Operating Systems Class - Assignment 2

João Ferreira & Pedro Cristina Marques
Department of Informatics Engineering
University of Coimbra
jpbat@student.dei.uc.pt | pgcm@student.dei.uc.pt
2009113274 | 2007184032

November 2011

Introdução

Neste projecto foi-nos proposto aplicar uma transformação a uma matriz, através da formula:

```
Ux,y = Ux,y + Cx(Ux+1,y + Ux-1,y - 2Ux,y)
            +Cy(Ux,y+1 + Ux,y-1 - 2Ux ,y)
```

tirando partido do princípio da programação concorrente, de `Shared Memory` e de `Memory Mapped Files` usando, claro, semáforos `POSIX` para aceder à memória partilhada.

O nome do ficheiro base é definido como uma constante global (`FIN`) no ficheiro `heat_concurrent.c`, no mesmo ficheiro são também definidos, tal como pedido,se o programa deve estar a correr ou não em `DEBUG MODE`, o número de `STEPS` que o programa deve executar e a dimensão do quadrado (`SQUARE_DIMENSION`).

Processo Pai

O processo pai começa por executar a rotina `init()` que inicializa a memoria partilhada e os semáforos. De seguida carrega a matrix do ficheiro para a memória partilhada, passando depois à transformação da mesma com o número de processos definido acima. Depois de meter os processos filhos a trabalhar faz `sem_wait` ao semáforo `change`, para que saiba quando os filhos acabaram de trabalhar e possa continuar com a execução do programa.

Quando volta a correr, este processo mata os filhos com o uso da rotina `kill`, enviando um sinal `SIG_KILL` a cada um dos `PIDs` dos filhos. Depois disto verifica se está num dos `STEPS` em que tenha que escrever para ficheiro e caso se confirme chama a rotina `write_to_file()`. Após isto faz a alteração da matriz à qual deve ir buscar os valores para trabalhar. Quando tudo isto estiver feito chama a rotina `terminate()` que fecha os semáforos e remove a `Shared Memory`.

SharedMemory, Semáforos e MemoryMappedFiles

No segmento de memória partilhada encontra-se a seguinte informação:
+`next` - que indica a cada processo qual a linha que deve processar.
+`this` - a versão da matriz à qual os processos devem ir buscar os valores.

No que diz respeito aos semáforos, foram necessários 3, sendo eles:
-`mutex` - garante o acesso em exclusão mutua às variáveis.
-`remainingLines` - informa os processos do número de linhas que ainda falta processar, e que garante que estes não o ultrapassam.
-`change` - controla o fluxo de trabalho do processo pai.

Por fim foram mapeados 4 ficheiros, sendo que isto foi feito à medida que se revelou ser necessário guardar os mesmos.

Rotina Worker()

Na rotina worker cada processo entra num ciclo infinito no qual faz `sem_wait()` ao semáforo `remaining_lines` e caso consiga prosseguir então acede ao `next` que está em `Shared Memory` para saber qual a linha que deve correr. Depois faz a transformação e verifica se foi o ultimo processo a correr. Caso tenha sido faz `sem_post()` no semáforo `change`, que faz com que o processo pai volte a trabalhar.

SpeedUp & Occupancy

Para termos noção de qual era o nosso `speedup` e a `ocupância` do nosso `CPU`, foi desenvolvido um script em `bash` que executa cada uma das versões (sequencial e concorrente) 30 vezes, guardando cada um deles para ficheiro. Após isso chama outro script que calcula a média, e o desvio padrão de cada uma das versões. Assim chegamos aos seguintes resultados:

Processes	Avg(ms)	StdDev(ms)	SpeedUp	Occupancy
1	20970	480	———	———
2	12375	255	1.69	84.73%

Para analisar a tabela convém primeiro referir as seguintes fórmulas: `SpeedUp` $= \frac{AvgSequencial}{AvgConcorrente}$ e `Occupancy` $= \frac{SpeedUp}{NumeroDeCores}$ e também que todos os testes foram corridos numa máquina com 2 `cores`, sendo que cada um deles tem `2.80Ghz` de velocidade.

Conclusão

Assim conclui-se que `Shared Memory` e `Memmory Mapped Files` são ferramentas que podemos utilizar, com percaução, para aumentar-mos a performance dos nossos projectos, uma vez que nos permite comunicar com bastante rapidez entre processos.