Sistemas Operativos

Trabalho Prático – Meta 1

Departamento de Engenharia Informática e de Sistemas

Engenharia Informática

Ano letivo 2018/2019

Carolina Oliveira – Nº 21270477 – P7

Cláudia Tavares – Nº 21270617 – P3

Índice

Introdução

O presente relatório tem como objetivo clarificar e explicar todas as decisões tomadas para a realização da Meta 1 do enunciado do trabalho prático da cadeira de Sistemas Operativos.

Estrutura do Projeto

Foi criado um *header file* que contém, portanto, as estruturas acima referidas e as constantes simbólicas que se seguem:

```
#define MAXCHAR 1000
#define TIMEOUT 10
#define TAM 50
#define NLINHAS 15
#define NCOLUNAS 67
```

O objetivo da criação das mesmas foi criar valores específicos comuns para as estruturas criadas.

- MAXCHAR refere-se ao número máximo de caracteres que o corpo da mensagem pode ter.
- TIMEOUT refere-se ao número de segundos que a mensagem vai estar a ser armazenada no sistema.
- TAM refere-se ao número de caracteres que o *username*, tópico e título podem ter.
- NLINHAS refere-se ao número de linhas que o campo de texto pode ter.
- NCOLUNAS refere-se ao número de colunas que o campo de texto pode ter.

As variáveis de ambiente criadas foram:

```
#define MAXMSG nmaxmsg
#define WORDSNOT fileWN
#define MAXNOT nmaxnot
```

- MAXMSG que corresponde ao número máximo de mensagens a armazenar.
- MAXNOT que corresponde ao número máximo de palavras proibidas.
- WORDSNOT que se refere ao nome do ficheiro de texto, no qual estão armazenadas as palavras proibidas.

Além disto, não pode ser esquecido, que teremos também o programa do servidor, do cliente e ainda o verificador. Neste caso, tanto o servidor como o cliente terão acesso ao *header file* referido anteriormente.

Makefile

O ficheiro makefile é constituído por regras de complicação que facilitam esse mesmo processo, como tal foram criadas algumas regras que compilam o programa referido (cliente, gestor, verificador), removem os executáveis (cleang, cleanc, cleang e clean que corresponde à junção das anteriores) e por fim, a regra all que compila todos os programas referidos. Como pode ser visualizado:

```
all: cliente gestor verificador

cliente:
    gcc -c cliente.c
    gcc header.h cliente.c -o cliente


gestor:
    gcc -c servidor.c
    gcc header.h servidor.c -o gestor


verificador:
    gcc -c verificador.c
    gcc verificador.c -o verificador


clean: cleang cleanc cleanv


cleang:
    rm gestor
    rm servidor.o


cleanc:
    rm cliente
    rm cliente.o


cleanv:
    rm verificador
    rm verificador.o
```

Estruturas de dados

utilizador

- username, string de tamanho TAM para guardar o *username* do utilizador;
- IDuser, número inteiro único para cada utilizador;

```
typedef struct utilizador{
    char username[TAM];
    int IDuser;
}user;
```

Estrutura com o propósito de guardar o *username* do utilizador para o mesmo aparecer na consola e em caso de *usernames* repetidos, o mesmo ser alterado. A distinção dos utilizadores será feita a partir de um ID único, atribuído a cada um.

A escolha desta estrutura deve-se principalmente à sua simplicidade de uso, servindo apenas para armazenamento temporário de informação.

servidor

- nclientes, número inteiro que representa o número de clientes ativos;

```
typedef struct servidor{
    int nclientes;
}server;
```

Estrutura responsável por guardar o número de clientes ativos no servidor. A escolha desta estrutura deve-se principalmente à sua simplicidade de uso, servindo apenas para armazenamento temporário de informação.

mensagem

- corpo, string de tamanho MAXCHAR para guardar o corpo da mensagem;
- topico, string de tamanho TAM para guardar o tópico da mensagem;
- titulo, string de tamanho TAM para guardar o título da mensagem;
- duração, número inteiro para guardar o número de segundos de duração da mensagem;

```
typedef struct mensagem{
    char corpo[MAXCHAR];
    char topico[TAM];
    char titulo[TAM];
    int duracao;
}msg;
```

Estrutura responsável por guardar as configurações do número de caracteres para cada parte da mensagem e do número de segundos que a mesma ficará armazenada no sistema.

A escolha desta estrutura deve-se principalmente à sua simplicidade de uso, servindo apenas para armazenamento temporário de informação.

editar

- msg *texto, ponteiro que aponta para a estrutura mensagem que está a ser editada;
- linha, número inteiro para guardar o número da linha que está a ser editada;
- coluna, número inteiro para guardar o número da coluna que está a ser editada;

```
typedef struct editar{
    msg *texto;
    int linha; //y
    int coluna; //x
}edit;
```

Estrutura responsável por guardar o local do cursor na edição de texto. A escolha desta estrutura deve-se principalmente à sua simplicidade de uso, servindo apenas para armazenamento temporário de informação.

pedido

- frase, string de tamanho MAXCHAR para guardar o texto introduzido;
- remetente, número inteiro para guardar o *pid* do cliente;

```
typedef struct pedido{
    char frase[MAXCHAR];
    int remetente;
}PEDIDO;
```

Estrutura responsável por guardar a frase que será enviada do cliente para o servidor, e o pid do seu remetente, para posteriormente ser enviado ao verificador. A escolha desta estrutura deve-se principalmente à sua simplicidade de uso, servindo apenas para armazenamento temporário de informação.

Funcionalidades realizadas

O presente trabalho na sua totalidade satisfaz todos os requisitos exigidos para a primeira meta.

Funcionalidades realizadas	
Requisitos	Estado
Planear e definir as estruturas de dados responsáveis por gerir as definições de funcionamento no gestor e no cliente.	Cumprido
Definir os vários *header files* com constantes simbólicas que registem os valores por omissão comuns e específicos do cliente e servidor bem como as estruturas de dados relevantes.	Cumprido
Desenvolver a lógica de leitura das variáveis de ambiente do gestor e do cliente, refletindo-se nas estruturas de dados mencionadas no ponto anterior.	Cumprido
Iniciar o desenvolvimento da leitura de comandos de administração do gestor implementado a leitura e validação dos comandos e respetivos parâmetros.	Cumprido
Implementação completa do comando *shutdown*.	Cumprido
Preparar a ligação entre gestor e verificador de forma a permitir testar a funcionalidade do filtro com algumas palavras enviadas a partir do gestor.	Cumprido
Desenvolver e entregar um *makefile* que possua os *targets* de compilação "*all*", "cliente", "gestor", "verificador" e "*clean*".	Cumprido

Verificação e validação

Para testar as variáveis de ambiente foram realizados vários testes com potenciais erros e na variável ambiente WORDSNOT foi realizado o teste com outros nomes de ficheiros de texto em que concluímos que o nome após ser alterado, abre o ficheiro de texto correto.

Para testar a introdução de comandos foi usada a mesma estratégia referida acima, passando por testar potenciais erros. Concluindo, que a introdução de comandos inválidos, mostra uma mensagem ao utilizador a referir o erro.

O teste da ligação entre o gestor e o verificador passou por criar um comando que enviasse uma frase para o verificador e após o retorno do valor de palavras proibidas de acordo com as palavras previamente escritas no ficheiro, pudemos comprovar a sua validação.

Comportamentos anómalos conhecidos

De momento, não há nenhum comportamento anómalo conhecido.