Universidade Federal de Viçosa
Campus Florestal
CCF 451 - Sistemas Operacionais
Prof. Daniel Mendes Barbosa

Trabalho Prático 02
Simulação de gerenciamento de processos

Camila Cota Guimarães - 2256
Marcos Vinicius Souza Mota - 2295
Samuel Jhonata Soares Tavares - 2282
Vinícius Kodama Reis - 2259

Florestal
2017

SUMARIO

1. INTRODUÇÃO

Consta neste relatório a documentação do segundo trabalho prático da disciplina de Sistemas Operacionais (SO). Nele apresenta-se o objetivo, o desenvolvimento, a metodologia utilizada e a conclusão que o grupo teve depois do trabalho realizado.

1.1. Objetivo

O objetivo deste trabalho é simular cinco funções do gerenciamento de processos, aprendidos em sala de aula: criar um processo, substituir sua imagem atual com uma nova imagem, transição de estado do processo, escalonamento de processo e troca de contexto.

2. DESENVOLVIMENTO

Após o entendimento da especificação do trabalho, deu-se início ao projeto, criando os tipos abstratos de dados (TADs) necessários, composto por seis tipos, um para cada tipo de processo e um para criação de uma fila de processos.

Cada TAD possui um arquivo “.h”. Cada um desses arquivos possui o escopo das funções de acordo com o TAD escolhido, que são compostos pelas seguintes funções e estruturas:

```
typedef struct Reporter{
  int *pipeEntrada;
  int *pipeSaida;

}Reporter;

void imprimeEstado(Manager *m, char* retorno);
```

Figura 1. Estrutura e escopo das funções do processo *Reporter*

```
typedef struct Commander{
    int pipeEntrada[2];
    int pipeSaida[2];
    Manager manager;
}Commander;

int carregaComandos(String nomeArq, char *retorno);
void inicializaCommander(Commander *c, int tipoEscalonamento, int tipoPreemptivo, int tipoEntrada);
void recebeComandos(Commander *c, int tipoEntrada);
int executaComando(Commander *c, char comando);
```

Figura 2. Estrutura e escopo das funções do processo *Commander*

```
typedef struct Cpu{
    int time;
    int variavel;
    int pc;
    int fatiaTempo; //qtd de tempo máxima para o processo executar
    int tempoUsado; //tempo ja usado na execucao
}Cpu;

typedef struct retornoCPU{
    int comando;
    int n;
    char arquivo[30];
}retornoCPU;

void executaProxInst(Processo *p, Cpu *cpu, retornoCPU* retorno);
void executaInstrucao(Processo *p, String instrucao, Cpu *cpu, retornoCPU* retorno);
```

Figura 3. Estrutura e escopo das funções da CPU

```c
typedef int TipoChave;

typedef struct {
    TipoChave indiceProcesso;
    int prioridade;
} TItem;

typedef struct Celula* Apontador;

typedef struct Celula {
    TItem Item;
    struct Celula* pProx; /* Apontador pProx; */
} TCelula;

typedef struct {
    Apontador pPrimeiro;
    Apontador pUltimo;
    int Contador;
} TLista;


void FLVazia(TLista* pLista);
int LEhVazia(TLista* pLista);
void LInsere(TLista *pLista,TItem* pItem);
int LRetira (TLista* pLista, int index);
void LImprime(TLista* pLista);
void LBuscaPrimeiro (TLista* pLista, int *indiceProcesso, int *prioridade);
```

Figura 4. Estrutura e escopo das funções da lista de processos

```c
typedef struct Manager{
    int *pipeEntrada;
    int *pipeSaida;
    int pipeRepEntrada[2];
    int pipeRepSaida[2];
    int time;
    Cpu cpu;
    Processo* pcbTable[QTD_PROCESSOS_MAX]; //vetor de processos
    TLista readyState;
    TLista blockedState;
    int runningState;
    int contadorID;
    int ultimoExecutado;
    int tempoMedio;
    int qtdProcessosEncerrados;
} Manager;

void inicializaManager(Manager *m, int tipoEscalonamento, int tipoPreemptivo);
void recebeComandosManager(Manager *m, int tipoEscalonamento, int tipoPreemptivo);
void inicializaEstruturas(Manager *m);
int criaProcesso(Manager *m, int id, int prioridade, String nomeArq);
void criaProcessoFilho(Manager *m, Processo *pai);
int insereListaPronto(Manager *m, int indice, int prioridade);
int insereListaBloqueado(Manager *m, int indice, int prioridade);
void trocaContexto(Manager *m, int fatiaTempo, int indexProcessoColocar);
int escalonar(Manager *m, int tipoEscalonamento);
```

Figura 5. Estrutura e escopo das funções do processo *Manager*

```c
typedef char* String;
typedef int Estado;

typedef struct Processo{
    int id; //id do processo
    int idPai;
    int pc; //contador de programa
    int variavel; //variavel a ser manipulada
    int prioridade;
    Estado estadoAtual; //estado atual do processo
    int tempoInicio;
    int tempoCPU;
    char texto[TAM_TEXTO][20]; //codigo do programa
    int qtdInst; //conta total de comandos

}Processo;

int carregaInstrucoes(Processo *p, String nomeArq);
```

Figura 6. Estrutura e escopo da função do *Processo*

Cada TAD também possui um arquivo para implementação das funções declaradas acima.

2.1. Descrição das funções

2.1.1. Funções do processo *Commander*

O *Commander* é o processo que inicializa a simulação do gerenciamento. Esse tem as seguintes funções implementadas para que funcione de forma correta: *inicializaCommander()*, *recebeComandos()* e *executaComando()*. O funcionamento dessas funções será descrito a seguir.

A função "*int carregaComandos(String nomeArq, char *retorno);*" é usada para carregar os comandos de um arquivo, caso a opção de entrada seja por arquivo.

Na função "*void inicializaCommander(Commander *c);*" cria-se o *pipe*, que será usado para fazer a comunicação entre os processos *Commander* e *Manager*, cria-se também, com a função *fork()* um processo filho, que rodará o *Manager*. Caso o processo seja um filho a função "*inicializaManager();"* é chamada. Caso seja um processo pai, chama a função "*recebeComando();"*.

A função "*void recebeComandos(Commander *c, int tipoEntrada);*" é responsável por receber a entrada (seja por arquivo ou pelo terminal), conferindo se a entrada é válida, e, quando válida, a envia para a função *executaComando.*

A função "*int executaComando(Commander *c, char comando);*" pega cada comando e envia para o processo *Manager*, caso seja do tipo Q ou U, dispara um o

Reporter, quando é do tipo P, ou finaliza o sistema imprimindo o tempo médio do ciclos, caso seja do tipo T.

2.1.2. Funções do processo *Manager*

O processo Manager vai executar os comandos recebidos do processo Commander, de acordo com o tipo de escalonamento escolhido pelo usuário.

*A função "void inicializaManager(Manager *m, int tipoEscalonamento, int tipoPreemptivo);" chama a função inicializaEstruturas(), depois cria o primeiro processo simulado, com prioridade 3, carregado de "programa1.txt", chama a trocaContexto() para colocar o processo criado na CPU e, por fim, chama a função "recebeComandosManager();".*

A função "*void recebeComandosManager(Manager *m, int tipoEscalonamento, int tipoPreemptivo);*" tem um loop "infinito", para receber comandos, que termina quando recebe um comando do tipo T.

Quando o comando é do tipo Q, olha se há processo em execução, e, caso exista, executa a próxima instrução. Se a instrução for do tipo F, cria um novo processo filho e pula N instruções no processo pai. Também verifica se é preciso escalonar (de acordo com o tipo de escalonamento). Além disso, incrementa o tempo de execução do sistema.

Quando o comando é do tipo U, retira o primeiro processo da lista de bloqueados, caso não esteja vazia e o insere na lista de prontos, mudando seu estado atual para Pronto.

Quando o comando é do tipo P, chama a imprimeEstado() da Reporter, que retorna o texto de informações, passando isso para o *Commander* printar na tela (através do pipe).

Quando o comando é do tipo T, calcula a média do ciclo, envia para a *Commander (*através do pipe) e finaliza o sistema.

"*void inicializaEstruturas(Manager *m);*" é utilizada para limpar a CPU e inicializar as listas de bloqueado e pronto.

*"int criaProcesso(Manager *m, int id, int prioridade, String nomeArq);"* aloca um novo processo na memória, seta os seus valores, carrega as instruções no seu texto e percorre pcbTable para achar a primeira posição vazia, onde é salvo.

"*void criaProcessoFilho(Manager *m, Processo *pai);*" aloca um novo processo, seta os seus valores de acordo com os valores do pai, modificando apenas o seu estado atual para Pronto, o id do pai e o tempo de CPU para 0, além de inseri-lo na PcbTable.

"*int insereListaPronto(Manager *m, int indice, int prioridade);*" cria um novo item, com índice e prioridade, e o adiciona na lista de Pronto.

"*int insereListaBloqueado(Manager *m, int indice, int prioridade);*" cria um novo item, com indice e prioridade, e o adiciona na lista de Bloqueado.

"*void trocaContexto(Manager *m, int fatiaTempo, int indexProcessoColocar);*" atualiza as informações do processo que estava em execução na CPU em PcbTable, caso houvesse um e coloca o novo processo na CPU, retirando-o da lista de Prontos.

2.1.2.1. Funcionamento da CPU

A CPU é a responsável por executar os processos, no caso, um processo por vez, que é escalonado e colocado na CPU quando há troca de contexto. Esses processos ficam salvos no vetor *PcbTable*.

A função "*void executaProxInst(Processo *p, Cpu *cpu, retornoCPU* retorno)*;" verifica se há instrução do processo que está na CPU para executar, se sim, vai na instrução do processo na linha do PC e envia para a função "*executaInstrucao();*".

A função "*void executaInstrucao(Processo *p, String instrucao, Cpu *cpu, retornoCPU* retorno);*" é utilizada para executar as instruções S, A, D, B, E, F e R de cada programa (no caso, um processo carregado). Nos casos em que a instrução é do tipo B (bloqueio), E (término), F (cria filho) ou R (substitui a imagem), é necessário retornar para a *Manager* que estas instruções precisam ser executadas, através do *retornoCPU*, que é uma estrutura com 02 inteiros para salvar o tipo do comando (definidos) e N, o número que foi escrito na instrução, bem como um vetor de caracteres com o nome do arquivo (usado quando é instrução tipo R).

A função *"int quebraNumero(String texto)*;*"* é usada para pegar o valor N da instrução. Para isso, ignora-se as duas primeiras posições do vetor de caracteres (que contém a instrução e o espaço) e utiliza apenas os próximos caracteres (até o final), pegando cada posição, da esquerda para a direita, transformando o caractere em um inteiro (subtrai por 48 por causa da tabela ASCII) e, somando ao valor que já havia sido calculado anteriormente (multiplicado por 10, uma vez que aumentou uma casa nas unidades).

2.1.3. Função do processo *Reporter*

O processo *Reporter* tem apenas uma função, a "*void imprimeEstado(Manager *m, char* retorno);*". Ela é utilizada para enviar para o *Manager* o estado atual do sistema e depois o *Manager* envia para o *Commander*.

O *Reporter* imprime informações dos processos presentes na PcbTable (id, id pai, prioridade, pc, tempo início, tempo de cpu e variável), o processo em execução (runningState), os processos nas filas de pronto(readyState) e bloqueado (blockedState) e informações da CPU (variável, tempo de uso e fatia de tempo).

2.1.4. Funções do *Processo Simulado*

O *P*rocesso é utilizado na lista *PcbTable*, e contém todas as informações para executar o programa.

A função "*int carregaInstrucoes(Processo *p, String nomeArq);*" é utilizada para carregar o código de instruções do arquivo do programa. Ela chama "*insereInstrucao();"* para colocar o texto no processo.

*A função "void informacoesProcesso(Processo *p);*" é utilizada para imprimir o estado atual do processo.

*A função "void insereIntrucao(Processo *p, String instrucao);*" é utilizada para inserir uma instrução no vetor de instruções (matriz de caracteres) texto, no processo, além de incrementar a quantidade de instruções para se fazer o controle posteriormente.

2.1.5. Funções do *TAD Fila*

A função "*void FLVazia(TLista* pLista);*" é utilizada para criar uma nova lista, com cabeça, alocando espaço na memória, onde a primeira posição da lista recebe a última posição da lista (lista vazia).

A função "*int LEhVazia(TLista* pLista);*" é utilizada para verificar se a lista é vazia ou não, o que acontece quando o primeiro e o último apontam para a mesma célula.

A função "*void LInsere (TLista *pLista,TItem* pItem);*" faz a alocação de um espaço na memória para fazer a inserção do item, inserindo-o no final.

"*int LRetira (TLista* pLista, int index);*" é utilizada para retirar o item da lista que contém o index desejado, retornando 1 em caso de sucesso, e 0 quando não é possível (lista vazia ou não existe o item com aquele index).

Na função "*void LImprime(TLista* pLista);*" imprime a quantidade de itens que tem na lista e os índices dos processos que estão na lista.

A função "*void LBuscaPrimeiro (TLista* pLista, int *indiceProcesso, int *prioridade)*" busca o primeiro processo de maior prioridade na lista, o retira e retorna o índice e a prioridade através dos atributo que são passados, para depois criar um novo item.

2.1.6. *Main*

Concluímos o programa com um único arquivo “main.c” que foi criado de acordo com a implementação dos TADs.

O programa principal funciona da seguinte maneira: uma sequência de menus foi criada, o primeiro é o menu onde o usuário pode escolher o tipo de escalonamento que será usado (lista de prioridade ou relógio - implementado pelo grupo), o menu seguinte é usado para escolher o tipo de escalonamento (preemptivo ou não preemptivo) e o terceiro e último menu é usado para saber o tipo de entrada dos comandos (prompt de comando ou arquivo).

Ao final, o *main* chama a função “*inicializaCommander();*”, que irá executar todo o programa.

2.2. Decisões tomadas pelo grupo

2.2.1. Escalonamento

Escolhemos dois tipos de escalonamento, escalonamento por prioridade e escalonamento por relógio.

A descrição do trabalho pedia para que implementássemos dois tipos de escalonamento, sendo um deles já descrito pelo trabalho e um segundo a escolha do grupo. Também decidimos que iríamos implementar os dois escalonamentos tanto preemptivo como não preemptivo, ou seja, se eles iriam executar até concluir/bloquear ou se poderiam ser escalonados após usar sua fatia de tempo.

O primeiro escalonamento, sendo o descrito pelo trabalho, foi o escalonamento por prioridade. Cada processo foi atribuído uma prioridade de 0 a 3, sendo 3 a mais alta, caso o processo fosse bloqueado antes do seu tempo de execução, sua prioridade diminui, caso o processo fosse escalonado antes do seu tempo de execução, sua prioridade aumenta. Este escalonamento por prioridade funciona da seguinte forma: a partir da lista de processos prontos para executar, procura o primeiro processo de maior prioridade e troca este processo de lugar com o processo sendo atualmente executado pela CPU. O processo retirado da CPU é inserida no final da lista de processos prontos, assim um ciclo entre processos em execução é criado. Caso não tenha processo para ser escolhido, a CPU ficará ociosa até que um processo seja desbloqueado e inserido na lista de processos prontos.

O segundo escalonamento, o desenvolvido pelo grupo, chamamos de escalonamento de relógio, bastante parecido com o escalonamento de páginas que aprendemos durantes as aulas, contudo para os processos. Funciona da seguinte forma: ao iniciar, ele busca o primeiro processo pronto da lista de processos, não da lista de processos

prontos, e guarda o índice desse processo, então ao ser chamado para escalonar, o relógio continua sua procura a partir do índice do último processo que estava em execução, caso chegue no final de lista, ele retornará ao início da mesma e continuará procurando. Ao encontrar um processo pronto disponível, ele fará a troca de contexto e atualizará qual o último processo em execução, o processo retirado volta ao estado de pronto. Se não houver nenhum processo pronto, a CPU ficará ociosa até que um processo seja desbloqueado.

Neste segundo escalonamento, a lista de processos prontos acaba sendo desnecessária, pois o que importa é qual o “slot” do vetor de processos o processo a ser executado ocupa. Pois à medida que processos finalizam, seu slot no vetor de processos é liberado, um novo processo ao ser criado preencherá o primeiro espaço vago que encontrar.

2.1.2. Fork

A *fork()* foi usada apenas no processo *Commander* e no *Manager*. Não foi possível usar para o *Reporter*, pois o *pipe* estava dando lixo de memória por causa de um estouro na pilha.

2.1.3. Leitura por terminal ou arquivo

As entradas da *Commander* podem ser feitas pela entrada do terminal ou por um arquivo. O qual os comandos são lidos de acordo com o tempo da entrada.

2.1.4. Comunicação entre *Commander* e *Manager*

A comunicação entre *Commander* e *Manager* é através dos *pipes*.

2.2. PrintScreens

Figura 7. Menu principal do programa com as funcionalidades já selecionadas

```
1- Preemptivo
2- Não Preemptivo

1
Tipo Entrada de Comandos:
1- Prompt de Comandos
2- Arquivo
1

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 0
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 3; PC: 0; Tempo Início: 0; Tempo CPU: 0; Variavel: 0
PROCESSO EM EXECUÇÃO:
        0 - ID:0; ID Pai: -1; Prioridade: 3; Variavel: 0
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 0
        Variável: 0
        Tempo de Uso: 0
        Fatia de tempo: 8
----------------------------------------------------------------


usuario@computador:~$
```

Figura 8. Mostra que o processo foi criado pela *Manager*

```
usuario@computador:~$  q

Comando Q

Executando instrução: S 1000

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 1
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 3; PC: 0; Tempo Início: 0; Tempo CPU: 0; Variavel: 0
PROCESSO EM EXECUÇÃO:
        0 - ID:0; ID Pai: -1; Prioridade: 3; Variavel: 0
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 1
        Variável: 1000
        Tempo de Uso: 1
        Fatia de tempo: 8
----------------------------------------------------------------


usuario@computador:~$
```

Figura 9. Primeiro comando Q e mostra estado atual do sistema

```
Comando Q

Executando instrução: D 50

Escalonando rodando agora 1

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 8
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 3; PC: 10; Tempo Início: 0; Tempo CPU: 8; Variavel: 1060
        1 - ID:1; ID Pai: 0; Prioridade: 3; PC: 3; Tempo Início: 2; Tempo CPU: 0; Variavel: 1010
PROCESSO EM EXECUÇÃO:
        1 - ID:1; ID Pai: 0; Prioridade: 3; Variavel: 1010
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
                0 - ID:0; ID Pai: -1; Prioridade: 3; Variavel: 1060
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 3
        Variável: 1010
        Tempo de Uso: 0
        Fatia de tempo: 8
--------------------------------------------------------------------


usuario@computador:~$
```

Figura 10. Vários comandos Q, até acontecer o primeiro escalonamento

```
usuario@computador:~$  q

Comando Q

Executando instrução: E

Escalonando rodando agora 0

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 15
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 3; PC: 10; Tempo Início: 0; Tempo CPU: 8; Variavel: 1060
PROCESSO EM EXECUÇÃO:
        0 - ID:0; ID Pai: -1; Prioridade: 3; Variavel: 1060
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 10
        Variável: 1060
        Tempo de Uso: 0
        Fatia de tempo: 8
--------------------------------------------------------------------


usuario@computador:~$
```

Figura 11. Primeiro processo é finalizado

```
Comando Q

Executando instrução: B

Escalonando rodando agora 1

usuario@computador:~$  p

Comando P


-----------------------INFORMAÇÕES DO SISTEMA-----------------------

Tempo do Sistema: 21
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 2; PC: 18; Tempo Início: 0; Tempo CPU: 14; Variavel: 1100
        1 - ID:2; ID Pai: 0; Prioridade: 3; PC: 11; Tempo Início: 15; Tempo CPU: 0; Variavel: 1060
PROCESSO EM EXECUÇÃO:
        1 - ID:2; ID Pai: 0; Prioridade: 3; Variavel: 1060
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
                0 - ID:0; ID Pai: -1; Prioridade: 2; Variavel: 1100
        Processos com prioridade 3
Informações da CPU:
        PC 11
        Variável: 1060
        Tempo de Uso: 0
        Fatia de tempo: 8
--------------------------------------------------------------------


usuario@computador:~$ 
```

Figura 12. Processo bloqueado

```
usuario@computador:~$  q

Comando Q

Executando instrução: E

Escalonando rodando agora -1

usuario@computador:~$  p

Comando P


-----------------------INFORMAÇÕES DO SISTEMA-----------------------

Tempo do Sistema: 28
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 2; PC: 18; Tempo Início: 0; Tempo CPU: 14; Variavel: 1100
PROCESSO EM EXECUÇÃO:
        Nenhum processo em execução
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
                0 - ID:0; ID Pai: -1; Prioridade: 2; Variavel: 1100
        Processos com prioridade 3
Informações da CPU:
        PC 5
        Variável: 10100
        Tempo de Uso: 7
        Fatia de tempo: 8
--------------------------------------------------------------------


usuario@computador:~$ 
```

Figura 13. CPU ociosa pois não tem processo pronto

```
        Fatia de tempo: 8
-----------------------------------------------------------------


usuario@computador:~$  u

Comando U

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 28
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 2; PC: 18; Tempo Início: 0; Tempo CPU: 14; Variavel: 1100
PROCESSO EM EXECUÇÃO:
        Nenhum processo em execução
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
                0 - ID:0; ID Pai: -1; Prioridade: 2; Variavel: 1100
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 5
        Variável: 10100
        Tempo de Uso: 7
        Fatia de tempo: 8
-----------------------------------------------------------------


usuario@computador:~$
```

Figura 14. Desbloqueia primeiro processo da lista de bloqueados com o comando U

```
Comando Q

usuario@computador:~$  u

Comando U

Lista de processos bloqueados vazia, impossível desbloquear processo

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 28
Processos:
        0 - ID:0; ID Pai: -1; Prioridade: 2; PC: 18; Tempo Início: 0; Tempo CPU: 14; Variavel: 1100
PROCESSO EM EXECUÇÃO:
        0 - ID:0; ID Pai: -1; Prioridade: 2; Variavel: 1100
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 18
        Variável: 1100
        Tempo de Uso: 0
        Fatia de tempo: 4
-----------------------------------------------------------------


usuario@computador:~$
```

Figura 15. Lista de processos bloqueados vazia

```
Escalonando rodando agora 0

usuario@computador:~$  q

Comando Q

Executando instrução: E

Escalonando rodando agora -1

usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 33
Processos:
PROCESSO EM EXECUÇÃO:
        Nenhum processo em execução
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 22
        Variável: 1140
        Tempo de Uso: 1
        Fatia de tempo: 8
----------------------------------------------------------------------


usuario@computador:~$
```

Figura 16. Todos os processos encerrados

```
usuario@computador:~$  p

Comando P


------------------------INFORMAÇÕES DO SISTEMA------------------------

Tempo do Sistema: 33
Processos:
PROCESSO EM EXECUÇÃO:
        Nenhum processo em execução
PROCESSOS PRONTOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
PROCESSOS BLOQUEADOS:
        Processos com prioridade 0
        Processos com prioridade 1
        Processos com prioridade 2
        Processos com prioridade 3
Informações da CPU:
        PC 22
        Variável: 1140
        Tempo de Uso: 1
        Fatia de tempo: 8
----------------------------------------------------------------------


usuario@computador:~$  t

Comando T
Terminando Manager
SISTEMA ENCERRADO!!!
Tempo Médio Total da cpu: 24.333334

Process returned 0 (0x0)   execution time : 285.380 s
Press ENTER to continue.
```

Figura 17. Sistema finalizado. Mostra o tempo médio da CPU

2.2.1. Comparando as implementações de escalonamento

Com os quatro tipos de escalonamentos possível implementados, resolveu-se fazer a comparação deles para analise do tempo médio total da CPU como pode ser visto nas imagens abaixo:

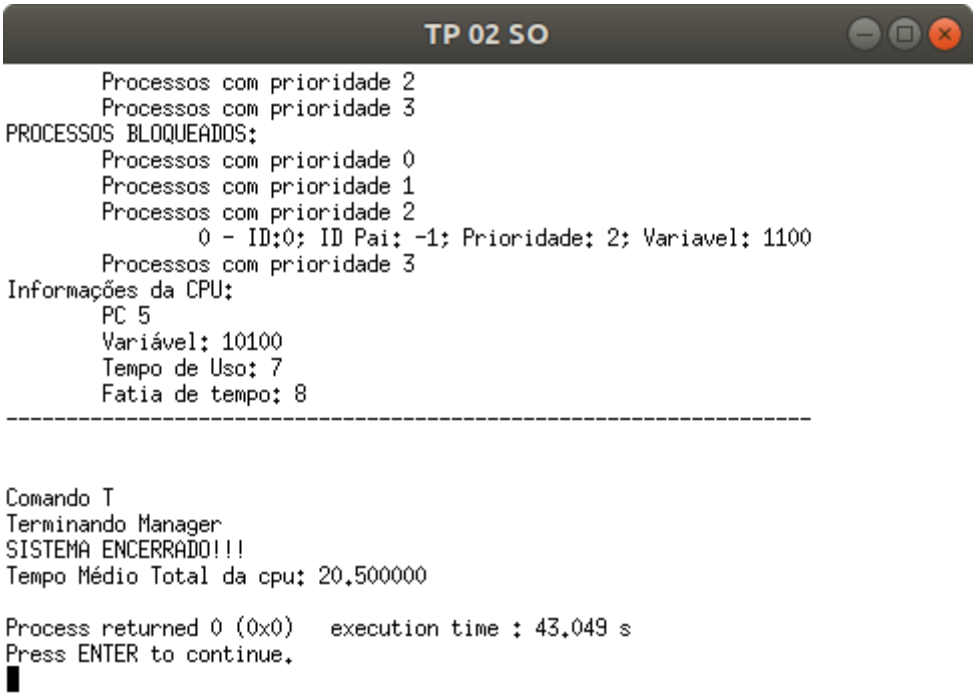

Figura 18. Tempo médio total da CPU para Lista de Prioridade Preemptiva

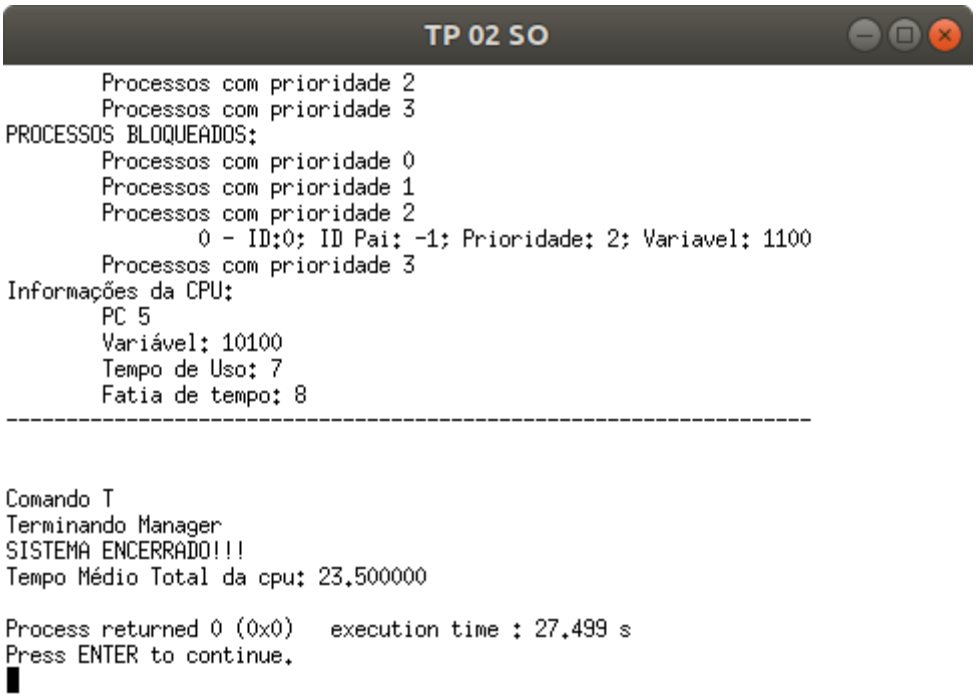

Figura19. Tempo médio total da CPU para Lista de Prioridade Não Preemptiva

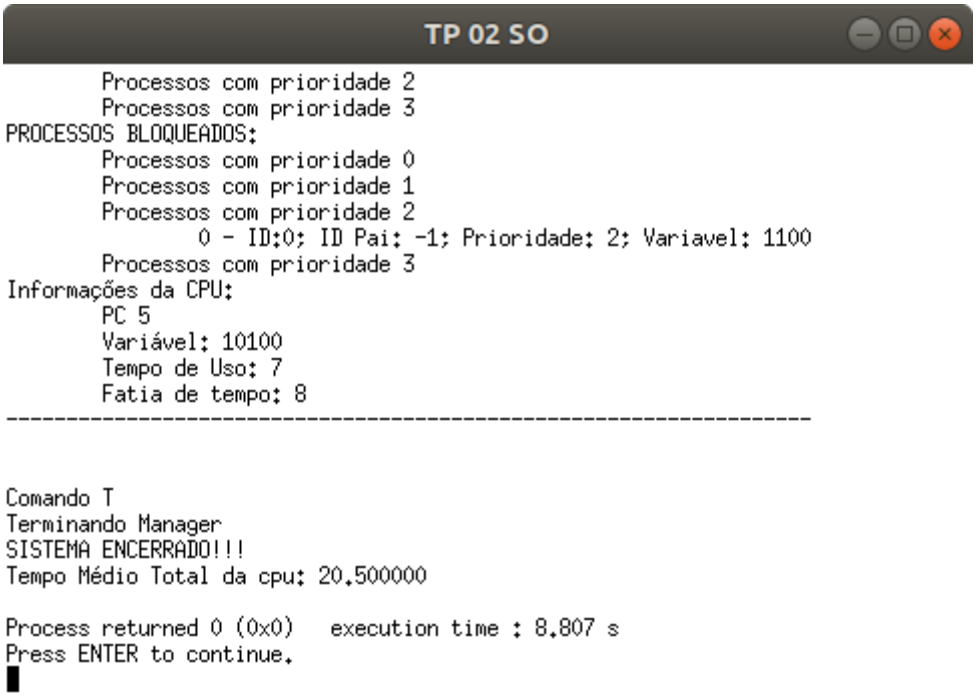

Figura 20. Tempo médio total da CPU para Relógio Preemptiva

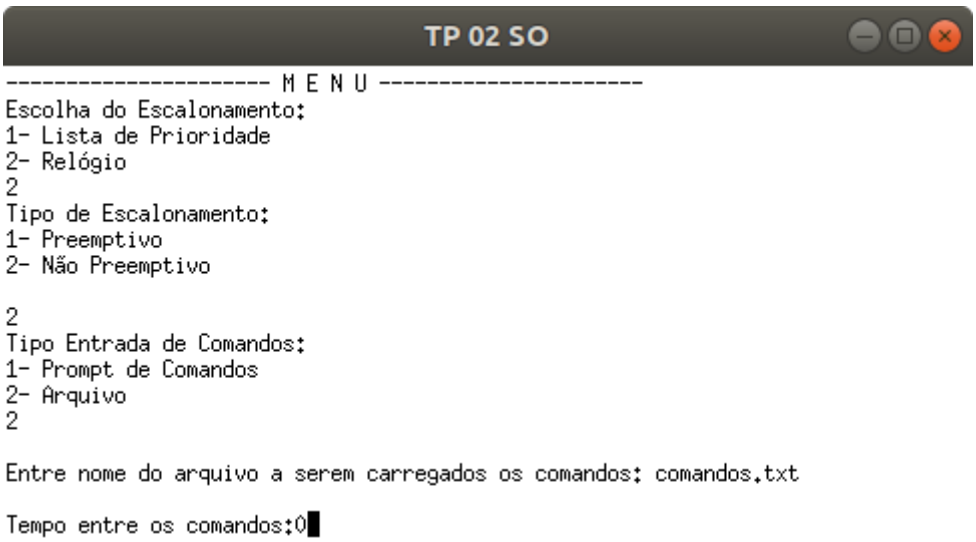

Figura 21. Tempo médio total da CPU para Relógio Não Preemptiva

Com o conjunto de programas utilizados, percebeu-se que o tipo de escalonamento preemptivo foi mais eficiente em ambos os casos, relógio e lista de prioridades. Porém, não é possível afirmar para todos os casos de testes com outros comandos e programas executando se a afirmação continuaria sendo verdadeira.

3. METODOLOGIA

Para melhor entendimento do funcionamento das funções foi lido detalhadamente à descrição do trabalho prático.

Para o desenvolvimento do programa foi usado o *CodeBlocks* para compilar e o *Atomic* para facilitar a programação de mais de uma pessoa do grupo ao mesmo tempo. Todo o trabalho foi feito em linguagem de programação *C* e foi usado o Sistema Operacional *Linux* para compilação e testes do programa.

4. CONCLUSÃO

Com a realização desse trabalho pratico, adquirimos habilidades das quais são essenciais para a disciplina de Sistemas Operacionais. Com isso aprimoramos os nossos conhecimentos gerais sobre o gerenciamento de processos em um Sistema Operacional.

REFERÊNCIAS

Tanenbaum, A. Sistemas Operacionais Modernos, 3ª Edição, Editora Pearson Prentice Hall, 2010.