Lab04-Memoria

Nome:Hao Yue Zheng-10408948
Nome:Smuel Zheng-10395781
Nome:Vitor Pasquarelli Cinalli-10401806

Linker de README

https://github.com/mauriciohao/Sistema-operacional.git

1.Considerando a estrutura de dados celula, crie três instâncias do objeto célula (três valores na lista);
2.Construa uma função que imprima todos os valores da lista;
3.Calcule a quantidade de memória gasta por cada instância da célula
4.Construa uma função que remove os elementos da lista;
5.Incremente sua função liberando a memória quando um elemento é liberado;
6.Calcule o máximo de elementos possíveis na fila, considerando a memória disponível no computador.

Codigo

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  // Necessário para funções e constantes do sistema, como sysconf.

// Estrutura para representar uma célula da lista encadeada.
struct reg {
    int conteudo;        // Conteúdo da célula, neste caso um inteiro.
    struct reg *prox;    // Ponteiro para a próxima célula da lista.
};
typedef struct reg celula; // Definição de tipo para facilitar a declaração de novas células.

// Protótipos das funções utilizadas no programa.
void imprime(celula *le);
void insere(int x, celula *p);
void libera_lista(celula *le);
int calcula_memoria();
int capacidade_maxima();
```

```
// Função para inserir um novo elemento na lista após a célula apontada por 'p'.
void insere(int x, celula *p) {
    celula *nova = malloc(sizeof(celula));  // Aloca memória para a nova célula.
    nova->conteudo = x;                     // Define o conteúdo da nova célula.
    nova->prox = p->prox;                   // A nova célula aponta para o próximo elemento após
'p'.
    p->prox = nova;                         // A célula 'p' agora aponta para a nova célula.
}

// Função para imprimir todos os elementos da lista encadeada.
void imprime(celula *le) {
    for (celula *p = le->prox; p != NULL; p = p->prox) {
        printf("%d\n", p->conteudo);        // Imprime o conteúdo de cada célula.
    }
}

// Função para liberar toda a memória alocada para a lista.
void libera_lista(celula *le) {
    celula *p = le->prox, *q;
    while (p != NULL) {
        q = p->prox;                        // Guarda o próximo elemento.
        free(p);                            // Libera a memória da célula atual.
        p = q;                              // Avança para o próximo elemento.
    }
    le->prox = NULL;                        // A cabeça da lista aponta para NULL após limpar a lista.
}

// Função para calcular a quantidade de memória utilizada por cada célula.
int calcula_memoria() {
    return sizeof(celula);
}

// Função para calcular o número máximo de células que podem ser alocadas na memória
disponível.
int capacidade_maxima() {
    long mem_total = sysconf(_SC_PHYS_PAGES) * sysconf(_SC_PAGE_SIZE); // Calcula o
total de memória disponível.
    return mem_total / sizeof(celula);                                 // Divide pelo tamanho de uma
célula.
}

// Função principal para executar operações com a lista encadeada.
int main() {
    celula cabeca;          // Célula cabeça para a lista.
    cabeca.prox = NULL;     // Inicialmente, a lista está vazia.

    // Insere elementos na lista.
    insere(10, &cabeca);
```

```
    insere(20, &cabeca);
    insere(30, &cabeca);

    printf("Elementos da lista:\n");
    imprime(&cabeca);

    printf("Memória por célula: %d bytes\n", calcula_memoria());
    printf("Capacidade máxima de elementos: %d\n", capacidade_maxima());

    libera_lista(&cabeca);  // Libera a memória alocada para a lista.

    return 0;
}
```

Soluçao