Memória principal e memória virtual

Rômulo Pedro Thomsen
romulot@edu.univali.br
Campus UNIVALI Itajaí

1. Introdução

O objetivo desse projeto é de forma prática abordar leitura de páginas de memória em 16 e 32 bits, foi utilizado a linguagem de programação C++ por conta de sua capacidade de mexer com operações bitwise, junto com o amplo acesso a documentação e a vasta quantia de bibliotecas para resolver quaisquer problemas que pudessem ser encontrados.

Ambos os projetos aceitam endereços de memória como argumento de linha de comando, e mostram informações na tela para o endereço selecionado, com o 32 bits possuindo informações sobre as duas páginas requeridas para a dupla paginação.

2. Implementação

Os parâmetros das operações bit a bit podem ser alterados no começo do projeto, sendo variáveis globais. Ambas as funções de obter as informações dos endereços estão escritas no começo dos dois códigos, sendo elas a *getPaginaDeslocamento()*

O código checa os argumentos colocados no código, e dá display em um erro e finaliza se for dado a quantia errada de argumentos, ou nenhum argumento.

A maior diferença nos códigos diz respeito ao começo, com as variáveis e a função previamente mencionada, o resto do código permanece o mesmo, ele apenas pega os valores obtidos com a função e o arquivo *data_memory.txt* e os mostra na tela.

16:

```
int valorDeslocamento = 12; // Contagem do deslocamento
int tamanhoPagina = 1 << valorDeslocamento;
int mascaraDeDeslocamento = (1 << valorDeslocamento) - 1; // Tamanho do
deslocamento pra operação 32

// Função para calcular a página e o deslocamento com base em um endereço
std::pair<int, int> getPaginaDeslocamento(int endereco) {
int pagina = endereco >> valorDeslocamento;
int deslocamento = endereco & (tamanhoPagina - 1);
return std::make_pair(pagina, deslocamento);
}
```

32:

```
int valorDeslocamento = 12;
int tamanhoPagina = 10;
int mascaraDeslocamento = (1 << valorDeslocamento) - 1;

std::tuple<int, int, int> getPaginaDeslocamento(int endereco) {
int paginaA = endereco >> (valorDeslocamento + tamanhoPagina); // Desloca os bits do
endereço para a direita
int paginaB = (endereco >> valorDeslocamento) & 0b1111111111; // Aplica uma
máscara para obter os bits da segunda página
int deslocamento = endereco & mascaraDeslocamento; // Aplica uma máscara de bits
para manter apenas os bits de deslocamento
return std::make_tuple(paginaA, paginaB, deslocamento);
}
```

3. Uso e resultado

Com o código compilado, é possível rodar ele direto do terminal ou linha de comando, aceitando um único parâmetro como o endereço de memória a ser visto.

Entrada	Saída
./16 19986	Endereco fisico: 19986 Deslocamento: 3602 Valor lido: 0 Pagina: 4
./32 19986	Endereco fisico: 19986 Deslocamento: 3602 Valor lido: 0 Primeira pagina: 0 Segunda pagina: 4

4. Considerações finais

Primeira coisa de se notar é que pela maior parte das informações, para essa carga específica de informação, a mudança de 16 para 32 bits não acarretou em extrair muita informação adicional, mas, para casos mais intensos possuir mais bits para trocar e guardar valores pode sim ser benéfico, isso é principalmente visto com a evolução dos antigos processadores para o que temos de hoje em dia, e como a quantia de bits de processamento foram aumentando.

De hoje em dia os computadores e outros aparelhos trabalham com *gigabytes* após *gigabytes* de informação, mas ainda há eletrônicos que possuem memória a risca, como chips de localização por exemplo, para esses sistemas é importante entender como a memória é manipulada para realizar melhor uso do *hardware*.