Documentação do algoritmo de paginação WSClock.

Docente: Ernesto de Souza Massa Neto
Discentes: João Vitor Mendes, Joseph Samuel Neiva

Gerenciamento de Memória e Paginação.

O gerenciamento de memória, em conjunto com o gerenciamento de processos, formam o fundamento de um sistema operacional. Sua importância reside fundamentalmente no fato do processador executar instruções trazidas da memória, sobre dados trazidos da memória e guardando resultados na memória. A memória é organizada em blocos ou quadros (frames) de tamanho fixo. A tabela de blocos livres registra quais blocos estão livres. Cada processo é dividido em páginas de igual tamanho que o bloco. Assim, uma página de um processo pode ser carregada em um bloco de memória. A Tabela de páginas registra em que bloco de memória cada página está carregada. As tabelas de páginas são mantidas pelo sistema operacional. Se uma página é acessada mas não está na memória, o S.O. consulta a tabela de blocos livres, aloca a página no bloco selecionado e atualiza a tabela de páginas.

Algoritmo de Paginação

- Algoritmo do Relógio
  Lista circular com ponteiro apontando para a página mais antiga, na forma de um relógio, a cabeça aponta para a página mais antiga, e quando ocorre um page fault, Inspeciona-se a cabeça da lista. Se o bit de referenciação for igual a 0, substitui-se a página da cabeça pela nova página, e avança a cabeça em uma posição, caso o bit de referência seja igual a 1, então avança a cabeça em uma posição, e repete o processo até encontrar página com bit igual a 0.

- Algoritmo Working Set (WS)
  No WS a paginação por demanda, as páginas são carregadas na memória somente quando são necessárias. O conjunto de páginas que um processo está efetivamente utilizando (referenciando) em um determinado tempo t. Objetivo principal é reduzir a falta de páginas, sendo assim, um processo só é executado quando todas as páginas necessárias no tempo t estão carregadas na memória. A idéia é determinar o working set de cada processo, e certificar-se de tê-lo na memória antes de rodar o processo.

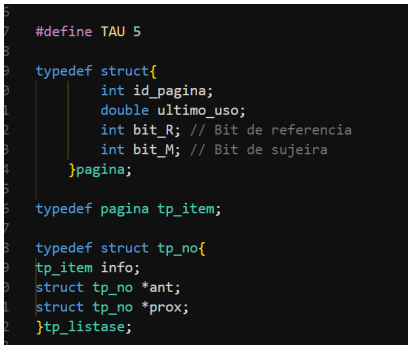
- Algoritmo WSClock
  É a combinação dos algoritmos do Relógio e Working Set. E é amplamente usado, devido à sua simplicidade e alta performance. Utiliza lista circular de páginas inicialmente vazias, e à medida que mais páginas são carregadas, entram na lista, formando um anel. Cada entrada contém o tempo de último uso, além dos bits de referenciação e sujeira.

Funcionamento do WSClock

A cada page fault, a página da cabeça é examinada primeiro. Se o bit R for igual a 1. A página foi usada durante o ciclo de clock corrente, não é candidata a remoção, então faz o bit R ser igual a 0, e avança a cabeça à próxima página, repetindo o algoritmo para esta página. Se o bit R for igual a 0, haverá outra condicional verificando se a idade da página for maior que o tamanho do working set t, e a página estiver limpa (bit M=0). então não está no working set e uma cópia válida existe no disco. A página então é substituída e a cabeça da lista avança. Se, contudo, a página estiver suja (bit M=1), então não possui cópia válida no disco, e agenda-se uma escrita ao disco, evitando troca de processo, para então avançar a cabeça da lista. Se a cabeça der uma volta completa na lista sem substituir, e pelo menos uma escrita no disco foi agendada, a cabeça continua se movendo, em busca de uma página limpa, em algum momento a escrita agendada será executada, marcando a página como limpa. Se nenhuma escrita foi agendada, todas as páginas estão no working set, na falta de informação adicional, substitua qualquer página limpa. Se nenhuma página limpa existir, escolha qualquer outra e a escreva no disco.

Código

O código a seguir é autoral, e foi construído na linguagem C.



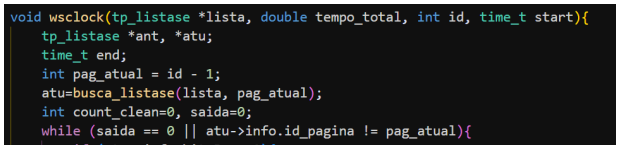
```c
#define TAU 5

typedef struct{
        int id_pagina;
        double ultimo_uso;
        int bit_R; // Bit de referencia
        int bit_M; // Bit de sujeira
    }pagina;

typedef pagina tp_item;

typedef struct tp_no{
tp_item info;
struct tp_no *ant;
struct tp_no *prox;
}tp_listase;
```

Primeiramente foi definido a constante t como igual a 5. Em seguida declaramos uma struct página, constituída pelo id da página, o tempo de último uso e os bits de Referência e Sujeira. Definimos tp_item como uma constante do tipo página, e uma struct tp_listase para acessar os endereços da lista circular, com as constantes info, para a informação atual, *ant, que é um ponteiro do struct, que irá apontar para a página anterior, e *prox irá apontar para a próxima página.



```c
void wsclock(tp_listase *lista, double tempo_total, int id, time_t start){
    tp_listase *ant, *atu;
    time_t end;
    int pag_atual = id - 1;
    atu=busca_listase(lista, pag_atual);
    int count_clean=0, saida=0;
    while (saida == 0 || atu->info.id_pagina != pag_atual){
```

A seguir temos, a função do WSClock, que recebe como parâmetro o endereço da lista, o tempo total no instante que a função for executada, o *id* da página, e o tempo inicial. Ademais, a função declara os ponteiros **ant* e **atu*, para as posições anteriores e atuais da página, Marca o tempo do instante em que a função foi executada, para posteriormente calcular a idade da página. *Atu* será o nó resultante da busca da lista em dita página. E para encerrar as declarações, a função tem um contador de páginas limpas e uma constante *saida* que servirá de estado booleano para realizar a saída da condição de loop do while.

```
while (saida == 0 || atu->info.id_pagina != *pag_atual){// Laço para substituição das páginas
    if (atu->info.bit_R == 1){// Condicional para verificar se o bit referêncial é igual a 1
        printf("Pagina atual:\n[id: %i|ultimo uso: %.11f|bit R: %i]\n",
        atu->info.id_pagina,atu->info.ultimo_uso,atu->info.bit_R);
        clean_bit_r(atu);// Lipando o bit de referência da pagina
        if (atu->prox!=NULL){//Condicional para verificar se o próximo nó é diferênte de nulo
            atu = atu->prox;// Avançando a lista para o proximo nó
        } else {
            atu = lista;// Voltando a lista para o primeiro nó
        }
        saida = 1;// Variavel para funcionamento correto do laço condicional
```

Nesse trecho havemos um loop while, para percorrer toda a lista até que a função percorra toda a lista. Em seguida, verifica se o bit R for igual a 1, se essa condição for verdadeira, imprime as informações do *id_pagina*, *ultimo_uso* e o *bit_R*, então executa a função *clean_bit_r();* para zerar o bit de referência, verifica se a próxima página é nula (fim da lista), e avança para o primeiro nó caso esteja no final, ou o próximo nó caso contrário. E realiza a saída, repetindo o algoritmo para esta página.

```
} else if(atu->info.bit_R == 0){// Caso bit de refência esteja zerado
    if ((tempo_total-atu->info.ultimo_uso)>TAU){// Verificação da idade da pagina em relação a TAU
        if (atu->info.bit_M == 1){// Condicional para verificar se a pagina ainda está suja
            printf("Pagina atual:\n[id: %i|ultimo uso: %.11f|bit R: %i]\n",
            atu->info.id_pagina,atu->info.ultimo_uso,atu->info.bit_R);
            clean_bit_m(atu);// Limpando a sujeira da pagina
            count_clean++; // Variavel que indica se alguma pagina foi limpa nessa passada
            if (atu->prox!=NULL){//Condicional para verificar se o próximo nó é diferênte de nulo
                atu = atu->prox;// Avançando a lista para o proximo nó
            } else {
                atu = lista;// Voltando a lista para o primeiro nó
            }
            saida = 1; // Variavel para funcionamento correto do laço condicional
```

Caso o bit R não seja igual a 1, verifica se o bit é igual a 0 (por via das dúvidas), e ainda há outra condicional verificando se a idade da página for maior que o tamanho do working set *TAU*, e logo depois se a página estiver suja, *bit_M==1,* em seguida, se a condição for verdadeira, imprime as informações do *id_pagina*, *ultimo_uso* e o *bit_R* , executa a função *clean_bit_r();* , soma o contador de páginas limpas e verifica se a próxima página é nula (fim da lista), e avança para o primeiro nó caso esteja no final, ou o próximo nó caso contrário. E realiza a saída, repetindo o algoritmo para esta página.

```
} else {// Caso a pagina esteja limpa ela será substituida
    printf("---------------------------------------\n");
    printf("Pagina substituida:\n[id: %i|ultimo uso: %.11f|bit R: %i]\n",
    atu->info.id_pagina,atu->info.ultimo_uso,atu->info.bit_R);
    printf("---------------------------------------\n");
    end = time(NULL);// Zerando o final do cronometro
    // Sustituição dos valores da pagina para os valores das novas paginas
    atu->info.id_pagina = id;
    atu->info.ultimo_uso = difftime(end, start);
    atu->info.bit_M = 1;
    atu->info.bit_R = 1;
    if (atu->prox!=NULL){//Condicional para verificar se o próximo nó é diferênte de nulo
        atu = atu->prox;// Avançando a lista para o proximo nó
    } else {
        atu = lista;// Voltando a lista para o primeiro nó
    }
    *pag_atual = atu->info.id_pagina; // Pegando o valor do id da proxima pagina
    return;// Encerramento da função
}
```

Caso a página esteja limpa, *bit_M==0*, imprime as informações do *id_pagina*, *ultimo_uso* e o *bit_R*, encerra o final cronômetro, e para agendar a escrita no disco, o *id_pagina* é atribuído o id, o *ultimo_uso* é calculado, e os bits R e M são atribuídos o valor 1, e avança para o primeiro nó caso esteja no final, ou o próximo nó caso contrário. E realiza a saída, repetindo o algoritmo para esta página. E por fim, o endereço da página atual recebe o id da próxima página, e realiza o *return;* para encerrar a função.