Universidade da Beira Interior

Grupo:

Mateus Aleixo, nº47700

Pedro Lourenço, nº48213

Samuel Dias, nº48184

Código elaborado:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <signal.h>
#include <string.h>
#include <time.h>

#define STOP_FILE "tokenctl.txt"
```

```c
int stop()
{
    int fd = open(STOP_FILE, O_RDWR);

    if (fd == -1)
    {
        printf("Erro a abrir ficheiro\n");
        exit(1);
    }

    char value;

    if (read(fd, &value, sizeof(char)) == -1)
    {
        printf("Erro a ler ficheiro\n");
        exit(1);
    }

    close(fd);

    return value == '1' ? 1 : 0;
}
```

Função de paragem do Token Ring, onde abrimos o ficheiro tokenctl.txt com todas as precauções necessárias caso não seja possível abrir ou ler o ficheiro. Se o a variável *value* for um o *return* é um, caso contrário o *return* vai ser zero.

```c
int main(int argc, char *argv[])
{
    if (argc != 5)
    {
        printf("Uso: tokenring <n> <p> <t> <max>\n");
        return 1;
    }

    // Número de processos
    int n = atoi(argv[1]);

    // Probabilidade de bloqueio
    float p = atof(argv[2]);

    // Tempo de bloqueio em segundos
    int t = atoi(argv[3]);

    // Valor máximo da token
    int max = atoi(argv[4]);

    int i;
    int pipes[n][2];
    char pipe_name[10];
```

Começa-se por verificar se o input corresponde ao esperado. Se não, o programa acaba após um *print* no terminal a explicar o processo de execução.

Se a execução for feita de forma correta, vão declaradas duas variáveis do tipo *Int* e um *array de char's*. Um i que vai ser auxiliar para qualquer ciclo for que for realizado, as inicializações dos pipes e por fim o *array de char's* que vai servir para dar nome aos pipes.

```c
// Criação dos pipes nomeados
for (i = 0; i < n; i++)
{
    sprintf(pipe_name, "pipe%dto%d", i + 1, (i + 2) > n ? 1 : i + 2);
    mkfifo(pipe_name, 0666);
    pipes[i][0] = open(pipe_name, O_RDONLY | O_NONBLOCK);
    pipes[i][1] = open(pipe_name, O_WRONLY);
}
```

Dá o nome devido e cria os *pipes* necessários usando a função mkfifo e, em seguida, abre os *pipes* para leitura e escrita através das opções disponíveis na função open. O argumento 0666 significa que o *pipes* será criado com permissões de leitura e escrita.

```
    // Criação do ficheiro de paragem
    creat(STOP_FILE, S_IRWXG | S_IRWXO | S_IRWXU);
    int stop_fd = open(STOP_FILE, O_RDWR);

    if (stop_fd == -1)
    {
        printf("Erroao criar ficheiro\n");
        return 1;
    }

    char start_value = '0';

    if (write(stop_fd, &start_value, sizeof(char)) == -1)
    {
        printf("Erro ao escrever no ficheiro\n");
        return 1;
    }

    close(stop_fd);

```

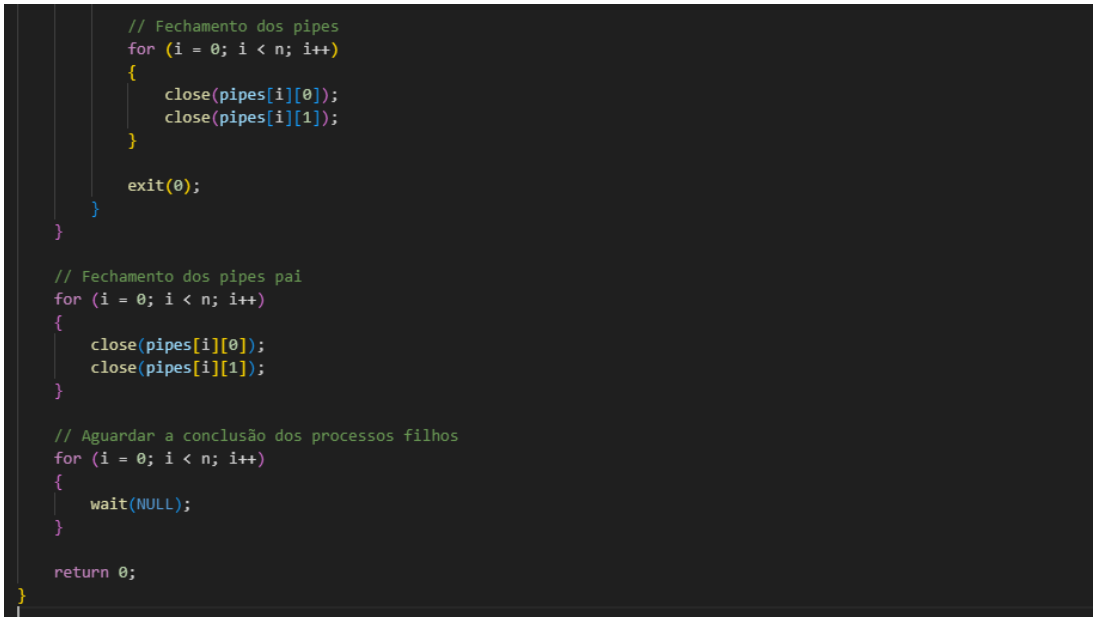
Criação do ficheiro de paragem, onde se verifica se é possível criar o ficheiro. Caso seja possível, o programa vai tentar escrever no ficheiro, lançando uma mensagem de aviso caso não consiga e fecha o ficheiro.

```c
// Criação dos processos
for (i = 0; i < n; i++)
{
    if (fork() == 0)
    {
        srand(getpid());
        int value = 0;
        int token_received = 0;
        while (1)
        {
            if (stop())
                break;
            // Leitura do token
            if (token_received)
            {
                read(pipes[i][0], &value, sizeof(int));
                token_received = 0;
            }

            // Incremento do valor do token
            value++;

            // Verificação do valor máximo do token
            if (value >= max)
                break;

            // Envio do token para o próximo processo
            write(pipes[(i + 1) % n][1], &value, sizeof(int));

            // Verificação da probabilidade de bloqueio
            float rand_value = (float)rand() / RAND_MAX;
            if (rand_value < p)
            {
                printf("[p%d] blocked on token (val = %d)\n", i + 1, value);
                sleep(t);
            }
        }
```

Esta parte do código cria um *loop* que vai criar n processos filho, onde cada processo filho vai ter a sua própria cópia das variáveis e dos *pipes*. Os processos filhos possuem um loop infinito que executa as seguintes etapas:

- Caso a função stop seja chamada, o programa vai automaticamente parar a sua execução
- Se um *token* foi recebido anteriormente, ele vai ser lido e analisado pelo *pipe* correspondente ao processo atual.
- Se o valor do *token* for maior ou igual ao limite máximo (max), o processo atual imprime o *token* no *pipe* que está ligado ao próximo processo e encerra o loop.
- Caso o contrário se verifique, o processo atual escreve o *token* no *pipe* correspondente ao próximo processo.

- Por fim, verifica a possibilidade de bloqueio. Se o valor nos *pipes* for menor que p, o processo atual imprime uma mensagem informando que está bloqueado e aguarda o tempo especificado no momento de execução do programa antes de continuar com a execução.



```
            // Fechamento dos pipes
            for (i = 0; i < n; i++)
            {
                close(pipes[i][0]);
                close(pipes[i][1]);
            }

            exit(0);
        }
    }

    // Fechamento dos pipes pai
    for (i = 0; i < n; i++)
    {
        close(pipes[i][0]);
        close(pipes[i][1]);
    }

    // Aguardar a conclusão dos processos filhos
    for (i = 0; i < n; i++)
    {
        wait(NULL);
    }

    return 0;
}
```

Após sair do loop, os processos filhos vão fechar os *pipes* que lhes estão associados. Ainda importante de referir que o programa ainda tem de fechar os *pipes* associados ao processo pai. Por fim, é usado um ciclo for com o objetivo de aguardar o fim da execução dos processos filhos, antes de encerrar a execução do programa