Luís Vilas, João Silva e João Brito

University of Minho, Department of Informatics, 4710-057 Braga, Portugal e-mail: a95641,a91671, a91697@alunos.uminho.pt

Resumo: Trabalho desenvolvido com base no "guião-1", com conhecimentos essenciais da linguagem C, onde o foco principal do guião esteve em operações sobre ficheiros, parsing de dados e gestão de memória.

Keywords: Verificação de users, Verificação de commits, Verificação de repositórios

Relatório

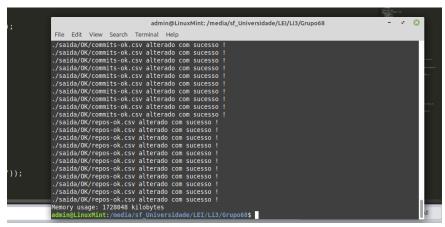
Num primeiro contacto com o guião, na aula, foi feito a struct typedef struct gh_user* GH_USER que contém todos os campos do ficheiro users, e o fopen. De seguida, declarou-se as funções int verifyInt(char* str), char* verifyStr(char* str), struct tm verifyTime (char* str), RETURN_ARRAY verifyArr(char* str), as quais usaram-se para verificar se os campos das structs eram válidas.

No ficheiro users.c foi declarada a função *GH_USER build_user(char* line)*, que utiliza as funções de verificação, referidas anteriormente, para conferir a validade das linhas do ficheiro de acordo com os parâmetros estabelecidos no guião e devolve a struct já validada. Esta função é chamada pela função *void loadUsers(char* fileName)*, que abre o ficheiro users.csv, guarda os users no ficheiro users-ok.csv e por sua vez é chamada pela *int main(int argc, char const *argv[])*.

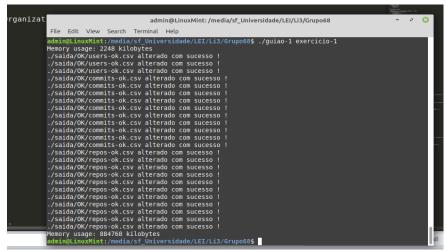
Já mais tarde, foi feito de forma idêntica aos ficheiros repos.c e commits.c, sendo que foram criadas novas structs para cada um deles (typedef struct gh_repos* GH_REPOS, typedef struct gh_commit* GH_COMMIT).

No exercício 1, encontrou-se um problema em gravar na memória as structs correspondentes ao users, aos commits e repositórios, pois usando a função void printMemory() reparou-se que estava a ser usada cerca de 8gb de ram, o que pode-se considerar um uso elevado de memória, e em certos casos o programa era "Killed". Deste modo, começou-se a optimizar o código escrito até ao momento usando free() e também deixou-se de utilizar vários campos das structs. Por isso, foram substituídas as funções, no caso do users.c, que é idêntico ao commits.c e repos.c, char* printUser(GH_USER u), void saveUser(GH_USER arrayUsers[], int numUsers, int delete),

void print_user(GH_USER u) (nos outros casos, as funções têm nome idêntico, mudando apenas o "User" por "Commit" e "Repos", respetivamente). Tendo em conta isto, apenas seriam usadas as structs para fazer a verificação dos dados e de seguida a sua remoção da memória, guardando apenas um array com, no máximo, 200 mil linhas do ficheiro original, as quais eram válidas.



F1- 2gb de RAM a ser utilizados.



F2- Após algumas optimizações, 800MB de RAM utilizados.

Por fim, no exercício 1, quando o array atingia o tamanho de 200 mil linhas gravou-se no ficheiro users-ok.csv, desalocando o espaço de memória ocupado por cada string, isto processava-se até não existirem mais linhas no ficheiro de entrada, foi feito o mesmo para os commits e para os repositórios.

No exercício 2, fez-se a gravação numa árvore binária com os IDs de todos os utilizadores, escritos no ficheiro users-ok.csv, de igual modo gravou-se o ID de todos os repositórios numa outra árvore binária. Após a gravação dos respetivos IDs, percorreuse o ficheiro commits-ok.csv e removeu-se commits inválidos tal como é pedido na alínea a) e b). Desalocou-se o espaço em memória dos IDs de repositórios.

Para a alínea c) e d), fizemos uma travessia a partir do ficheiro commits-final.csv, gerado nas alíneas anteriores, guardando numa árvore binária todos os IDs de

repositórios que teriam commits. Seguidamente, percorre-se novamente o ficheiro repos-ok.csv para remover todos aqueles repositórios cujo o IDs não constava na árvore binária dos IDs de repositórios com commits. Por fim, copiou-se o ficheiro users-ok.csv para o ficheiro users-final.csv.