Trabalho Prático – Programação em C para Unix

Descrição geral

O trabalho prático consiste na construção de um sistema de jogos de dominó *online* em tempo real. A interface será em consola/modo texto, e todos os jogadores estarão na mesma máquina UNIX, ou seja, não se pretende a utilização de mecanismos de comunicação em rede – apenas mecanismos de comunicação inter-processo.

Arquitectura geral e lógica de funcionamento do sistema.

O sistema vai ser composto por vários processos que interagem entre si segundo a lógica de cliente-servidor usando para tal mensagens, sinais e outros recursos que forem relevantes. Um desses processos é designado de servidor e gere as regras do jogo, as peças de dominó, o estado do jogo, e os jogadores. Também serve de intermediário entre todos os processos envolvidos. Os restantes processos são designados de clientes, os quais têm como principal função interagir com o utilizador (jogador), servindo de ponte entre um jogador e o servidor.
O servidor armazena e gere toda a informação e controla todos os aspectos centrais do jogo. O cliente serve principalmente de interface com os jogadores e não valida as regras nem armazena informação local acerca das peças (caso contrário um cliente propositadamente feito de forma viciada poderia "enganar" o servidor e fazer batota, por exemplo, apresentando peças boas).

Constituição detalhada do sistema

O sistema é constituído por vários programas, os quais darão origem a processos a correr numa máquina Unix.

- **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 (neste caso, um conjunto de peças de dominó, peças "na mão" de cada jogador, quais os jogadores existentes, de quem é a vez, etc.). Só deve estar a correr um processo com este programa de cada vez. Se se tentar lançar um novo, o novo detecta a situação e termina.
- **Cliente**. Este programa permite a um utilizador juntar-se a um jogo (desde que não vá já a meio), jogar, o que inclui toda a funcionalidade de um jogo de dominó, consultar as peças que tem na mão, ver as peças já jogadas, obter nova peça, e desistir do jogo. O acesso de um utilizador ao sistema de jogos de dominó é livre, não necessitando de nenhuma conta

previamente criada (username/password). No entanto o servidor vai exigir a identificação do utilizador (um nome qualquer) para efeitos de interface com o utilizador.

Não existe nenhuma comunicação directa cliente-cliente. O servidor é o ponto central neste sistema e é o responsável por toda a informação e controlo de regras.

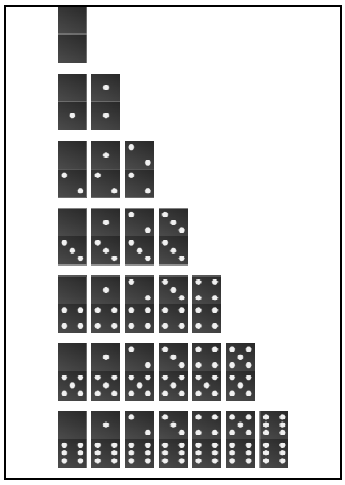
Lançamento dos programas

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 a situação e termina logo, ficando a correr apenas a primeira – há várias formas de detectar, proponha e use-a). A execução de uma segunda cópia do servidor pode ser usada para envio de ordens ao servidor (detalhes são apresentados mais adiante). O servidor pode ser terminado pelo envio do sinal SIGUSR2 pela linha de comandos. Quando termina o servidor notifica os clientes para também terminarem.
- **Cliente**. O cliente pode estar a correr muitas vezes em simultâneo com ele próprio, tantas vezes quantos os jogadores. O cliente, 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 processos (inclusive outros clientes a correr) mantêm-se em execução.

Regras do jogo

O jogo a implementar é o jogo habitual de dominós. O conjunto de peças é aquele que deriva das combinações de peças com pares de números de 0 a 6 (ver figura). As peças devem ser representadas de uma forma minimalista em modo texto mas que se percebam (em vez de pintas, apresentam-se números). O jogo é linear, ou seja, não admite ramificações. Isto significa que os "dobles" (as peças com o mesmo valor em ambas a pontas) são dispostas na sequência linear das restantes, e não na perpendicular. Assim, uma nova peça apenas pode ser jogada em dois locais: ou à esquerda, ou à direita da linha constituída pelas peças já jogadas. Esta simplificação tem a ver com o facto de o jogo ser em modo texto.



O jogo é criado por um jogador, que é logo considerado como participante desse jogo. Após a sua criação, o servidor aguarda um intervalo de tempo durante o qual aceita novos jogadores. O número de segundos a aguardar é especificado na criação do jogo. Passado o tempo máximo, o jogo começa

automaticamente. A ordem de jogadas é a ordem pela qual os jogadores se juntaram ao jogo, sendo o primeiro aquele que o criou.

No início de um jogo o servidor notifica os jogadores que o jogo começou e atribui a cada participante um conjunto inicial de peças sorteadas do molho geral de peças. As peças restantes ficam disponíveis para serem obtidas por jogadores quando não têm peças que possam ser jogadas. O número de inicial de peças a dar a cada jogador é definido na variável de ambiente NUMPECAS. Cada vez que um jogador joga, o servidor notifica todos os jogadores que esse jogador jogou, qual foi a peça jogada, qual a sequência de peças actual e qual o jogador seguinte a jogar. Uma jogada de um jogador fora da sua vez é ignorada. Sempre que um jogador vai buscar uma peça ao molho (o conjunto de peças que ainda não foram atribuídas a ninguém), o servidor notifica todos os jogadores que o jogador foi buscar uma peça ao molho (mas não diz qual é a peça, evidentemente). O servidor detecta automaticamente o fim de jogo e avisa todos os participantes que o jogo terminou e o nome do vencedor. As condições de fim de jogo são as seguintes: um jogador jogou a última peça (e ganha); todos os jogadores passaram (ganha quem tem menos peças – em caso de empate não ganha ninguém).

Funcionalidade de cada programa

Servidor

Este programa tem a seguinte funcionalidade:

- Gere toda a informação do jogo e jogadores e é o único que interage directamente com os dados e é responsável pela gestão das peças de dominó, das regras e dos jogadores.
- Fica a correr em *background* e não permite a execução continuada de mais do que um servidor em simultâneo. É aceite a execução momentânea de um segundo servidor para envio de ordens (ver mais adiante).
- Pode ser terminado via SIGUSR2 enviado a partir da linha de comandos.
- Não interage com o utilizador directamente. Não faz quaisquer perguntas via consola. Pode, quando muito, enviar mensagens para o ecrã para informação do que se passa internamente no sistema.
- Quando termina avisa todos os outros programas para terminarem também.
- Execução momentânea de um segundo servidor. Pode executar-se um segundo servidor em simultâneo com o primeiro apenas com carácter temporário para obter parâmetros da linha de comando e assim receber ordens (de um eventual administrador) que serão enviadas ao primeiro servidor. O servidor deve detectar automaticamente se já está a correr em segundo lugar ou não. Se for em segundo lugar, vai obter os argumentos de linha de comandos e envia as ordens que representam ao primeiro servidor através de *named pipes*, sem exclusão

de outros mecanismos que sejam necessários. O segundo servidor termina imediatamente após enviar ao primeiro servidor as ordens recebidas via linha de comandos.

As ordens suportadas via argumentos de linha de comandos para a segunda execução são:

- **show** – faz o servidor apresentar no ecrã a informação acerca do estado interno: se está a decorrer um jogo ou não, quais os jogadores *logados* no sistema, quais os que estão a jogar, se estiver a decorrer um jogo.
- **close** – manda o servidor encerrar. Os clientes são todos notificados.

Cliente

Este programa efectua a interacção entre um jogador e o sistema. O estilo de interacção segue a lógica consola: modo texto e comandos escritos (não é para usar o paradigma dos menús). A primeira coisa que o programa faz é pedir o nome ao utilizador para identificar em termos de interface com o utilizador. Nesta fase o jogador pode apenas introduzir o nome, ou então escrever "exit" e sair (e portanto "exit" não é aceite como nome logo a nível do cliente). O cliente envia o nome introduzido ao servidor que o aceitará apenas se não estiver nesse momento mais ninguém *logado* com esse nome. Só após a introdução e aceitação de um nome é que o cliente avança para a parte em que reconhece os comandos para interagir com o jogo.

A comunicação entre o servidor e o cliente é feita através de *named pipes*, sem exclusão de outros recursos que possam ser necessários. Todas as ordens que afectam o jogo (jogar peça, etc.) são, evidentemente, efectuadas pelo servidor, que é quem tem acesso directo aos dados. Este programa, ao terminar, notifica o servidor.

Os comandos reconhecidos pelo cliente são os seguintes

- **exit** – faz com que o programa termine. O servidor é notificado. Se o jogador estivesse a meio de um jogo, aplicam-se as regras relativas ao abandono de um jogo (ver adiante).
- **logout** – fecha a sessão do utilizador. O programa não termina. Simplesmente reverte à fase inicial em que lhe é pedido o nome. Idem quanto ao abandono de um jogo a meio.
- **status** – apresenta no ecrã a informação acerca de se está algum jogo criado (e se sim, se está a decorrer), e quais os jogadores participantes.
- **users** – apresenta no ecrã a informação acerca de quais os jogadores actualmente logados. Para cada um é apresentado o nome e o número de vitórias acumuladas durante a execução actual do servidor, e se estão actualmente a jogar.
- **new *nome intervalo*** – pede ao servidor para criar um novo jogo com o nome *nome*. O comando só é aceite pelo servidor se não estiver a decorrer actualmente nenhum jogo. O número de segundos que o servidor vai aguardar durante os quais aceita novos participantes é dado em *intervalo*. O número máximo de jogadores é 4 (valor fixo). O número de peças a dar inicialmente a cada jogador é o especificado na variável de ambiente NUMPECAS. Passado o intervalo de tempo especificado, o jogo começa automaticamente se tiver pelo menos dois participantes. Se não houver pelo menos dois participantes, o jogo é cancelado.

- **play** – pede ao servidor para participar no jogo. Apenas tem efeito se existir nesse instante um jogo criado mas ainda não iniciado.
- **quit** – desiste do jogo. Apenas tem efeito se o jogador estiver a participar num jogo. As peças deste jogador que ainda não tiverem sido usadas passam para o molho (as peças que não pertencem a nenhum jogador de onde se retiram peças quando não se consegue jogar nenhuma).

Os comandos a usar no cliente, específicos ao decorrer de um jogo (e que não têm efeito se o jogador não estiver a jogar no momento), são os seguintes:

- **tiles** – mostra as peças na mão do jogador. As peças são apresentadas de uma forma minimalista/modo texto mas que se perceba. Cada uma é identificada por um número que irá permitir mais tarde ao jogador identificar a peça a jogar.
- **info** – mostra o número de peças no molho comum de peças, o nome dos jogadores participantes e o número de peças que cada um tem. Os jogadores são apresentados pela ordem em que jogam e é indicado o próximo a jogar.
- **game** – mostra as peças já jogadas. Devem ser apresentadas com um aspecto linear (umas a seguir às outras) de uma forma minimalista mas que se entenda.
- **play *num* left | right** – joga a peça identificada pelo número *num* (o numero foi apresentado no comando *tiles*). A peça é jogada à esquerda ou à direita das que já se encontram jogadas consoante foi escrito *left* ou *right* (escreve-se ou left ou right mas não ambos e o caracter "|" também não se escreve). O servidor valida se a peça pode mesmo ser jogada.

  Nota: a peça é testada de duas maneiras: assumindo a peça com o valor *a* numa ponta e *b* na outra – peça (a,b), o servidor verifica se tanto o valor *a* como o *b* "encaixa" na extremidade especificada (left/right). Não esquecer que a peça (a,b) é a mesma peça que (b,a) e tanto pode ser jogada como (a,b) como (b,a). A peça é jogada e apresentada no ecrã da forma que encaixar. Se não encaixar o jogador deve escolher outra peça. Se nenhuma servir tem que obter uma nova. Se já não houver, tem que passar a vez.
- **get** – obtém uma nova peça do molho. O servidor verifica se o jogador precisa mesmo de uma peça nova. Se houver uma peça na mão do jogador que possa ser jogada, o comando não tem efeito. A peça obtida é sorteada pelo servidor de entre as que estão no molho.
- **pass** – prescinde da jogada. Apenas pode fazer isso se não tiver nenhuma peça que possa ser jogada e não houver mais nenhuma no molho. O servidor verifica estas condições.
- **help** – mostra as peças que podem ser jogadas para o caso de jogador estar com dificuldades em ver isso por si só.
- **giveup** – desiste do jogo. As peças deste jogador que ainda não tinham sido jogadas passam para o molho.

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*. Provavelmente 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. Nesses casos o trabalho é o mesmo e não será simplificado (também não vale a pena enviar e-mails acerca disso).
- O trabalho vale quatro valores na nota final da disciplina. Tem mínimos de 25% que serão dispensados para os alunos que tenham 90% de presenças nas aulas práticas.
- Prazo de entrega: final do dia de 7 de Junho (sexta), 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_1314_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). A não conformidade com este formato pode levar desde penalizações, atrasos no lançamento, ou até a não atribuição da nota.

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 a simplicidade estrutural do problema, 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 para não haver mal entendidos.
- **Implementação** – Deve ser racional, não desperdiçar recursos do sistema. As soluções encontradas para cada problema no trabalho devem ser claras. O estilo de programação deve seguir as boas práticas de programação. 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. Nota: 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.