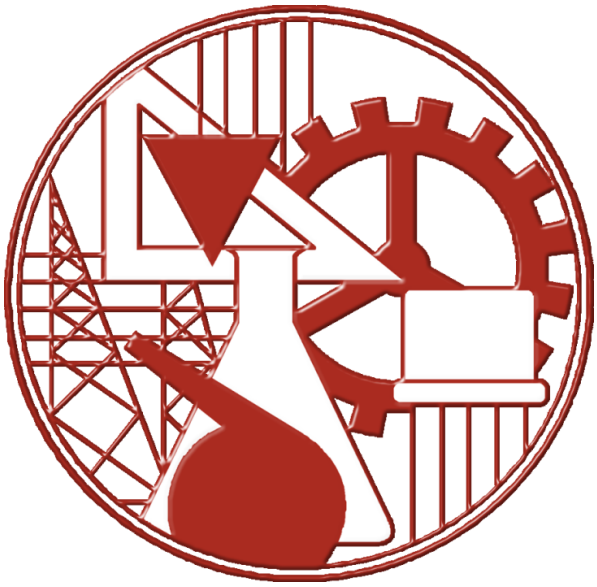
Instituto Superior de Engenharia de Lisboa

Semestre de Verão 2013/2014



PROGRAMAÇÃO DE SISTEMAS COMPUTACIONAIS

Relatório 3ªSérie

Grupo 1:
Ana Sequeira – 35479
Diogo Poeira – 36238
Daniel Silva – 39119
Turma LI31D

Docente:

Carlos Martins

Índice

Exercício 1 3
Exercício 2 4
Exercício 3 5
Exercício 4 6
Bibliografia 9

Exercício 1

http_get_file:

Neste módulo encontra-se a função *write_data* que é uma função auxiliar, que recebendo um *pointer* para o que se pretende escrever, o seu tamanho e o número de elementos, escreve para a *stream* de destino também passada como parâmetro, retornando o número de elemento escritos com sucesso para a *stream* de output.

Na função *http_get_file* são realizados inúmeros *set's* a opções do curl, mas antes é iniciado o *handler* do mesmo para ser possível realizar o *http get*, se a iniciação não se realizar com sucesso é enviado um erro para o utilizador. Seguidamente realizam-se então os *set's* referidos, primeiro é definido o URL do *handler*, posteriormente define-se que o *handler* deve fazer *redirect*, ou seja, segue qualquer caminho enviado pelo *http header*. É enviada toda a data para a função definida mais em cima, *write_data* através de outro *set*. É realizado o *fopen* que vai abrir o ficheiro passado como parâmetro ao método para onde se vai escrever então o resultado. Por fim é fechado o ficheiro onde é realizado um *flush* a todos os buffers para serem usados posteriormente, é testado o *CURLres* para verificar se existiram erros e é realizado o *cleanup* do *handler*, que liberta todos os recursos usados até então.

Exercício 2

http_get_json

A resolução deste módulo é bastante semelhante ao módulo anterior, uma vez que em ambos os casos é necessário aceder a um servidor HTTP com o objectivo de obter informação e armazená-la. Mas, neste caso, em vez de toda a informação ser escrita num ficheiro, esta será guardada em memória, e posteriormente retornada num json_t.

Para a escrita em memória, foi utilizada uma função auxiliar write_to_buffer, explicada na aula e depois disponibilizada pelo engenheiro.

Assim sendo, após a declaração de uma estrutura com um char * e um size_t , seguiu-se o mesmo processo que no exercício anterior, à excepção de em vez de mandar-mos escrever para um ficheiro, mandámos escrever para a memória. Depois de tudo isto, criámos um json_t onde colocámos toda a informação e retornando no fim essa mesma variável.

Exercício 3

Biblioteca dinâmica e makefile:

Na terceira alínea é pedido que se crie uma biblioteca dinâmica. Para tal, e visto ser-nos pedido no enunciado que se crie um *makefile* para gerar executáveis e bibliotecas, foi necessário criar um *makefile* para criar a biblioteca, assim como o executável para o quarto exercício. Após visualização e análise de vários exemplos diferentes de *makefile*, percebemos o seguinte:

- É boa prática criar "*labels"* com todas as flags que vão ser utilizadas pelo compilador (neste caso gcc);

- A estrutura é bastante simples, sendo que é apenas necessário perceber que o que está do lado esquerdo é o que vamos gerar e o que está do lado direito é o necessário para o gerar (a criação do que está do lado esquerdo depende de tudo o que está do lado direito), e que depois dessa definição, é necessário chamar o compilador e dizer-lhe como compilar e o que compilar;

- Ao correr no terminal o comando *make*,este apenas vai gerar a primeira label do ficheiro *makefile*, assim sendo, criámos uma *label* que irá gerar a biblioteca e o executável do exercício 4;

- É também boa prática ter uma label que remova os ficheiros, que no nosso caso, irá remover todos os ficheiros com a extensão .o, a biblioteca , os ficheiros executáveis e ainda a pasta criada pelo executável do exercício 4;

Uma vez que o makefile executa comandos da linha de comandos, para a remoção da pasta gerada pelo exercício 4, foi utilizado o comando rm , tal como para a remoção de todos os outros ficheiros. Ao tentar executar o clean , deparámo-nos com um erro ," rm: cannot remove `Thoth': Is a directory ". Após uma rápida pesquisa, percebemos que é necessária mais uma flag : -r. Esta flag faz com que sejam removidos os ficheiros em forma hierárquica, isto é, remove todos os ficheiros que estejam dentro da pasta e só depois é que remove a pasta.

Para a geração da biblioteca (extensão .so), é necessário gerar todos os ficheiros com extensão .o dos quais a biblioteca vai depender. Para além dessa necessário, é ainda preciso criar um ficheiro do tipo .h com a assinatura de todos os métodos e todas as bibliotecas desejadas. No nosso caso, decidimos meter todos os #include que são comuns em todos os módulos na nossa biblioteca, sendo que assim apenas é feito um #include (à excepção de dois módulos que utilizam ainda outra biblioteca).

Exercício 4

html_representation.c

Este ficheiro .c tem como objetivo facilitar o uso das *tags* do *html*. Assim sendo, este módulo tem uma função para cada uma das *tags* que são utilizadas na série de exercícios. Todas estas funções são muito parecidas no entanto, para facilitar a compreensão achou-se por bem fazer uma função para cada uma das *tags*. Cada um destas funções tem a seguinte estrutura, uso da função *malloc* pra alocar um espaço de memória suficiente para a escrita do conteúdo passado por parâmetros envolvido na *tag* correspondente, uso da função *sprintf* para escrever para o *char pointer* e posterior retorno do mesmo. Tem ainda uma função que concatena a página *html* e uma outra que concatena duas *Strings* e despeja o resultado num *char pointer* passado por parâmetro, ambas as funções seguem a mesma estrutura.

libWork.h

Esta biblioteca contém os recursos que são comuns à maior parte dos módulos e contem a definição de uma estrutura *workitem* que contem os dados referentes a um trabalho de certa cadeira.

json_resource_finder.c

Este módulo tem o propósito de, a partir do *url* base encontrar os trabalhos (*work items*) da turma passada por argumentos ao programa. Para isso usa varias funções todas elas com objectivos diferentes mas estrutura semelhante.

A função *find_class* recebe o resultado da chamada à função *http_get_json* com o *url* base da api do *thoth* e a partir dai vai buscar o *array* de classes que este *json_t* contem. Ainda nesta função, esse *array* é percorrido até se encontrar a turma especificada. É retornado um *json_t* com o resultado da função *http_get_json* com o *url* da turma encontrada ou NULL se não for encontrada nenhuma.

A função *get_projects* recebe um *json_t* que representa a turma encontrada na função *find_class* e vai buscar os *workitems* da mesma. Esta função retorna um *json_t* com o resultado da função *http_get_json* com o *url* dos *workitems*.

A função get_*workitem* recebe um *json_t* que representa o *array* de *workitems* e um inteiro que representa o índice desse *array* que se pretende obter. Esta função retorna um *json_t* com o resultado da função *http_get_json* com o *url* do *workitem* pretendido.

A função parse_*workitem* recebe um *json_t* que representa um *workitem* e o que faz é obter os dados do *json_t* referentes ao *workitem* e passa-los para uma estrutura *workitem* para facilitar o acesso aos dados. Esta função retorna um *workitem* com os dados que estão presentes no *json_t* passado por parâmetro.

workitem_to_html.c

Este módulo apenas contém uma função que recebe por parâmetro uma estrutura que representa um *workitem* e um *char pointer* onde o *workitem* vai ficar representado em *HTML*. Esta função passa os dados do *workitem* às funções do módulo *html_representation.c* para os inserir em *td* (*table data*) e no final imprime-os para uma linha usando a função *tr* (*table row*). Posteriormente é retornado o *char pointer* passado por parâmetro com a representação do *workitem* contida no mesmo.

save_thoth_work.c

Este módulo faz as ligações com todos os outros. Contém 4 funções sendo uma delas a *main*.

A função *concat_document_url*, que segue o mesmo padrão usado nas funções de *html_representation.c* , o que faz é concatenar o *url* para o enunciado do *workitem* atual, esta função recebe como parâmetro um *char pointer* para onde irá escrever, o titulo da cadeira e o id do *workitem*

A função *concat_attachment_url* que é funciona da mesma maneira que a função *concat_document_url* mas para o anexo do *workitem*.

A função *concat_title* que recebe quatro *char pointer*s como parâmetro sendo eles o titulo para onde a função vai escrever, a sigla, o semestre e o nome da turma, que a função vai concatenar separados por "/" no final a função retorna o ponteiro com tudo concatenado.

A função *parse_projects* recebe por parâmetros um *json_t* que representa o *array* de *workitems* e um *char pointer* que representa o título da turma. O que esta função faz é imprimir uma tabela em *html* para representar os dados pedidos dos *workitems*. Esta função usa as funções do *html*_representation para criar a tabela e a função *workitem_to_html* para representar os *workitems* à medida que o *array* vai sendo percorrido. É ainda feito o *download* do enunciado e do anexo caso ele exista através da função http_get_file. No final esta função retorna um *char pointer* com o conteúdo da tabela *html*.

A função *main* recebe como argumentos a sigla, o semestre e o nome da turma que se pretende obter os trabalhos. O que esta função faz é a partir das funções do *json_resource_finder* encontrar o *array* de trabalhos pretendido, criar a directoria com o nome "Thoth" se ela ainda não existir, fazer a chamada á função *parse_projects* para obter a representação em *html* dos trabalhos, bem como obter os recursos referentes aos mesmos e finalmente imprimir a representação dos trabalhos para um ficheiro *.html* dentro da directoria criada.

Bibliografia

https://jansson.readthedocs.org/en/2.6/index.html

http://curl.haxx.se/libcurl/c/simple.html

http://curl.haxx.se/libcurl/c/sepheaders.html

http://curl.haxx.se/libcurl/c/getinmemory.html

http://www.cyberciti.biz/faq/delete-or-remove-a-directory-linux-command/

http://www.jsoneditoronline.org/