Universidade Federal De São João Del Rei

Campus Tancredo De Almeida Neves

SISTEMAS OPERACIONAIS
SIMULADOR DE PROCESSOS

Alunos: Alexandre Antônio Lima da Silva - 192050029
Júlio César de Sousa - 192050049

Professor: Rafael Sachetto

Minas Gerais
31/10/2021

1 Introdução **3**
1.1 Como compilar e executar o programa 4
1.2 Exemplos de entrada 4
1.3 Exemplos de saída 5

2 Process Commander **10**
2.1 Funcionalidades **10**

3 Process Manager **10**
3.1 Funções 10
3.1.1 int conta_linhas(char* arquivo); 10
3.1.2 CPU criar(char *arquivo); 10
3.1.3 void insere_tabelapcb(CPU *cpu); 11
3.1.4 void executa_processo_simulado(); 11
3.1.5 void teste_escalonador (int tempo); 11
3.1.6 void escalonar (); 11
3.1.7 celula_pcb *busca(CPU *cpu); 11
3.1.8 void bloqueia(); 11
3.1.9 void encerra(); 11
3.1.10 void reporter(); 11

4 Fila **11**
4.1 void adiciona(Fila **fila, CPU *id); 12
4.2 Fila *remover(Fila **fila); 12

5 Estruturas de dados e variáveis globais **12**
5.1 Estrutura de Dados 12
5.1.1 Struct programa 12
5.1.1 Struct CPU 12
5.1.1 Struct celula_pcb 12
5.1.1 Struct tabela_pcb 12
5.1.1 Struct Fila 12
5.2 Variáveis globais 12
5.2.1 int t = 0; 12
5.2.2 CPU *atual; 12
5.2.3 Fila *e_pronto = NULL; 13
5.2.4 Fila *e_bloqueado = NULL; 13
5.2.4 tabela_pcb *t_pcb = NULL; 13

6 Conclusão **13**

7 Bibliografia **13**

1 Introdução

O intuito do trabalho era a implementação de um simulador de um gerenciador de processos dividido em módulos, dos quais podemos chamá-los de: *commander*, encarregado de criar um *pipe*, um processo do tipo *process manager*, de forma que, fique lendo comandos repetidamente, um comando a cada segundo. Esses comandos são passados ao *process manager* através do *pipe*. Esse processo simulado fica encarregado de executar instruções predefinidas. Já o outro módulo é o *Process Manager* que fica encarregado do gerenciamento dos processos.

Após a execução do programa, era necessário que houvesse um *Reporter* para fazer prints com status dos processos executados.

1.1 Como compilar e executar o programa

Para compilar o programa em Linux, basta que o usuário entre no terminal e siga os comandos descritos abaixo:

Para compilar o código utilize o comando:
make all

E o executável será chamado de "commander".
Para iniciar o programa basta usar o comando:
./commander < entrada

É imprescindível que o usuário se certifique que no diretório em que está tentando fazer a execução do programa estejam presentes os arquivos:

- fila.c
- fila.h
- commander.c
- manager.c
- manager.h
- programa_a
- programa_b
- init
- entrada

Caso algum desses arquivos estejam em falta, por favor verificar o .zip disponibilizado para utilização do software.

1.2 Exemplos de entrada

Abaixo podemos ver exemplos de arquivos de entrada, os que serão visualizados aqui, são os arquivos disponibilizados no .zip. Lembrando que é de suma importância que em todos esses arquivos a última linha seja vazia.

```
1 Q
2 P
3 Q
4 P
5 Q
6 P
7 Q
8 P
9 Q
10 P
11 Q
12 P
13 Q
14 P
15 Q
16 P
17 Q
18 P
19 Q
20 P
21 Q
22 P
23 Q
24 P
25 U
26 P
27 Q
28 P
29 Q
30 T
31
```

Imagem 1 - “entrada”

```
1 S 10
2 A 5
3 F 1
4 R programa_a
5 A 1
6 D 10
7 E
8
```

Imagem 2 - “init”

```
1 S 100
2 F 1
3 R programa_b
4 A 20
5 D 10
6 E
7
```

Imagem 3 - “programa_a”

```
1 S 1000
2 A 50
3 B
4 E
5
```

Imagem 4 - “programa_b”

1.3 Exemplos de saída

Abaixo podemos observar alguns exemplos de saída, tendo como entrada os arquivos descritos acima.

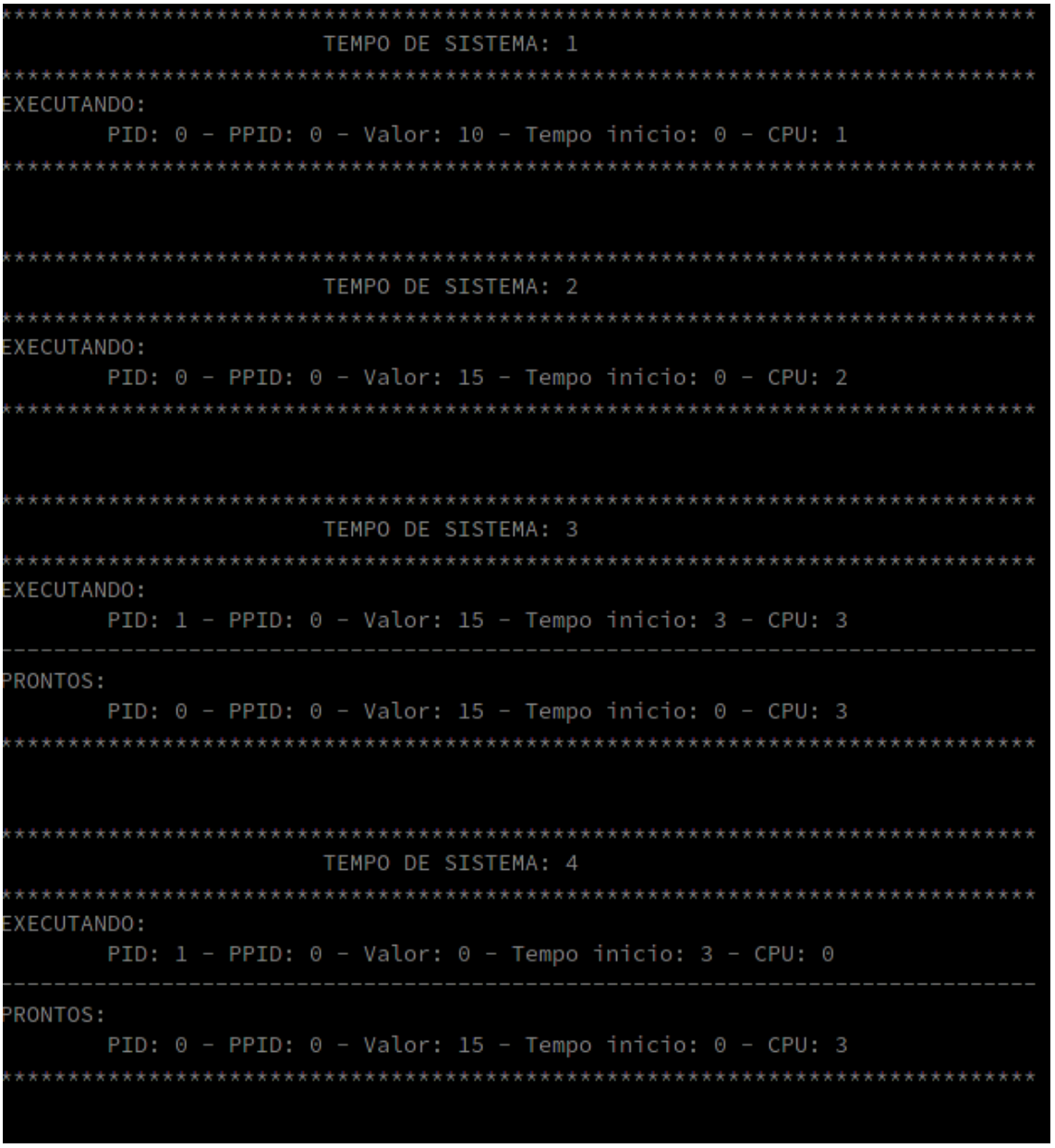

Imagem 1 - Exemplo de saída

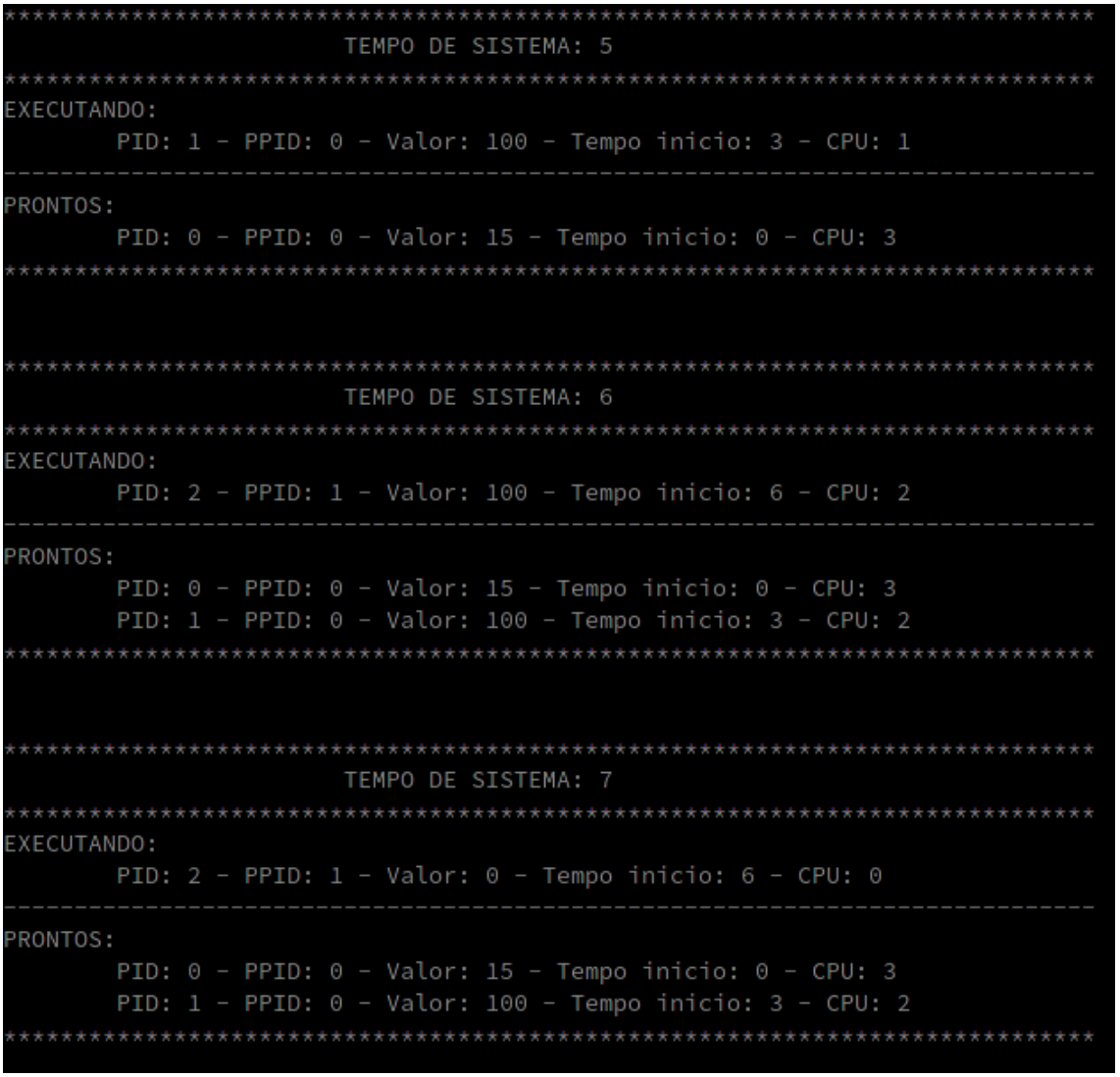

Imagem 2 - Exemplo de saída

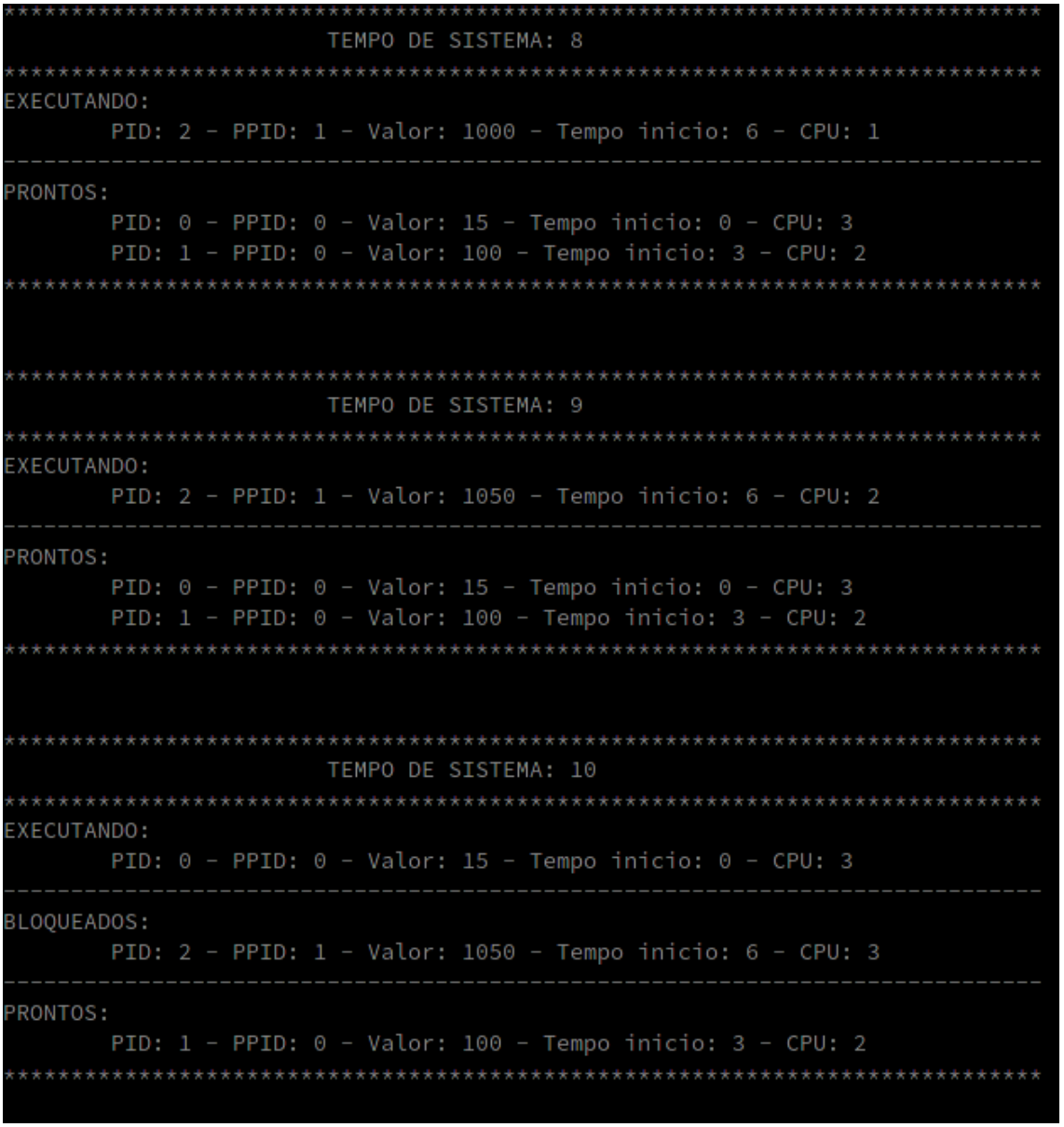

Imagem 3 - Exemplo de saída

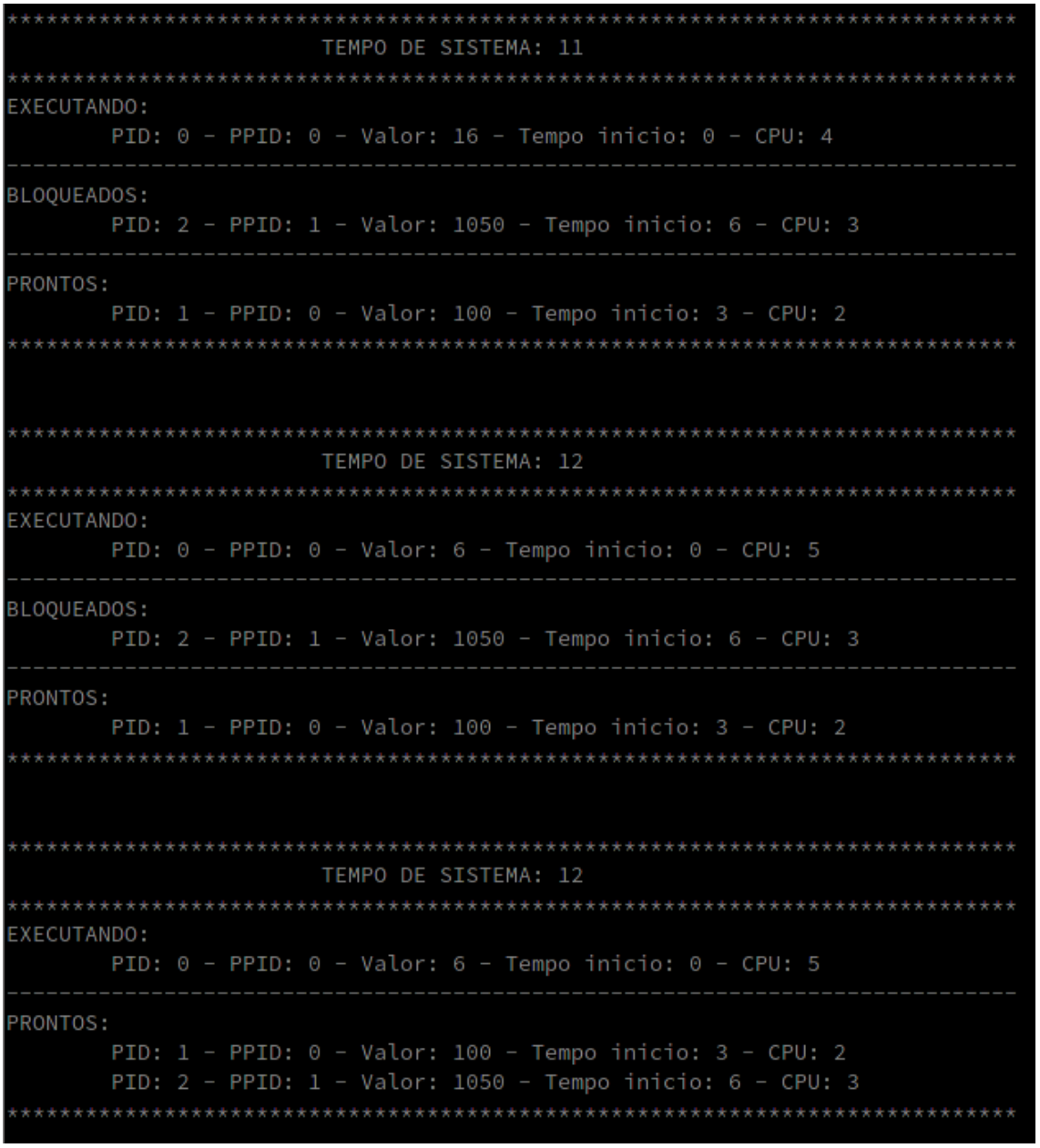

Imagem 4 - Exemplo de saída

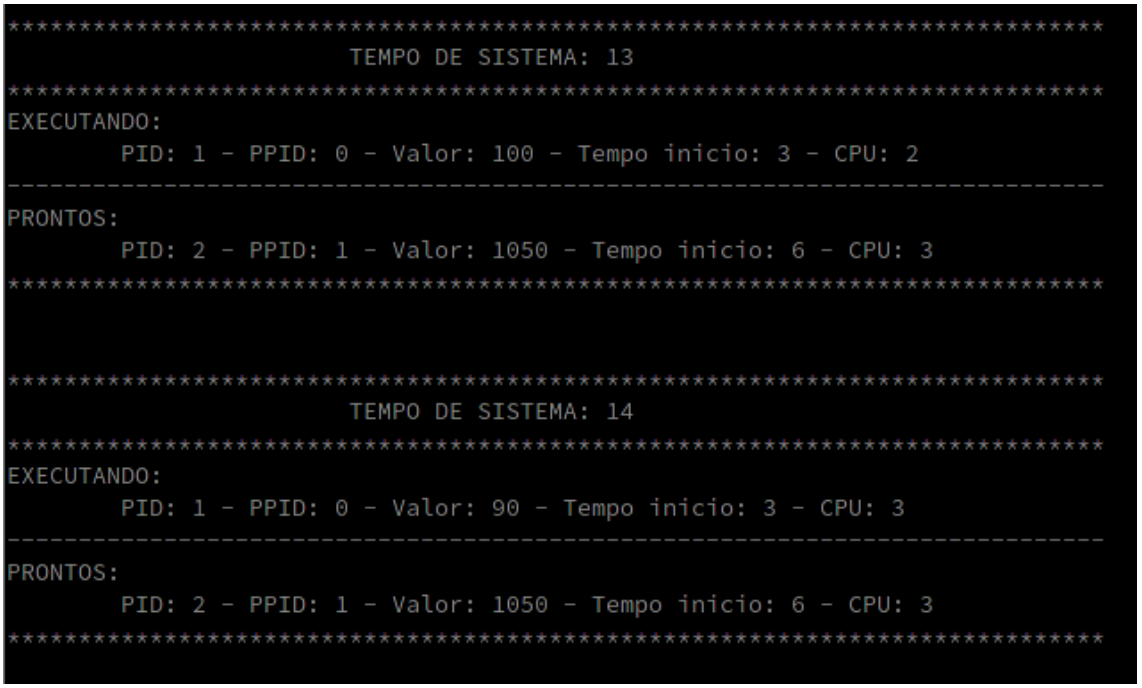

Imagem 5 - Exemplo de saída

2 Process Commander

2.1 Funcionalidades

O commander é semelhante ao arquivo disponibilizado no portal didático da disciplina de Sistemas Operacionais, ministrada pelo professor Rafael Sachetto. A alteração que fizemos foi em um Else em que ele utilizava um while para capturar os comandos, transformamos essa operação em um do while e renomeamos o nome da variável que armazena o comando.

3 Process Manager

3.1 Funções

3.1.1 int conta_linhas(char* arquivo);

Função que conta o número total de linhas do arquivo passado como parâmetro. Sua complexidade é dada por O(n).

3.1.2 CPU criar(char *arquivo);

Função que cria e inicializa uma struct CPU, em seguida lê o arquivo passado como parâmetro linha por linha e guarda todos os comandos em “array_programas”. Sua complexidade é dada por O(n).

3.1.3 void insere_tabelapcb(CPU *cpu);

Caso a tabela PCB esteja vazia, a CPU passada por parâmetro é inicializada na posição 0 da tabela, caso contrário é inserida na posição seguinte.Sua complexidade é dada por O(1).

3.1.4 void executa_processo_simulado();

Função onde é realizada a próxima instrução do processo atual, além de incrementar o tempo do sistema em 1. Sua complexidade é dada por O(n).

3.1.5 void teste_escalonador (int tempo);

Testa se o tempo já passou de um "limite", para saber se é necessário escalonar, no caso de o tempo ter passado, é chamada a função de escalonar. Sua complexidade é dada por O(1).

3.1.6 void escalonar ();

Escalona o processo atual, no caso de terminar sua execução ou passar de seu tempo "limite" definido na função "teste_escalonador". Além de atualizar o programa atual. Sua complexidade é dada por O(n).

3.1.7 celula_pcb *busca(CPU *cpu);

Busca por um processo passado como parâmetro, no caso de sucesso, é retornado seu endereço. Sua complexidade é dada por O(n).

3.1.8 void bloqueia();

Bloqueia o processo atual e o adiciona na lista dos bloqueados. Sua complexidade é dada por O(n).

3.1.9 void encerra();

Encerra o processo atual e o escalona, para um outro processo entrar em execução. Sua complexidade é dada por O(n).

3.1.10 void reporter();

Imprime dados de todos os processos atuais na tela do usuário. Sua complexidade é dada por O(n).

4 Fila

4.1 void adiciona(Fila **fila, CPU *id);

Adiciona um novo elemento no final da fila, caso seja a primeira posição é inserido na primeira posição. Sua complexidade é dada por O(n).

4.2 Fila *remover(Fila **fila);

Remove o primeiro elemento da fila. Sua complexidade é dada por O(1).

5 Estruturas de dados e variáveis globais

5.1 Estrutura de Dados

5.1.1 Struct programa

Estrutura para guardar os comandos retirados dos arquivos de leitura.

5.1.1 Struct CPU

Estrutura onde está guardada todos os comandos de todos os programas, além de dados de tempo, contador de programas e o status.

5.1.1 Struct celula_pcb

Estrutura onde guarda dados dos processos.

5.1.1 Struct tabela_pcb

Estrutura onde estão guardadas todas as células Pcb, assim formando a tabela.

5.1.1 Struct Fila

Estrutura que representa um fila tradicional(FIFO), primeiro a entrar será o primeiro a sair.

5.2 Variáveis globais

5.2.1 int t = 0;

Variável que contém o tempo de execução do sistema.

5.2.2 CPU *atual;

Estrutura onde guarda o processo que está sendo executado.

5.2.3 Fila *e_pronto = NULL;

Fila para guardar os processos prontos.

5.2.4 Fila *e_bloqueado = NULL;

Fila para guardar os processos prontos.

5.2.4 tabela_pcb *t_pcb = NULL;

Tabela PCB de todos os programas.

6 Conclusão

Neste trabalho tivemos como objetivo a implementação de um algoritmo de gerenciamento de recursos onde os desafios eram inúmeros, desde a abstração da ideia inicial, para inicializar a implementação do algoritmo em si, até às trocas de contexto e bloqueio e desbloqueio de processos, por exemplo. Após iniciarmos a implementação pudemos ir aprendendo o funcionamento desse gerenciador de recursos de acordo com o quanto avançávamos em seu desenvolvimento, de forma que, auxiliasse em nossa evolução. A linguagem C por si só já é um grande desafio para implementação de sistemas um pouco mais complexos, isso aliado ao desafio proposto nos trouxe uma visão diferente sobre as chamadas de sistema e suas particularidades.

Durante o desenvolvimento, através de testes de entradas e observações do funcionamento do algoritmo, foi possível atingir o objetivo inicial e finalizar a implementação do algoritmo de gerenciamento de recursos. Portanto, após finalizado, foi possível observar que esse trabalho é de grande aprendizado e crucial para o entendimento dessa parte da matéria.

7 Bibliografia

- Documentação do Linux;
- Documentação do C;
- Materiais da disciplina (videoaulas, slides, encontros síncronos).