Universidade Federal de Viçosa - Campus Florestal

Disciplina: Sistemas Operacionais

Professor: Daniel Mendes Barbosa

Trabalho Prático 3

Eduardo Vinicius – 3498

Pablo Ferreira – 3480

Samuel Sena – 3494

Florestal – MG

2020

SUMÁRIO

1. Introdução

O trabalho consiste em duas tarefas A e B, que possuem o objetivo de simular uma memória principal em forma de vetor e simular o funcionamento de memória virtual, respectivamente. Ambas tarefas realizadas tendo como base o trabalho prático 2 da disciplina.

Para compilar o programa, criamos um arquivo *makefile*, portanto basta iniciar uma instância de um terminal navegado até a pasta de uma das tarefas (em algum sistema operacional Linux) e compilar o programa com o uso do comando *make*. Em seguida, para executar o programa, é possível com o uso do comando *make run* ou simplesmente: *./EXEC* .

O código se encontra com diversos comentários que auxiliam no entendimento do funcionamento e fluxo de execução.

2. Decisões do Projeto

Para a implementação do trabalho decidimos fazer uma modificação profunda no código do trabalho anterior. Novas estruturas de dados necessitavam ser criadas, além de modificações nas estruturas já existentes. Logo, a realização da tarefa A teve como base o código do trabalho prático II e em seguida a realização da tarefa B foi realizada tendo como base o código obtido na tarefa A.

3. Implementação

Tarefa A

A implementação da tarefa A do trabalho consistiu na adaptação de um vetor estático de tamanho pré definido de tamanho "*MAXMEM"*. Devido a isso, em locais no código onde antes os valores inteiros pertencentes a processos eram simplesmente alocados dinamicamente, definidos como 0 (instrução D) ou ainda operações aritmeticas (instruções A, S), tiveram a implementação modificada e foi chamada a função de alocação no vetor estático de memória. Função esta dividida em duas opções de alocação: FirstFit e NextFit. Para alternar entre os algoritmos de alocação de memória basta ajudar o valor do define FIRSTFIT, quando o valor é definido para 0 é executado o algoritmo NextFit, e quando o valor do define é definido para 1 é executado. Caso um processo tente alocar suas variaveis ou um processo tente criar um processo filho e não

haja espaço para o mesmo criar sua variável, o mesmo será bloqueado e o seu contador de programa não será incrementado (permitindo que a mesma instrução execute na próxima oportunidade de execução). Caso um processo termine, pela execução da instrução T ou simplesmente as instruções acabem, a memória alocada por aquele processo será liberada e um processo anteriormente bloqueado será desbloqueado. Além disso, uma implementação que busca simular um disco (assim como recomendado pelo monitor no fórum de dúvidas) foi implementado e sempre que um processo se encontra bloqueado, tem seu conteudo da memória movido para o vetor de memória em disco. E por fim, novas funções de impressão (para imprimir o vetor de memória) e calculo de metricas do uso de memória foram implementadas. As figuras abaixo ilustram detalhes citados na implementação:

Figura 1

```c
#define MAXTAM 100
#define BUFFER 256
#define MAXMEM 15
#define FIRSTFIT 1 //1 para Firstfit e 0 para NextFit


typedef char Instrucao[20]; //Armazena uma instrução

typedef struct Disco{
    int memoria[MAXMEM*10];
    int mapadebits[MAXMEM*10];
}Disco;

typedef struct FirstFit{
    int memoria[MAXMEM];
    int mapadebits[MAXMEM];
    int nospercorridos;
    int totalalocados;
    int erroemalocar;
    int totalfragmentos;
}FirstFit;

typedef struct NextFit{
    int memoria[MAXMEM];
    int mapadebits[MAXMEM];
    int nospercorridos;
    int ultimaalocacao;
    int totalalocados;
    int erroemalocar;
    int totalfragmentos;
}NextFit;
```

Novas estruturas de dados criadas .

Figura 2

```
if(FIRSTFIT)
  verificador = AlocaFirstFit(cpu->valorInteiro,cpu->Quant_Inteiros,n1,cpu->Alocado_V_inteiros,&cpu->Pos_Alocado);
else
  verificador = AlocaNextFit(cpu->valorInteiro,cpu->Quant_Inteiros,n1,cpu->Alocado_V_inteiros,&cpu->Pos_Alocado);
if(verificador){
  cpu->Alocado_V_inteiros =1; //Foi alocado, porem apenas posição especificada foi inicializada com 0;
  cpu->contadorProgramaAtual++;  //Contador do programa atual so incrementa se instrucao D foi realizada com sucesso
}
else{  //em caso de falta de memoria, processo é bloqueado
  free(cpu->valorInteiro);
  if (FIRSTFIT){
    DesalocaFirstFit(cpu->Quant_Inteiros,cpu->Pos_Alocado);
   }
  else{
    DesalocaNextFit(cpu->Quant_Inteiros,cpu->Pos_Alocado);
   }
  AlocaDisco(cpu->valorInteiro,cpu->Quant_Inteiros,0,0,&cpu->Pos_Disco);
  for(int i = 1;i < cpu->Quant_Inteiros;i++){
    AlocaDisco(cpu->valorInteiro,cpu->Quant_Inteiros,i,1,&cpu->Pos_Disco);
   }
    cpu->Alocado_V_inteiros = 0;
    EnfileiraBloqueado(estadobloqueado, processo);
```

Verificação de espaço disponível durante alocação de variáveis.

Figura 3

```
int AlocaFirstFit(int temp[],int qtd,int n,int flag,int *pos){  // retorna 0 caso haja err
    int espaco = 0;
    if(flag == 0){
        for(int i = 0;i < MAXMEM;i++){
            if(firstfit.mapadebits[i] == 0){
                espaco++;
            }
            else{
                espaco = 0;
            }
            if(qtd == espaco){
                firstfit.memoria[i+1-espaco+n] = temp[n];
                firstfit.mapadebits[i+1-espaco+n] = 1;
                *pos = i+1-espaco;
                firstfit.nospercorridos += *pos;
                firstfit.totalalocados += 1;
                printf("\nAlocado %d em %d\n",temp[n],i+1-espaco+n);
                return 1;
            }
        }
    }
    else{
        firstfit.memoria[*pos+n] = temp[n];
        firstfit.mapadebits[*pos+n] = 1;
        printf("\nAlocado %d em %d\n",temp[n],*pos+n);
        return 1;
    }
    printf("\nERRO! Nao foi possivel alocar o progama devido a falta de espaco\n");
    firstfit.erroemalocar += 1;
    return 0;
}
```

Função de alocação FirstFit.

Figura 4

```
case 'T': /* Termina esse processo simulado. */
    printf("\nProcesso de PID: %d TERMINOU!\n",pcbTable->vetor[estadoexec->iPcbTable].pid);
    if(FIRSTFIT)
      DesalocaFirstFit(cpu->Quant_Inteiros,cpu->Pos_Alocado);
    else
      DesalocaNextFit(cpu->Quant_Inteiros,cpu->Pos_Alocado);
    DesbloqueiaProcesso(estadobloqueado,estadopronto); //Desbloqueando um processo devido a liberação de memoria
    free(cpu->valorInteiro);
    RetiraPcbTable(pcbTable, estadoexec->iPcbTable, processo); // Precisa desalocar o programa.
    *processo = ColocaOutroProcessoCPU(cpu, estadopronto);
    time->time++;
    break;
```

Liberando espaço após termino de execução de processo.

Figura 5

```
int AlocaNextFit(int temp[],int qtd,int n,int flag,int *pos){ // retorna 0 caso haja erro
    int espaco = 0;
    if(flag == 0){
        for(int i = nextfit.ultimaalocacao;i < MAXMEM;i++){
            if(nextfit.mapadebits[i] == 0){
                espaco++;
            }
            else{
                espaco = 0;
            }
            if(qtd == espaco){
                nextfit.memoria[i+1-espaco+n] = temp[n];
                nextfit.mapadebits[i+1-espaco+n] = 1;
                *pos = i+1-espaco;
                nextfit.nospercorridos += *pos;
                nextfit.totalalocados += 1;
                printf("\nAlocado %d em %d\n",temp[n],i+1-espaco+n);
                nextfit.ultimaalocacao= *pos+qtd;
                return 1;
            }
        }
        espaco = 0;
        for(int i = 0;i < nextfit.ultimaalocacao;i++){
            if(nextfit.mapadebits[i] == 0){
                espaco++;
            }
            else{
                espaco = 0;
            }
            if(qtd == espaco){
                nextfit.memoria[i+1-espaco+n] = temp[n];
                nextfit.mapadebits[i+1-espaco+n] = 1;
                *pos = i+1-espaco;
                nextfit.nospercorridos += *pos;
```

Função de alocação NextFit.

Tarefa B

A implementação da tarefa B...

4. Testes de execução

Os testes de execução foram feitos levando em consideração os arquivos de entrada: Controle.txt (arquivo de instruções lido pelo controle), Programa.txt (arquivo de programa 1), Programa2.txt (arquivo de programa 2), Programa3.txt (arquivo de programa 3).

Tarefa A

Ao decorrer da execução, o conteúdo na memória e as filas de processos são impressos na tela, com o intuito de acompanhar o que acontece no programa. Durante a execução abaixo, foi definido *MAXMEM* como 15 e o algoritmo foi executado duas vezes, utilizando em cada vez um algoritmo diferente. A impressão da memória realizada pelo programa exibe todas as posições da memória (Nessa execução, 15 posições), sendo que as posições não alocadas são representadas pelo carácter "#". As figuras abaixo exibem o preenchimento da memória nos algoritmos FirstFit (Figuras a esquerda) e NextFit (Figuras a direita):

Figura 6

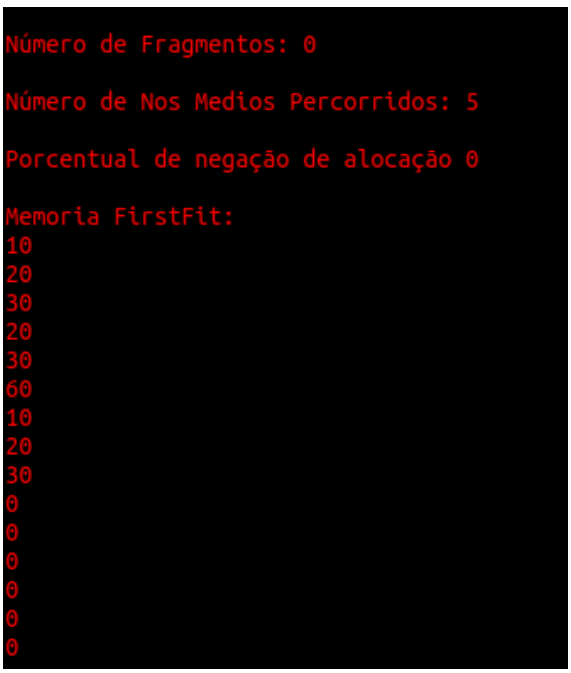
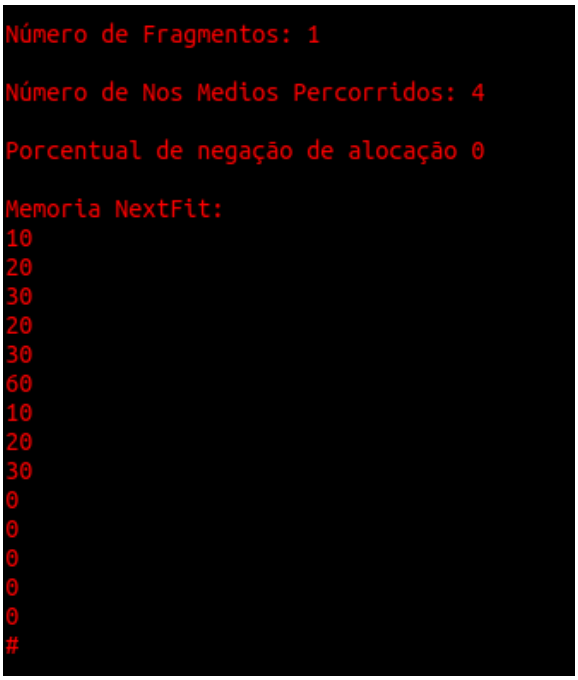


Figura 7



Primeira impressão.

Figura 8

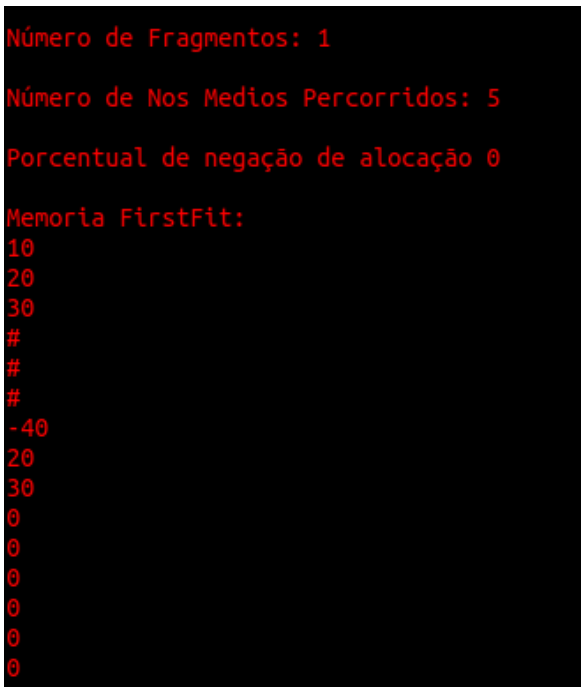
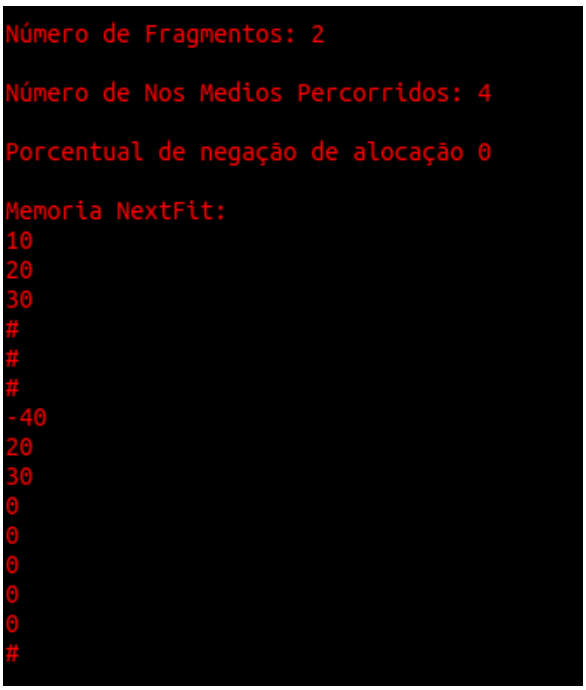


Figura 9



Segunda impressão.

Figura 10

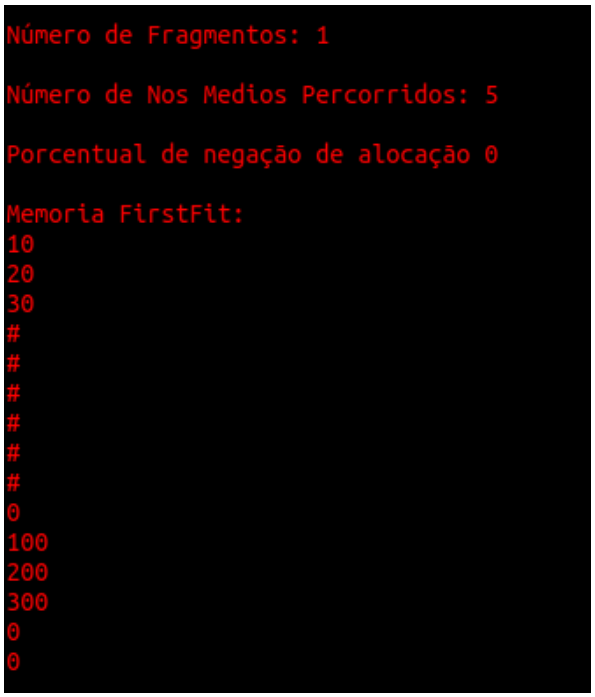
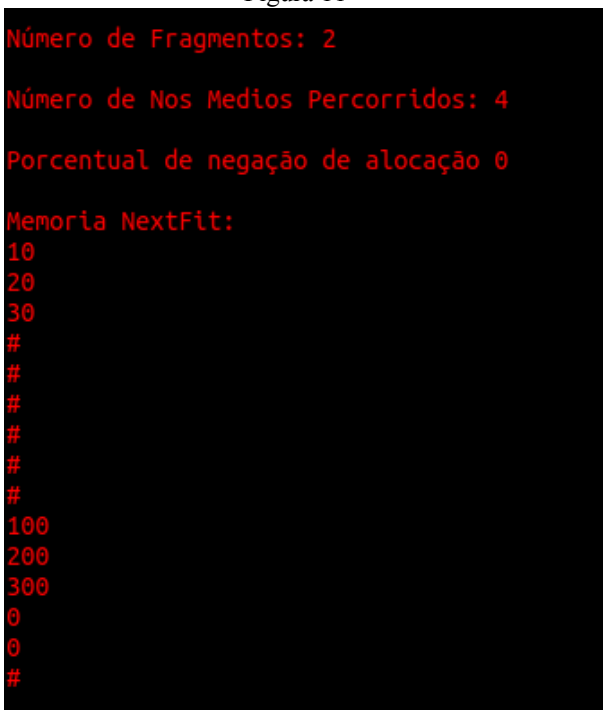


Figura 11



Terceira impressão.

É clara a diferença na política de alocação de memória em cada um dos algoritmos. Lembrando que, na execução aqui demonstrada, o tamanho do vetor de memória havia sido configurado para 15 para uma visualização rápida e fácil com poucos processos. Caso necessário, é possível escolher o tamanho desejado para o vetor de memória, apenas definindo com outro valor o define *MAXMEM* no código.

5. Referência Bibliográfica

TANENBAUM, Andrew S. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson, 2010. 653 p.