Trabalho Prático – Programação em C para Unix

Descrição geral

O trabalho prático consiste na construção de um sistema de vendas de viagens em tempo real, semelhante aos portais de viagens (excepto a interface com o utilizador, que será em modo texto/consola). Ao contrário dos sistemas análogos reais, o sistema deste enunciado destina-se a ser utilizado apenas por utilizadores na mesma máquina Unix e não envolve comunicação em rede.

Arquitectura e lógica de funcionamento do sistema.

O sistema vai armazenar informação relativa a viagens aéreas entre cidades. Não interessa a que país pertencem as cidades, e não vai ser necessário considerar viagens com paragens intermédias (por exemplo, viajar entre *A* e *B* só será possível se existir um voo de *A* para *B*. Existir ou não um voo de *A* para *C* e de *C* para *B* é irrelevante).

Os utilizadores do sistema vão efectuar consultas de voos e marcações de viagens. Não existe o conceito de dinheiro. Os voos têm um determinado número de vagas que são ocupadas quando alguém "compra" uma passagem, ou que são libertadas quando alguém desmarca uma passagem.

Para além da origem e destino, vagas, ocupantes (demais dados que forem relevantes), cada voo tem uma data de partida (o dia apenas). O sistema tem uma data que é simulada e não tem nada a ver com a data da máquina. Vai ser possível fazer avançar essa data por um comando específico (ver mais adiante). Todos os voos cuja data for igual ou inferior à nova data são considerados como já tendo sido efectuados, deixando de estar em aberto para marcar ou desmarcar lugares e passam para o histórico.

O sistema vai então lidar com informação de cidades, voos, e passageiros. Estes dados podem ser consultados, acrescentados e manipulados de acordo com a utilização do sistema.

Constituição do sistema

O sistema é constituído por vários programas.

- **Servidor**. Este programa corresponde ao elemento central que faz a ponte entre todos os outros, armazena, e gere toda a informação com a qual o sistema lida.
- **Admin**. Este programa permite adicionar e gerir voos e cidades. Permite também, obter informação relativa à ocupação de voos, mudar a hora do sistema, e informação geral do funcionamento actual do sistema. Também permite efectuar alguma administração dos terminais.
- **Terminal**. Este programa permite obter informação sobre os voos existentes, marcar e cancelar uma passagem aérea. Não permite efectuar nenhuma adição de voos ou cidades, nem efectuar administração ao sistema.

Não existe nenhuma comunicação directa entre terminais e admin. O servidor é o ponto central neste sistema e é o responsável por toda a informação.

Lançamento dos programas

Todos os programas são lançados através da linha de comandos. Cada programa pode ser lançado a partir de uma sessão/*shell* diferente sem que isso afecte a sua funcionalidade (mas serão sempre lançados a partir da mesma conta de utilizador da máquina). Os pormenores do lançamento e execução de cada um são os seguintes:

- **Servidor**. Quando lançado, passa para execução em *background*. Não permite estar a correr mais do que uma vez (a segunda "cópia" detecta isso – há várias formas de detectar, proponha e use uma – e termina logo, ficando a correr apenas a primeira). O servidor pode ser terminado pelo envio do sinal SIGUSR1 pela linha de comandos ou por pedido do programa admin. Quando termina o servidor notifica o admin e os terminais para também terminarem.
- **Admin**. Tal como o servidor, em determinado instante apenas um admin pode estar em execução. Quando lançado, não passa para background. Fica em execução em *foreground* interagindo normalmente com o utilizador. Quando termina, nada de especial acontece (mas avisa o servidor, para que este saiba que o admin já não está a correr). Os restantes programas mantêm-se em execução.
- **Terminal**. O terminal pode estar a correr muitas vezes em simultâneo com ele próprio, simulando pontos de venda distintos a interagirem com a mesma central (exemplo, muitas agências de viagens a interagirem com a mesma companhia aérea). O terminal, quando se executa fica a interagir com o utilizador de forma

habitual. Quando termina, avisa o servidor e mais nada de especial acontece. Os restantes programas (inclusive outras instâncias do Terminal) mantêm-se em execução.

Funcionalidade de cada programa

Servidor

Este programa tem a seguinte funcionalidade:

- Gere toda a informação e é o único que interage directamente com os dados.
- Fica a correr em *background*.
- Pode ser terminado via SIGUSR1 enviado da linha de comandos, ou por um pedido enviado pelo programa admin.
- Não deve permitir a execução de mais do que um servidor em simultâneo.
- Não interage com o utilizador directamente.
- Quando termina avisa todos os outros programas para terminarem também.
- No início lê os dados do sistema de um ou mais ficheiros. Os dados dos voos estão em ficheiro binário, e o nome desse ficheiro está na variável de ambiente SOFICHEIRO. O formato desse ficheiro é o que for necessário e razoável.
- Quando termina salvaguarda os dados dos voos no ficheiro binário.
- Valida/gere o acesso de utilizadores no sistema.
- Existe um utilizador especial do sistema, cuja password está armazenada num ficheiro de texto cujo nome é sempre SOADMPASS (maiúsculas). Este ficheiro tem uma palavra apenas que é a password. Não existem menús/opções para modificar essa password.

Admin

Este programa tem como objectivo interagir com o administrador do sistema e não se destina a marcar/desmarcar viagens. O programa recolhe as instruções do administrador e envia-as por *named pipes* ao servidor.

A forma de interacção é muito semelhante ao conceito da linha de comandos: existe uma espécie de ciclo em que se lê uma cadeia de caracteres do écran, que corresponde a uma ordem do utilizador. Analisa-se essa cadeia de caracteres, e conforme o que tiver sido escrito, desencadeia-se uma determinada acção. Quando o ciclo termina, o programa termina. Do ponto de vista do utilizador parece que se está a interagir com uma espécie de

shell. **Nota**: o programa mantém-se em execução após a execução de cada ordem do utilizador. Não é para ter que invocar o programa para cada ordem.
O programa começa por pedir a password de administrador. Essa password é validada pelo programa servidor. O programa admin não sabe qual é a password e é o servidor que lhe vai responder se aquilo que o utilizador escreveu é a password certa ou não. De cada vez que o utilizador introduzir uma password errada, o programa obriga-o a esperar 5 segundos. Após três tentativas o programa espera mais 5 segundos e sai. Após a password certa ter sido introduzida (note-se que apenas é pedida a password), o sistema vai então ler as ordens do utilizador.

Todos as ordens que envolvem o envio de um pedido ao servidor (ou seja, quase todas as ordens) levam ao servidor a password juntamente com a informação relativa a essa ordem. Isto serve para impedir que "alguém" conhecendo o protocolo de comunicação admin – servidor fizesse um programa que enviasse ordens ao servidor sem passar pelo pedido da password primeiro. Em cada ordem recebida, o servidor valida a password e todas as ordens, uma por uma, implicam o teste de autenticidade,

As ordens suportadas são as seguintes (as letras em itálico são para ser substituídas por valores concretos, obviamente). Todas as acções que mudam o estado do sistema (data, voos, etc., são efectuados pelo servidor. O papel deste programa é o de enviar o pedido ao servidor).

- **shutdown** – manda terminar o servidor e portanto, o sistema todo.
- **addcity *nome*** – acrescenta uma cidade com o nome indicado. No máximo há 30 cidades.
- **delcity *nome*** – remove a cidade com o nome indicado desde que não haja nenhum voo por fazer que envolva essa cidade.
- **seepast** – mostra no écran o histórico dos voos que já tiveram lugar (já estão no passado). O histórico é mantido num ficheiro de texto. Esse ficheiro é sempre controlado pelo servidor, (dê o nome que entender a esse ficheiro) com a informação "voo *id* de cidade para cidade com *X* pessoas ocorreu no dia *Y*" (id, x e y são os valores concretos).
- **addvoo *id origem destino*** – adiciona um novo voo com identificação *id* da cidade *origem* para a cidade *destino*. Cada voo tem um *id* (string) que é único entre os voos actualmente em aberto (aberto = ainda não estão no passado). Todos os voos têm inicialmente 5 lugares (são aviões pequenos).

- **cancel *id*** – cancela o voo identificado por id, acrescentando assim algum realismo a este programa. O voo nem sequer vai para o histórico.
- **mudadata *dia*** – muda a data do sistema para o dia indicado em dia. O formato é um simples inteiro – é o número de dias que decorreu desde uma data qualquer de referência algures no passado. Quando o servidor arranca, a data é lida da variável de ambiente SODATA. Seja lá o que for o número que lá estiver, é essa a data actual (os voos que forem lidos inicialmente do ficheiro são logo verificados para ver se já estão no passado e se estiverem, passam para o histórico). Só se pode mudar a data para a frente (o inteiro tem que ser maior que o da data actual). Quando a data é mudada, os voos são verificados. Os que passam a estar no "passado" passam para o histórico.
- **getdata** – mostra no écran a data (dia). Não esquecer que o dia é mantido pelo servidor, pelo que este programa tem que lhe perguntar essa informação.
- **info** – mostra no écran o número de programas terminais que estiverem neste momento a funcionar e ligados ao servidor. Opcionalmente, acrescente outra informação que achar relevante.
- **adduser *username password*** – adiciona um utilizador do programa terminal (ou seja, adiciona um "agente de viagens"). Assuma que o login e password nunca tem mais que 20 caracteres cada. Os utilizadores são armazenados num ficheiro que tem sempre o nome SOAGENTES (ficheiro binário)
- **exit** – termina o programa admin e volta para a consola/Shell.

Terminal

Este programa efectua a interacção entre um agente de viagens e o sistema. O estilo de interacção é semelhante ao do programa admin. A interacção desenrola-se em duas fases. Num primeiro momento, apenas dois comandos são reconhecidos: login e exit. Após a utilização bem sucedida do comando login, entra-se na segunda fase em que todos os comandos são reconhecidos excepto o login.

Este programa, ao terminar, notifica o servidor. Tal como no caso do programa admin, quem faz as manipulações de dados é o servidor. O programa terminal e o servidor comunicam via *named pipes*.

Os comandos reconhecidos são os seguintes

- **login *username pass*** – O programa lê os *username* e *password* e envia-os ao servidor, o qual responderá indicando se é válido ou não. Após o login bem sucedido, o *username* e *password* ficam memorizados no próprio programa terminal e são sempre enviados em cada pedido dirigido ao servidor, tal como no cado do programa terminal e pelas mesmas razões.
- **exit** – faz com que o programa termine. O servidor é notificado.
- **logout** – fecha a sessão do utilizador e reverte à fase 1 da interacção com este programa.
- **mudapass *passwordantiga passwordnova*** – Pede ao servidor para mudar a password do utilizador. O valor de passwordantiga tem que bater certo com a password actual.
- **lista** – lista todos os voos que ainda não se concretizaram. (id, origem destino, lugares vagos)
- **pesquisa *origem destino*** – semelhante ao anterior mas apenas para voos com a origem e destino indicados
- **marca *id num*** – marca um lugar no voo *id* para a pessoa com passaporte com o número *num*. Uma pessoa não pode aparecer duas vezes no mesmo voo e quem verifica isso é o servidor, evidentemente. Só tem efeito para voos que ainda não se realizaram.
- **desmarca *id num*** – faz o oposto do comando anterior.

Requisitos gerais de implementação

Os programas devem ser implementados de uma forma eficiente, sem desperdiçar os recursos do sistema nem o tempo do processador. Podem existir diversas situações de erros potenciais que não são explicitamente mencionadas no enunciado. Estas situações devem ser identificadas e o programa deve lidar com elas de uma forma controlada e estável. O terminar abruptamente o programa não é considerado uma forma adequada.

Este sistema destina-se a correr numa máquina com o sistema operativo Linux. A linguagem de programação a utilizar é a linguagem C. Apenas se pretende um sistema local (ou seja, clientes e servidor correm na mesma máquina).

O mecanismo principal de comunicação para ordens entre clientes e servidor é o de *named pipes UNIX*. Vai ser também necessário recorrer à utilização de sinais.

Observações

Em algumas situações pode ocorrer que os programas precisem de dar atenção a duas coisas em simultâneo, tal como ler um comando do utilizador e dar atenção a uma notificação oriunda do servidor informando que algo ocorreu (seja o que for). Chama-se a atenção que o mecanismo de sinais em Unix é muito útil para este tipo de situações, em que se deixa o cliente numa tarefa e, quando chega um sinal (enviado pelo servidor?) se deve ir dar atenção a outra coisa, regressando depois à primeira. Os exemplos nas aulas teóricas e práticas sobre sinais esclarecem este aspecto.

Regras gerais

- O trabalho pode ser realizado em grupos de até dois alunos no máximo (sem excepções, não vale a pena enviar e-mails com pedidos nesse sentido). Também pode ser realizado individualmente.
- O trabalho vale quatro valores na nota final da disciplina.
- Prazo de entrega: final do dia de 10 de Junho (Segunda), 23:00h. Cada dia de atraso na entrega será penalizado com 25% da nota do trabalho.
- A entrega do trabalho é feita sob a forma de um arquivo zip contendo os ficheiros do projecto (código fonte) e o pdf do relatório. O arquivo tem obrigatoriamente o nome

  so_1213_tp_nome1_numero1_nome2_numero2.zip

  em que nome1 e numero1 são o nome e número de um dos alunos do grupo e nome2 e número2 dizem respeito ao segundo aluno do grupo (se existir). O não conformismo com este formato pode levar a penalizações.

Avaliação do trabalho

Para a avaliação do trabalho serão tomados em conta os seguintes elementos:

- **Arquitectura do sistema** – Embora não haja muita margem para inovação dado o grau de detalhe deste enunciado, há aspectos relativos à interacção dos vários processos que devem ser cuidadosamente planeados de forma a se ter uma solução elegante, leve e simples. A arquitectura deve ser bem explicada no relatório.
- **Implementação** – Deve ser racional, não desperdiçar recursos do sistema ou processos. As soluções encontradas para cada problema no trabalho devem ser claras. O estilo de programação deve seguir as normas da boa estruturação. O código deve ter

comentários relevantes. Os recursos do sistema devem ser usados de acordo com a sua natureza.

- **Relatório** – Deverá ser escrito e entregue um relatório completo descrevendo a estratégia e modelos seguidos, a estrutura da implementação e as opções tomadas (máximo de 10 páginas). Este relatório será enviado juntamente com o código no arquivo submetido via moodle e também será entregue em mão (i.e., impresso) na defesa.
- **Defesa** – Os trabalhos são sujeitos a defesa individual onde será testada a autenticidade dos seus autores. Pode haver lugar a mais do que uma defesa caso subsistam dúvidas quanto à autoria dos trabalhos. A nota final do trabalho é directamente proporcional à desenvoltura demonstrada durante a defesa. Elementos do mesmo grupo podem ter notas diferentes consoante o empenho individual na sua realização. Atenção: apesar da defesa ser individual, ambos os elementos do grupo devem comparecer ao mesmo tempo. A falta à defesa implica automaticamente a perda da totalidade da nota do trabalho.
- Os trabalhos que não funcionam são fortemente penalizados independentemente da qualidade do código fonte apresentado. Os trabalhos que nem sequer compilam terão uma nota extremamente baixa.
- A identificação dos elementos de grupo deve ser clara e inequívoca (tanto no arquivo como no relatório). Trabalhos anónimos não são corrigidos.