Relatório Sistemas Operacionais

Algoritmos de Substituição LRU e MFU

Julia Rodrigues Gubolin, Gustavo Pereira Pazzini, Gustavo Kenji Kuroda de Oliveira e Paulo Tavares Borges

22 de julho de 2021

Sumário

1.0 Introdução

Neste relatório, a equipe irá expor todo o processo de construção dos algoritmos de substituição LRU e MFU.

2.0 Objetivos

Com este trabalho, a equipe visa explicar o funcionamento dos códigos que simulam o comportamento dos algoritmos de substituição LRU e MFU, de maneira simples e objetiva.

3.0 Implementação e funcionamento

Para a implementação, foi utilizada a linguagem de programação C. Abaixo, os códigos serão explicados por partes, cada uma delas sendo uma função na implementação.

A equipe tomou como base para a confecção do código que a lista encadeada representa a memória, a qual recebe o endereço, e que o arquivo .log contém as operações que serão executadas na memória. Essas operações estão estruturadas da seguinte maneira: em uma linha do arquivo temos, mais à esquerda, o endereço de memória que deve receber a ação e, mais à direita, a operação que deve ser feita naquele endereço. As operações possíveis são duas: “R” indicando leitura e “W” indicando escrita.

O funcionamento geral do projeto está da seguinte forma: assim que o programa é iniciado, é definido o tamanho de uma página como sendo 64 e o tamanho da memória como sendo 8192. Com isso, pode ser calculado o número de páginas que estarão disponíveis na memória, o qual é 128.

Após estas definições, o arquivo “operações.log”, já pode ser aberto para leitura e suas informações podem ser analisadas.

Enquanto o arquivo possuir informações, a cada linha lida a variável “operacoes” é incrementada, indicando o número de operações que estão sendo concluídas. Se ao final da linha o símbolo lido for “W” ou “w”, o programa irá ou criar uma página nova ou substituir uma já salva na memória, caso o número de páginas possíveis já tenha sido ultrapassado. Por outro lado, se ao final da linha o símbolo lido for “R” ou “r”, a página que contém o endereço lido é pesquisada e, caso encontrada, o contador “acessada” é incrementado em um. Caso a página não exista, ela é escrita na memória.

Por fim, existem duas verificações: caso o arquivo lido esteja formatado de maneira equivocada, o programa é encerrado e o usuário recebe uma mensagem de aviso; e caso o arquivo de entrada esteja vazio, o usuário também é notificado com uma mensagem e o programa é encerrado.

A seguir, serão apresentadas as funções que garantem o funcionamento mencionado acima.

3.1 Função main

Na função main, temos a leitura do arquivo de operações e a adição ou leitura das páginas representadas pela lista dinâmica dependendo se o símbolo lido for “R” ou “W”.

Ao final da leitura, são mostradas na tela as informações sobre o funcionamento do LRU e a memória é limpa, ou seja, a lista é destruída.

```
int main(){
    arq = "operacoes.log";
    tamPag = 64;//Varia normalmente de 2 a 64
    tamMem = 8192;//Varia normalmente de 128 a 16384
    numPags = tamMem/tamPag;
    if(strlen(arq) > 0){
        file = fopen(arq, "r");
        while(fgets(linha, 20, file) != NULL){
            operacoes++;
            strncpy(end_temp, linha, 8);
            end_temp[8] = '\0';
            if(linha[9] == 'W' || linha[9] == 'w'){
                escrever_end(end_temp);
            }
            else if(linha[9] == 'R' || linha[9] == 'r'){
                if(encontrar(end_temp)){
                 acessada++;
                }
                else{
                    pag_n_encontrada++;
                    escrever_end(end_temp);
                }
                lidas++;
            }
            else{
                printf("ERRO: Os dados do arquivo de entrada estao em
formato incorreto.");
                return 0;
            }
        }
    }
    else{
        printf("ERRO: Arquivo de entrada invalido.");
        return 0;
    }
    printf("\nExecutando o simulador...\n");
    printf("Tamanho da memoria: %dKB\n", tamMem);
    printf("Tamanho das paginas: %dKB\n", tamPag);
```

```
    printf("Tecnica de reposicao: LRU\n");
    printf("Numero de paginas: %d\n", numPags);
    printf("Operacoes no arquivo de entrada: %d\n", operacoes);
    printf("Operacoes de leitura: %d\n", lidas);
    printf("Operacoes de escrita: %d\n", escritas);
    printf("Paginas acessadas(hit): %d\n", acessada);
    printf("Paginas de leitura nao encontradas(misses): %d\n",
pag_n_encontrada);
    printf("Numero de paginas sobrepostas(writebacks): %d\n",
sobreposta);
    printf("Taxa de falhas de pagina(page fault): %f%% \n",
falha/escritas*100);
    limpa_memoria();
    return 0;
}
```

3.2 Início do programa

No início, temos a declaração das bibliotecas que serão utilizadas, bem como a declaração da struct “Pagina”, a qual é a base para uma lista encadeada simples, e as variáveis globais que serão utilizadas durante a implementação e que serão responsáveis por armazenar os resultados que serão vistos pelo usuário.

O atributo “cont” da struct pagina está apenas na implementação do MFU.

As variáveis “*primeira” e “*última” indicam o início e o final da lista, respectivamente, e auxiliam, em ambos os algoritmos (LRU E MFU) a encontrar as páginas que devem ser substituídas.

A variável “*file” do tipo FILE é um ponteiro para o arquivo de operações que devem ser obtidas ao iniciar o programa.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <stdbool.h>

typedef struct Pagina{

  char endereco[9];

  //Contador de acesso
```

```
  int cont;

  struct Pagina *prox;

}Pagina;

//Variáveis globais

char *arq, linha[20], end_temp[9];

int   tamPag,   tamMem,   numPags,   operacoes=0,   lidas=0,   escritas=0,
acessadas=0, pag_n_encontrada=0, sobreposta=0, pags_usadas=0;

float falha=0;

FILE *file;

Pagina *primeira, *ultima;
```

3.3 Nova página

Na função para criar uma nova página, o parâmetro recebido é o valor que representa o endereço onde a página deve ser armazenada.

Para criá-la, foi utilizado o malloc, já que é uma lista dinâmica. Seus campos “endereco”, “cont” (apenas para MFU) e “prox” recebem os novos valores, neste caso: “valor”, 0 e NULL, respectivamente.

Caso não exista uma página na memória ainda, os ponteiros “*primeira” e “*última” passam a apontar para a nova página, senão, apenas o ponteiro “*ultima” muda sua posição e passa a apontar para a página recém inserida.

Por fim, se o número de páginas usadas for menor que o definido na função main, aumentamos o contador para páginas usadas e o contador para páginas escritas.

```
void nova_pag(char valor[9]){

     Pagina *atual = (Pagina*)malloc(sizeof(Pagina));

       strcpy(atual->endereco, valor);

         atual->cont = 0;

         atual->prox = NULL;

         if(pags_usadas == 0){

           primeira = atual;

           ultima =  primeira;
```

```
        }
        else {
            ultima->prox = atual;
            ultima = atual;
        }
        if(pags_usadas < numPags)
            pags_usadas++;
        escritas++;
    }
```

3.4 MFU e LRU

Cada uma das funções abaixo faz parte de seu respectivo código, ou seja, não compartilham a mesma implementação. Porém, a única mudança está no nome.

Neste método, será criada uma nova página e, se o número de páginas já usadas for igual ao limite aceito pelo sistema, descartamos a primeira passando o ponteiro que aponta para ela para a próxima.

```
void MFU(char valor[9]){
    nova_pag(valor);
      if(pags_usadas == numPags)
         primeira =  primeira->prox;
}
```

```
void LRU(char valor[9]){
    nova_pag(valor);
    if(pags_usadas == numPags)
      primeira =  primeira->prox;
}
```

3.5 Encontrar

A função encontrar tem como objetivo encontrar uma página referenciada no arquivo .log baseada em seu endereço.

Nela, a lista de páginas será percorrida e, caso o endereço passado como parâmetro tenha sido encontrado em alguma página, a função retorna verdadeiro e, caso contrário, retorna falso.

A função encontrar está presente em ambos os códigos, porém possui diferenças na implementação.

Encontrar do LRU:

Neste caso, as ações “ultima->prox = temp”, ”ultima = tmp” e “tmp->prox = NULL” não precisam de nenhuma verificação adicional, elas iram ocorrer de qualquer maneira caso o endereço buscado seja igual ao endereço de alguma página.

```
bool encontrar(char valor[9]){
    Pagina *tmp = primeira, *prev = NULL;
    while(tmp != NULL){
        if(strcmp(tmp->endereco, valor)==0){
                if(prev != NULL){
                    if(tmp->prox != NULL)
                        prev->prox = tmp->prox;
                }
                else {
                  primeira =  primeira->prox;
                }
                ultima->prox = tmp;
                ultima = tmp;
                tmp->prox = NULL;
            return true;
        }
        prev = tmp;
        tmp = tmp->prox;
    }
    return false;
}
```

Encontrar do MFU:

No MFU, para que as ações “aux = primeira”, ”primeira = tmp” e “tmp->prox = aux” ocorram, o valor do contador da página apontada por tmp deve ser maior que o contador da última página. Se esta condição for verdadeira, a página apontada por tmp se torna a primeira página da lista, a qual tem mais chance de ser substituída em caso de page fault.

```
bool encontrar(char valor[9]){
    Pagina *tmp = primeira, *prev = NULL;
    while(tmp != NULL){
```

```
        if(strcmp(tmp->endereco, valor)==0){
                if(prev != NULL){
                    if(tmp->prox != NULL){
                        tmp->cont++;
                        prev->prox = tmp->prox;
                    }
                }
                else {
                  primeira =  primeira->prox;
                }

        if(tmp->cont >= primeira->cont){
            aux = primeira;
            primeira = tmp;
            tmp->prox = aux;
        }
            return true;
        }
        prev = tmp;
        tmp = tmp->prox;
    }
    return false;
}
```

3.6 Substituir e escrever endereço

Na função de substituir uma página, a única mudança que ocorre de uma implementação para outra é no nome da função chamada, porém o funcionamento em ambos, LRU e MFU, é o mesmo. Seu objetivo é substituir a página mais referenciada ou em desuso por mais tempo, dependendo do algoritmo em questão.

A função para escrever endereço deve escrever o endereço passado por parâmetro em uma página. Caso ainda haja espaço, uma nova página é criada e o endereço é atribuído a ela. Se não, ocorre uma falha e uma página deve ser substituída.

Substituir página do MFU:

```
void substituir_pag(char valor[9]){
    MFU(valor);
    sobreposta++;
}
```

Substituir página LRU:

```
void substituir_pag(char valor[9]){
    LRU(valor);
```

```
    sobreposta++;
}
```

Escrever endereço de LRU e MFU:

```
void escrever_end(char valor[9]){
    if(pags_usadas < numPags){
        nova_pag(end_temp);
    }
    else{
        falha++;
        substituir_pag(end_temp);
   }
}
```

3.7 Limpa memória

A função de limpar memória tem como objetivo apenas destruir a lista encadeada ao final da execução do programa. Ambos os códigos possuem essa função, porém com algumas diferenças na implementação.

Limpar memória no LRU:

```
void limpa_memoria(){
    Pagina *tmp = primeira;
    while(tmp != NULL){
        tmp = tmp->prox;
        free(tmp);
    }
    fclose(file);
}
```

Limpar memória no MFU:

```
void limpa_memoria(){
    Pagina *ant = NULL, *tmp = primeira;
    while(tmp != NULL){
        ant = tmp;
        tmp = tmp->prox;
        free(ant);
    }
    fclose(file);
}
```

5.0 Conclusão

A equipe encontrou como ponto de dificuldade manusear a lista dinâmica para tratá-la como uma simulação da memória, ainda mais que ela precisaria receber as operações que deveriam ser executadas de alguma maneira.

Diante do exposto, podemos concluir que ambos os algoritmos possuem o mesmo objetivo, porém o comportamento é distinto. Além disso, o melhor funcionamento está no LRU, já que este se aproxima do algoritmo ideal.