Proposta de Trabalho

Titulo: caso de estudo dominó

Sistemas Operativos

Pedro Sobral
pmsobral@ufp.edu.pt
André Ribeiro Pinto
arpinto@ufp.edu.pt

Fevereiro de 2017

Universidade Fernando Pessoa

Faculdade de Ciência e Tecnologia

1 Descrição do Problema

O Dominó é um jogo tradicional constituído por várias peças rectangulares divididas ao meio. Cada uma das partes possui um número (sob a forma de pontos) entre 0 (ausência de pontos) e 6 (seis pontos). No total podem existir 28 peças correspondentes às várias combinações de números nas duas partes de cada peça.

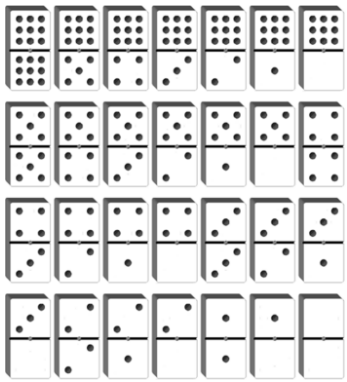

Figura 1: Conjunto total de peças de um jogo de dominó.

Na versão tradicional do dominó as peças são colocadas na mesa de jogo pelos jogadores de forma sequencial. Cada jogador coloca, à vez, uma das suas peças na mesa de jogo desde que esta encaixe numa das duas pontas da sequência de peças existente. Na sequência de peças visível todas as peças vizinhas possuem extremidades com o mesmo número de pontos.

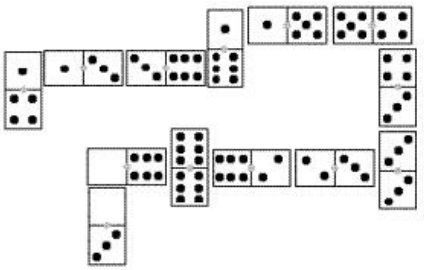

Figura 2: Exemplo de uma sequência possível de peças num jogo de dominó.

Neste projeto foi implementado uma variação solitária do jogo do dominó tradicional, permitindo apenas um jogador (o computador) e incorporando as alterações que se descrevem a seguir:

1. Não existe baralho, apenas o jogo de mão inicial retirado do conjunto total de peças disponível. Durante o jogo não é possível "comprar" mais peças e acrescentá-las ao jogo de mão;
2. O jogo de mão pode ter, inicialmente, um número de peças diferente de 7 (inferior ou superior);
3. O jogo termina quando não houver opções de encaixe disponíveis para o jogo de mão a ser utilizado nessa jogada.

Os algoritmos desenvolvidos implementam o jogador computador para que este possa pesquisar todas as sequências possíveis. Por exemplo, o jogador computador parte dum conjunto de *n* peças de dominó (jogo de mão único) e tenta criar uma sequência de encaixe que use todas as peças disponíveis (solução óptima). Nesta situação de jogo, e caso não consiga, o jogador computador deverá tentar usar o máximo número de peças disponíveis. Para o exemplo descrito a estratégia do jogador computador segue um algoritmo de pesquisa (*backtracking search*) que lhe permite descobrir qual a maior sequência de encaixe possível partindo de um conjunto inicial com *n* peças.

A procura por todas as soluções deste problema requer um esforço computacional considerável que cresce exponencialmente com o número de peças existente na mão. A solução inicial apresentada executa essa procura exaustiva no contexto de um único processo e sem nenhum tipo de paralelismo. Os processadores modernos possuem vários núcleos de processamento o que lhes permite executar em paralelo real diversas tarefas. Dado que a atividade em causa neste projeto é altamente paralelizável há todo o interesse em utilizar as potencialidades do Sistema Operativo (e do hardware que este controla) para acelerar de forma significativa a resolução do problema.

O objetivo deste trabalho, no contexto da cadeira de Sistemas Operativos, é expor os alunos aos conceitos de programação paralela e concorrente no contexto do sistema operativo UNIX recorrendo às chamadas ao sistema POSIX. Numa primeira fase do projeto será implementada uma solução para o problema envolvendo múltiplos processos em que será efetuada a utilização de diferentes técnicas de comunicação entre processos. Numa segunda fase será implementada uma solução recorrendo a tarefas POSIX (*threads*) com a utilização de estruturas de dados partilhadas protegidas por mecanismos de sincronização. Em todo o projeto o aluno deverá utilizar exclusivamente as chamadas ao sistema POSIX nativas (open, read, write,...) e não as

funções das bibliotecas C (fopen, fread, fwrite,...). Em todas as chamadas ao sistema devem ser testadas as eventuais condições de erro recorrendo à função `perror()`.

2 Requisitos Funcionais

O projeto será divido em duas fases que serão detalhadas nesta secção:

Fase 1

1. O código inicial disponibilizado deve ser cuidadosamente analisado, comentado e testado.
2. O código deverá ser modificado de forma a que o processo inicial (processo Pai) proceda à criação de N processos filhos (em que N é o número de peças da mão). Cada um do processos Filho vai começar a sua pesquisa exaustiva pelas soluções do jogo tendo como ponto de partida uma das peças da mão. Cada um dos Filhos mantêm em memória o conjunto de soluções que encontrar. Quando a pesquisa terminar, cada processo Filho envia um sinal ao Pai indicando esse facto. Quando todos os filhos notificarem o Pai da finalização da respetiva pesquisa o Pai calcula tempo decorrido entre o lançamento do primeiro processo filho e a recepção do sinal de finalização de pesquisa do último filho (**Tempo de Pesquisa**).
    a. Na versão 1, após a finalização da pesquisa das soluções, cada um dos processos Filho vai gerar um ficheiro no diretório corrente (cujo nome é o correspondente identificador do processo – PID) que deverá conter as soluções encontradas por esse processo. Após escrever as soluções para o ficheiro o processo Filho termina. Quando todos os processos filhos terminarem o processo Pai informa o utilizador do **Tempo de Pesquisa** e o **Tempo de Comunicação** e termina também.
    b. Na versão 2, vai existir um canal de comunicação (*pipe*) que permitirá a comunicação entre os processos Filho e o processo Pai. Existe apenas um *pipe* que será lido pelo processo Pai e escrito por todos os processos Filhos. De forma a ser possível identificar a origem das sequências, cada Filho fará preceder cada sequência pelo seu PID. Será necessário definir um formato de mensagem adequado para este fim. Será da responsabilidade do processo Pai a escrita final do ficheiro no diretório

corrente identificado pelo seu PID com todas as sequências geradas pelos seus filhos. Antes de terminar o processo Pai calculará também neste caso o **Tempo de Comunicação** (que inclui a leitura do *pipe* e a escrita para ficheiro).

c. Na versão 3, vai existir um canal de comunicação do tipo *unix domain socket* entre o processo Pai e cada um dos processos Filhos. O processo Pai manterá um *server socket* que os Filhos contactarão quando chegar o momento de submeter as soluções do problema após a fase de pesquisa. O processo Pai irá fazer uso da chamada ao sistema *select* para recolher o resultado de cada um dos seus Filhos e, tal como na versão 2, irá guardar as soluções em ficheiro calculando também o **Tempo de Comunicação** neste caso.

d. Para uma valorização extra existirá uma versão 4 idêntica à versão 3 com exceção do canal de comunicação usado. Neste caso serão usados *Internet sockets* para a comunicação entre os processos Filhos e o Processo Pai. Deverá também neste caso ser apresentada a estimativa do **Tempo de Comunicação**.

Fase 2

1. A fase 2 do projeto terá como ponto de partida o código original fornecido para o trabalho. Ou seja, as versões 1,2,3 e 4 não são relevantes para esta fase.
2. O código deverá ser modificado de forma a que o *main thread* proceda à criação de N *worker threads* (em que N é o número de peças da mão). Cada dos *worker threads* vai iniciar a sua pesquisa exaustiva pelas soluções do jogo tendo como ponto de partida uma das peças da mão. Cada *worker thread* mantêm em memória o conjunto de soluções que encontrar. Quando a pesquisa terminar, cada *worker thread* regista na correspondente posição de um vetor partilhado o tempo de finalização da fase de pesquisa. De seguida cada *worker thread* irá proceder à inserção das eventuais sequências encontradas numa estrutura de dados partilhada (*hash table*). Quando a inserção terminar cada *worker thread* regista na correspondente posição de um vetor partilhado o tempo de finalização da fase de inserção e termina a sua função. Quando todos os *worker threads* terminarem o *main thread* apresenta o **Tempo de Pesquisa** e o **Tempo de Inserção**, escreve um ficheiro com as sequências encontradas e termina.

3. Para uma valorização extra, o *main thread* poderá proceder à ordenação das sequências e registar no ficheiro final o conjunto das soluções de forma ordenada.

As versões 1 e 2 da fase 1 serão apresentadas na semana de 27 a 30 de Março. As restantes fases do trabalho serão apresentadas na semana de 22 a 26 de Maio.

3 Documentação

3.1 Anotações e comentários no código fonte

3.2 Relatório

Os alunos deverão entregar um ficheiro de texto, que descreva as funcionalidades/requisitos implementados, parcialmente implementados e não implementados. Devem mencionar sempre o número do requisito de acordo com a numeração utilizada neste documento de especificação:

1. Funcionalidades implementadas: devem descrever de forma resumida todas as funções desenvolvidas para assegurar os requisitos funcionais solicitados.
2. Funcionalidades não implementadas: nesta secção devem identificar de forma resumida as funcionalidades não implementadas; devem ainda apontar-se as justificações/dificuldades que impediram o seu desenvolvimento.

4 Submissão

A aplicação final (código) deve estar depurada de todos os erros de sintaxe e de acordo com os requisitos funcionais pedidos. Só serão considerados os programas que não contenham erros de sintaxe e implementem as funcionalidades pedidas. A documentação, em html ou pdf, juntamente com todo o código-fonte desenvolvido, deve ser submetida num ficheiro zip/rar (project.zip) na plataforma elearning.ufp.pt.