRec4, 0, MAX_RECURSO4);

    sem_init(&semEsperaFilaRec1, 0, 1);
    sem_init(&semFilaRec1, 0, MAX_FILA_RECURSO1);

    sem_init(&semEsperaFilaRec2, 0, 1);
    sem_init(&semFilaRec2, 0, MAX_FILA_RECURSO2);

    sem_init(&semEsperaFilaRec3, 0, 1);
    sem_init(&semFilaRec3, 0, MAX_FILA_RECURSO3);

    sem_init(&semEsperaFilaRec4, 0, 1);
    sem_init(&semFilaRec4, 0, MAX_FILA_RECURSO4);
}

// Inicializa os mutexes.
void mutexesInit()
{
    pthread_mutex_init(&trincoUser, NULL);
    pthread_mutex_init(&order, NULL);
    pthread_mutex_init(&orderP, NULL);
    pthread_mutex_init(&orderNP, NULL);
    pthread_mutex_init(&trincoNumUsers, NULL);
    pthread_mutex_init(&trincoNumUsersEntrada, NULL);
    pthread_mutex_init(&trincoUserRec1, NULL);
    pthread_mutex_init(&trincoUserFilaRec1, NULL);
    pthread_mutex_init(&trincoUserRec2, NULL);
    pthread_mutex_init(&trincoUserFilaRec2, NULL);
    pthread_mutex_init(&trincoUserRec3, NULL);
    pthread_mutex_init(&trincoUserFilaRec3, NULL);
```

```
    pthread_mutex_init(&trincoUserRec4, NULL);
    pthread_mutex_init(&trincoUserFilaRec4, NULL);
    pthread_mutex_init(&trincoDuracaoRec1, NULL);
    pthread_mutex_init(&trincoDuracaoRec2, NULL);
    pthread_mutex_init(&trincoDuracaoRec3, NULL);
    pthread_mutex_init(&trincoDuracaoRec4, NULL);
}

// Inicializa valores dos semáforos e faz a leitura das configurações base.
void valueInit()
{
    srand((unsigned)time(NULL));

    semInit();

    mutexesInit();

    readFile();
    printf(WHT "\nLeitura efetuada.\n" RESET);
    criarRecursos(4);
}

// Função principal para mudar ou sair do recurso atual do utilizador.
void mudaOuSai(struct utilizador user, int lugar)
{
    user.numRecursoEncontra = lugar;
    if(user.numRecursoEncontra == 1)
    {
        rec1(user);
    }
    else if(user.numRecursoEncontra == 2)
    {
        rec2(user);
    }
    else if(user.numRecursoEncontra == 3)
    {
        rec3(user);
```

```
    }
    else if(user.numRecursoEncontra == 4)
    {
        rec4(user);
    }
    else if(user.numRecursoEncontra == 5)
    {
        pthread_mutex_lock(&trincoNumUsers);
        sem_post(&semNumUsersAquaparque);
        numUsers--;
        vagasAquaparque++;
        int semValuer = getValue(&semNumUsersAquaparque);
        printf(MAG "                                                                                    O
utilizador %2d saiu do Aquaparque\n" RESET, user.idUtilizador);
        infoSend(socketfd, 2, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
        pthread_mutex_unlock(&trincoNumUsers);

        pthread_exit(NULL);
    }
    else
    {
        printf(RED "Erro: O recurso do user %2d está a %2d\n" RESET, user.idUtilizador,
user.numRecursoEncontra);
    }
}

// Função para entrada no recurso 1, levando em consideração prioridade e condições do
recurso 1.
void entrarRec1(struct utilizador user)
{
    pthread_mutex_lock(&trincoUserRec1);

    numUsers1++;
    vagasRec1--;
    printf(CYN "                              O utilizador %2d entrou na piscina\n" RESET,
user.idUtilizador);
    infoSend(socketfd, 3, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
```

```
    pthread_mutex_unlock(&trincoUserRec1);

    pthread_mutex_lock(&trincoDuracaoRec1);
    sleep(user.tempoR1);
    pthread_mutex_unlock(&trincoDuracaoRec1);

    pthread_mutex_lock(&trincoUserRec1);
    numUsers1--;
    vagasRec1++;
    printf(MAG "                                                    O utilizador %2d vai sair da
piscina\n" RESET, user.idUtilizador);
    infoSend(socketfd, 3, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec1);

    sem_post(&semNumUsersRec1);
}

// Função principal que simula a entrada ou não do utilizador no recurso 1.
void rec1(struct utilizador user)
{
    if (vagasRec1 > 0 && numUsers1 < numMaxUtilizadoresRec1 && vagasFilaRec1 ==
numMaxFilaRecurso1)
    {
        sem_wait(&semNumUsersRec1);
        entrarRec1(user);
        userSai(user, 0);
    }
    else if(vagasRec1 == 0 || numUsers1 >= numMaxUtilizadoresRec1 || vagasFilaRec1 <=
numMaxFilaRecurso1)
    {
        if(vagasFilaRec1 != 0)
        {
            sem_wait(&semFilaRec1);
            vagasFilaRec1--;

            printf(YEL "                          Entrada do utilizador %2d na fila da piscina\n"
RESET, user.idUtilizador);
```

```
            sem_wait(&semEsperaFilaRec1);
            sem_wait(&semNumUsersRec1);

            sem_post(&semFilaRec1);
            vagasFilaRec1++;

            sem_post(&semEsperaFilaRec1);
            entrarRec1(user);

            userSai(user, 0);
        }
        else if(vagasFilaRec1 == 0)
        {
            printf(BLU "A piscina e a sua fila estão cheios\n" RESET);
            userSai(user, 1);
        }
        else
        {
            printf(RED "Erro: As vagas da fila da piscina estão a %2d\n" RESET,
vagasFilaRec1);
        }
    }
    else
    {
        printf(RED "Erro: As vagas da piscina está a %2d e a quantidade de users na piscina
está a %2d\n" RESET, vagasRec1, numUsers1);
    }
}

// Função para entrada no recurso 2, levando em consideração prioridade e condições do
recurso 2.
void entrarRec2(struct utilizador user)
{
    pthread_mutex_lock(&trincoUserRec2);

    numUsers2++;
```

```
    vagasRec2--;
    printf(CYN "                    O utilizador %2d entrou no tobogã\n" RESET,
user.idUtilizador);
    infoSend(socketfd, 4, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec2);

    pthread_mutex_lock(&trincoDuracaoRec2);
    sleep(tempoR2);
    pthread_mutex_unlock(&trincoDuracaoRec2);

    pthread_mutex_lock(&trincoUserRec2);
    numUsers2--;
    vagasRec2++;
    printf(MAG "                                    O utilizador %2d vai sair do
tobogã\n" RESET, user.idUtilizador);
    infoSend(socketfd, 4, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec2);

    sem_post(&semNumUsersRec2);
}

// Função principal que simula a entrada ou não do utilizador no recurso 2.
void rec2(struct utilizador user)
{
    if (vagasRec2 > 0 && numUsers2 < numMaxUtilizadoresRec2 && vagasFilaRec2 ==
numMaxFilaRecurso2)
    {
        sem_wait(&semNumUsersRec2);
        entrarRec2(user);
        userSai(user, 0);
    }
    else if(vagasRec2 == 0 || numUsers2 >= numMaxUtilizadoresRec2 || vagasFilaRec2 <=
numMaxFilaRecurso2)
    {
        if(vagasFilaRec2 != 0)
        {
            sem_wait(&semFilaRec2);
```

```
            vagasFilaRec2--;

            printf(YEL "                    Entrada do utilizador %2d na fila do tobogã\n"
RESET, user.idUtilizador);

            sem_wait(&semEsperaFilaRec2);
            sem_wait(&semNumUsersRec2);

            sem_post(&semFilaRec2);
            vagasFilaRec2++;

            sem_post(&semEsperaFilaRec2);

            entrarRec2(user);

            userSai(user, 0);
        }
        else if(vagasFilaRec2 == 0)
        {
            printf(BLU "O tobogã e a sua fila estão cheios\n" RESET);
            userSai(user, 2);
        }
        else
        {
            printf(RED "Erro: As vagas da fila do tobogã estão a %2d\n" RESET,
vagasFilaRec2);
        }
    }
    else
    {
        printf(RED "Erro: As vagas do tobogã está a %2d e a quantidade de users no tobogã
está a %2d\n" RESET, vagasRec2, numUsers2);
    }
}

// Função para entrada no recurso 3, levando em consideração prioridade e condições do
recurso 3.
```

```
void entrarRec3(struct utilizador user)
{
    pthread_mutex_lock(&trincoUserRec3);

    numUsers3++;
    vagasRec3--;
    printf(CYN "                    O utilizador %2d entrou no escorrega\n" RESET,
user.idUtilizador);
    infoSend(socketfd, 5, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec3);

    pthread_mutex_lock(&trincoDuracaoRec3);
    sleep(tempoR3);
    pthread_mutex_unlock(&trincoDuracaoRec3);

    pthread_mutex_lock(&trincoUserRec3);
    numUsers3--;
    vagasRec3++;
    printf(MAG "                                        O utilizador %2d vai sair do
escorrega\n" RESET, user.idUtilizador);
    infoSend(socketfd, 5, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec3);

    sem_post(&semNumUsersRec3);
}

// Função principal que simula a entrada ou não do utilizador no recurso 3.
void rec3(struct utilizador user)
{
    if (vagasRec3 > 0 && numUsers3 < numMaxUtilizadoresRec3 && vagasFilaRec3 ==
numMaxFilaRecurso3)
    {
        sem_wait(&semNumUsersRec3);
        entrarRec3(user);
        userSai(user, 0);
    }
```

```
   else if(vagasRec3 == 0 || numUsers3 >= numMaxUtilizadoresRec3 || vagasFilaRec3 <=
numMaxFilaRecurso3)
   {
      if(vagasFilaRec3 != 0)
      {
         sem_wait(&semFilaRec3);
         vagasFilaRec3--;

         printf(YEL "                    Entrada do utilizador %2d na fila do escorrega\n"
RESET, user.idUtilizador);

         sem_wait(&semEsperaFilaRec3);
         sem_wait(&semNumUsersRec3);

         sem_post(&semFilaRec3);
         vagasFilaRec3++;

         sem_post(&semEsperaFilaRec3);

         entrarRec3(user);

         userSai(user, 0);
      }
      else if(vagasFilaRec3 == 0)
      {
         printf(BLU "O escorrega e a sua fila estão cheios\n" RESET);
         userSai(user, 3);
      }
      else
      {
         printf(RED "Erro: As vagas da fila do escorrega estão a %2d\n" RESET,
vagasFilaRec3);
      }
   }
   else
   {
```

```
        printf(RED "Erro: As vagas do escorrega está a %2d e a quantidade de users no escorrega está a %2d\n" RESET, vagasRec3, numUsers3);
    }
}

// Função para entrada no recurso 4, levando em consideração prioridade e condições do recurso 4.
void entrarRec4(struct utilizador user)
{
    pthread_mutex_lock(&trincoUserRec4);

    numUsers4++;
    vagasRec4--;
    printf(CYN "                    O utilizador %2d entrou no snack-bar\n" RESET, user.idUtilizador);
    infoSend(socketfd, 6, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec4);

    pthread_mutex_lock(&trincoDuracaoRec4);
    sleep(tempoR4);
    pthread_mutex_unlock(&trincoDuracaoRec4);

    pthread_mutex_lock(&trincoUserRec4);
    numUsers4--;
    vagasRec4++;
    printf(MAG "                                    O utilizador %2d vai sair do snack-bar\n" RESET, user.idUtilizador);
    infoSend(socketfd, 6, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    pthread_mutex_unlock(&trincoUserRec4);

    sem_post(&semNumUsersRec4);
}

// Função principal que simula a entrada ou não do utilizador no recurso 4.
void rec4(struct utilizador user)
{
```

```
   if (vagasRec4 > 0 && numUsers4 < numMaxUtilizadoresRec4 && vagasFilaRec4 ==
numMaxFilaRecurso4)
   { // O recurso tem vagas e o user entra
      sem_wait(&semNumUsersRec4);
      entrarRec4(user);

      userSai(user, 0);
   }
   else if(vagasRec4 == 0 || numUsers4 >= numMaxUtilizadoresRec4 || vagasFilaRec4 <=
numMaxFilaRecurso4)
   { // O recurso não tem vagas
      if(vagasFilaRec4 != 0) // A fila do recurso tem vagas
      {
         sem_wait(&semFilaRec4);
         vagasFilaRec4--;

         printf(YEL "                    Entrada do utilizador %2d na fila do snack-bar\n"
RESET, user.idUtilizador);

         sem_wait(&semEsperaFilaRec4);
         sem_wait(&semNumUsersRec4);

         sem_post(&semFilaRec4);
         vagasFilaRec4++;

         sem_post(&semEsperaFilaRec4);

         entrarRec4(user);

         userSai(user, 0);
      }
      else if(vagasFilaRec4 == 0)
      {
         printf(BLU "O snack-bar e a sua fila estão cheios\n" RESET);
         userSai(user, 4);
      }
      else
```

```
        {
            printf(RED "Erro: As vagas da fila do snack-bar estão a %2d\n" RESET,
vagasFilaRec4);
        }
    }
    else
    {
        printf(RED "Erro: As vagas do snack-bar está a %2d e a quantidade de users no
snack-bar está a %2d\n" RESET, vagasRec4, numUsers4);
    }
}

// Simula a entrada na fila do parque aquático para utilizadores prioritários.
void filaPrio_Aquaparque(struct utilizador user)
{
    pthread_mutex_unlock(&orderP);
    printf(YEL "  O utilizador %2d chegou à fila prioritária do Aquaparque\n" RESET,
user.idUtilizador);
    sem_wait(&semFilaP_Aquaparque);
    numUsersP++;

    if(j == 0)
    {
        sem_wait(&semEsperaNP_Aquaparque);
        j++;
    }

    sem_wait(&semNumUsersAquaparque);
    printf(YEL "    O utilizador prioritário %2d vai entrar no Aquaparque\n" RESET,
user.idUtilizador);
    sem_post(&semNumUsersAquaparque);
    sem_post(&semFilaP_Aquaparque);
    numUsersP--;
}

// Simula a entrada na fila do parque aquático para utilizadores não prioritários.
void filaNaoPrio_Aquaparque(struct utilizador user)
```

```
{
    pthread_mutex_unlock(&orderNP);
    printf(YEL "    O utilizador %2d chegou à fila NÃO prioritária do Aquaparque\n" RESET,
user.idUtilizador);
    sem_wait(&semFilaNP_Aquaparque);
    numUsersNP++;

    sem_wait(&semNumUsersAquaparque);
    printf(YEL "     O utilizador NÃO prioritário %2d cede a entrada no Aquaparque aos
prioritários\n" RESET, user.idUtilizador);
    sem_post(&semNumUsersAquaparque);
    sem_wait(&semEsperaNP_Aquaparque);

    printf(YEL "    O user NÃO prioritário %2d vai entrar no Aquaparque\n" RESET,
user.idUtilizador);
    sem_post(&semFilaNP_Aquaparque);
    numUsersNP--;
}

// Obtém o valor atual de um semáforo.
int getValue(sem_t *sem)
{
    int valor;
    sem_getvalue(sem, &valor);
    return valor;
}

// Função principal executada por cada thread de utilizador.
void * users(void * pointer)
{
    struct utilizador user = criaUser();

    pthread_mutex_lock(&order);

    if(vagasAquaparque == 0)
    {
        if(user.prioridade == 1)
```

```
    {
        if(numUsersP >= maximoFila)
        {
            printf(RED "O user %2d é prioritário, mas a fila prioritária do Aquaparque está
cheia, logo ele desistiu\n" RESET, user.idUtilizador);

            pthread_mutex_unlock(&order);

            pthread_exit(NULL);
        }
        else if(numUsersP < maximoFila && numUsersP >= 0)
        {
            pthread_mutex_unlock(&order);
            pthread_mutex_lock(&orderP);
            filaPrio_Aquaparque(user);
            pthread_mutex_lock(&order);
        }
        else
        {
            printf(RED "Erro: A fila prioritária do Aquaparque está com %2d vagas\n" RESET,
maximoFila - numUsersP);
        }
    }
    else if(user.prioridade == 0)
    {
        if(numUsersNP >= maximoFila)
        {
            printf(RED "O user %2d é NÃO prioritário, mas a fila NÃO prioritária do
Aquaparque está cheia, logo ele desistiu\n" RESET, user.idUtilizador);

            pthread_mutex_unlock(&order);

            pthread_exit(NULL);
        }
        else if (numUsersNP < maximoFila && numUsersNP >= 0)
        {
            pthread_mutex_unlock(&order);
```

```
            pthread_mutex_lock(&orderNP);
            filaNaoPrio_Aquaparque(user);
            pthread_mutex_lock(&order);
        }
        else
        {
            printf(RED "Erro: A fila NÃO prioritária do Aquaparque está com %2d vagas\n"
RESET, maximoFila - numUsersNP);
        }
    }
    else
    {
        printf(RED "Erro: O user %2d tem a prioridade a %d\n" RESET, user.idUtilizador,
user.prioridade);
    }
}

pthread_mutex_lock(&trincoUser);

if (!user.entrou)
{
    pthread_mutex_lock(&trincoNumUsersEntrada);
    sem_wait(&semNumUsersAquaparque);
    vagasAquaparque--;
    numUsers++;
    int semValue = getValue(&semNumUsersAquaparque);
    printf(GRN "O utilizador %2d entrou no Aquaparque\n" RESET, user.idUtilizador);
    infoSend(socketfd, 2, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
    user.entrou = true;
    pthread_mutex_unlock(&order);
    pthread_mutex_unlock(&trincoNumUsersEntrada);
}

pthread_mutex_unlock(&trincoUser);
if(numUsersP == 0 && j == 1)
{
    sem_post(&semEsperaNP_Aquaparque);
```

```
    }
    usleep(50000);  // Evita espera ativa

    mudaOuSai(user, user.numRecursoEncontra);
}

// Inicia a simulação, cria threads para os recursos e utilizadores, e controla o fluxo.
void sim(int socketfd)
{
    valueInit();
    infoSend(socketfd, 1, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);

    int zero = 0;
    int um = 1;
    int dois = 2;
    int tres = 3;

    printf(WHT "\nAquaparque:\n" RESET);
    pthread_create(&tIdRecursos[0], NULL, recursos, &zero);
    pthread_create(&tIdRecursos[1], NULL, recursos, &um);
    pthread_create(&tIdRecursos[2], NULL, recursos, &dois);
    pthread_create(&tIdRecursos[3], NULL, recursos, &tres);

    for(int i = 0; i < quantidadeRecursos; i++)
    {
        pthread_join(tIdRecursos[i], NULL);
    }

    for(int i = 0; i < numMaxUtilizadoresCriar; i++)
    {
        pthread_create(&tIdUtilizadores[i], NULL, users, NULL);
    }

    for(int i = 0; i < numMaxUtilizadoresCriar; i++)
    {
        pthread_join(tIdUtilizadores[i], NULL);
    }
```

```
    infoSend(socketfd, 20, numUsers, numUsers1, numUsers2, numUsers3, numUsers4);
}

// Função principal para inicializar as threads e recursos.
int main(int argc, char*argv[])
{
    srand(time(NULL));
    socketfd = criaSocket();

    sim(socketfd);

    close(socketfd);
    printf(WHT "Simulação terminada\n" RESET);

    free(meuAquaparque.recursos);

    exit(0);
}
```