Universidade federal São João Del Rei
Campus: Tancredo de Almeida Neves
Departamento de Ciência da Computação
Sistema Operacional

Trabalho Sistemas Operacionais: GERENCIADOR DE PROCESSOS

Alunos:

Alex de Andrade Soares, 182050080
Lucas Rômulo de Souza Resende, 172050036
Sidney de O. Felipe Júnior,7 172050095

Professor: Rafael Sachetto

Fevereiro
2021

z

Conteúdo

1 Introdução

Um dos conceitos fundamentais do sistema operacional moderno e o *gerenciamento de processos*, nele as prioridades de processos em execução são definidas.

Levando em consideração a quantidade de processo e as prioridades deles, a função principal do gerenciador de processos é a execução daquele processo no processador(CPU ou GPU) de forma mais adequada aquela situação atual.

Esse trabalho foi proposto para servir de similador de um gerenciador de processos, com o objetivo de implementar as seguintes objetivo:

- 3 tipos de processos : Commander, process manager e report
- 5 funções Principais do Gerenciamento:
  - Criação de Processo
  - Substituição do processo atual por outro processo
  - Transição de estados de um processo,
  - escalonamento
  - troca de contexto.
- 4 tipo de chamada de sistema do Linux:
  - fork() , no qual cria uma cópia do processo em execução;
  - wait() , na qual espera o processo filho terminar a execução;
  - pipe() , ele envia uma variável para o processo filho ou algum outro processo;
  - sleep(), tem o processo ficar "parado", por uma quantidade de segundos solicitado.

2 Metodologia

A simulacao do gerenciador de processos se consiste nos processos principais: *Commander, Processo Manager e o Reporter*, o inicio da simulacao e feita pelo processos Manager.
Essa simulação para ser feita foi preciso somente de um editor de texto e o sistema operacional Linux. sendo assim: Para a execuçao do simulador pelo

terminal,

Caminhe ate o diretório do trabalho

$ cd caminho_do_sistema/ProcessManager/

Para Compilar o programa

$ make

$ make install

Para Executar o simulador do gerenciador de processos

$ cd caminho_do_sistema/ProcessManager/bin

$ ./process_manager

Para Limpar os arquivos .o da pasta

$ make clean

3 Desenvolvimento

O programa consiste em dois elementos principais: o processo commander e o processo manager, a execução do simulador é iniciada pelo processo commander, e o controle dos processos simulados é feito pelo processo manager e reporter. Com isso iremos fazer uma rápida revisão das próximas funções implementadas e futuramente mostrando sua funcionalidade com um maior detalhamento

3.1 O Processo Commander

O processo commander primeiramente cria um pipe e depois um processo do tipo process manager. Ele então lé os comandos repetidamente (1 por segundo) da entrada padrão e os passa para process manager através do pipe. Existem 4 tipos de comandos:

- **Q** : Fim de uma unidade de tempo.

- **U** : Desbloqueie o primeiro processo simulado que está na fila de bloqueados.
- **P** : Imprima o estado atual do sistema.
- **T** : Imprima o tempo de retorno médio e finalize o simulador.

No Commander é usada as chamadas do sistema *Fork(),wait(), pipe(), sleep()* (1 tópico 3)

3.2 O Processo Simulado

Cada processo simulado consiste em um programa que manipula o valor de um única variável inteira. Sendo assim, o estado de um processo simulado em um instante de tempo consiste no valor da sua variável inteira e o valor de seu contador de programa. Um processo simulado é formado por uma sequência de instruções. Existem 7 tipos de instruções:

- **S** : n: Atualiza o valor da variável inteira para n.
- **A** : n: Soma n na variável inteira.
- **D** : n: Subtrai n na variável inteira.
- **B** : Bloqueia o processo simulado.
- **E** : Termina o processo simulado.
- **F** : n: Cria um novo processo simulado. O novo processo é uma cópia exata do pai. O novo processo executada instrução imediatamente após a instrução **F**, enquanto o pai continua n instrução após **F**.
- **R** : nome do arquivo: Substitui o programa do processo simulado com o programa no arquivo nome do arquivo, e atualiza o valor do contador de programa para a primeira instrução do novo programa.

3.3 O Process Manager *(PM)*

O process manager simula 5 gerenciamento de processo, o *PM* cria o primeiro processo simulado em *id = 0*. O programa para esse processo é lido de um arquivo com o nome *init*. Esse é o único processo criado por conta própria pelo **PM**. Todos os outros processos são criados em resposta a execução da instrução **F**. também temos os seguintes processos:

3.3.1 A Criação de um processo

Na criação de processo le-se um arquivo com as instruções, tamanho do arquivo e nome do arquivo na struct CPU e insere-a na tabela PCB.

3.3.2 Troca de Imagem

A substituição do processo atual por outro processo, também chamado de troca de imagem, nele substituímos o código do processo atual por um outro programa solicitado.

3.3.3 Troca de Contexto

A troca de contexto ocorre quando o processo simulado usa uma instrução **F** (3.2 tópico 6), nela cria-se uma cópia do processo no qual esta em execução *pai* e insere na tabela PCB, no qual começa a executar o processo filho, na qual e a copia do seu pai.

3.3.4 Transição de Estado

Todo processo tem 3 estados que ele transita entre eles ate ser concluído, esses estados são conhecidos como:

- **Bloqueado** : Nesta Estado, sabemos que o processo que estava em execução e nele o processo executou uma instrução B, fazendo com que sua execução seja interrompida e seu estado seja salvo na memoria e este processo seja enviado para a fila de bloqueados.
- **Em Execução** : Quando o processo chega neste estado falamos que o programa esta em execução atualmente na CPU.
- **Pronto** : Quando o processo esta pronto, ele esta em um processo no qual não esta em execução. mas pode ser escalonado a qualquer momento.

3.3.5 Escalonamento

O escalonamento ocorre quando se tem uma escolha de qual processo deve ser executado a partir de uma lista de processo prontos, sendo esse processo executados com uma diferença de 2 segundo entre eles.

3.4 O Reporter

O reporter tem como Função imprimir na tela o estado dos processos atual na simulação

```
*****************************************************************
Estado do sistema:
*****************************************************************\\

TEMPO ATUAL: tempo
PROCESSO EXECUTANDO:
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
BLOQUEADO:
Fila processos bloqueados:
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
...
pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
PROCESSOS PRONTOS:

pid, ppid, prioridade, valor, tempo inicio, CPU usada ate agora
*****************************************************************
```

3.5 Estruturas

3.5.1 Manager

Tipo abstrato que armazena o estado atual do sistema

```
typedef struct process_manager_t{
    int timer;
    int next_id, next_pcb;          // pcb_table
    int next_ready, next_blocked;    // queues
    process_t* cpu;
    process_t** pcb_table;
    int* ready;
    int* blocked;
} process_manager_t;
```

- Timer: tipo inteiro que representa o tempo atual em que a CPU se encontra. Next_id: tipo inteiro que armazena o id que o próximo processo criado receberá.
- Next_pcb: tipo inteiro que armazena a quantidade de processos existentes na tabela de processos

- Next_ready: tipo inteiro que armazena a quantidade de processos existente na fila de processos prontos para serem executados.
- Next_blocked: tipo inteiro que armazena a quantidade de processos existentes na fila de processos bloqueados.
- Cpu: ponteiro do tipo process_t que armazena o endereço do processo que está ocupando a CPU no momento.
- Pcb_table: matriz de vetores do tipo process_t contendo todos os processos criados durante a execução do programa.
- Ready: ponteiro do tipo inteiro que armazena o endereço que contém o primeiro ID de processo da fila de processos prontos para serem executados.
- Blocked: ponteiro do tipo inteiro que armazena o endereço que contém o primeiro ID de processo da fila de processo bloqueados.

  OBS: pcb_table, ready e blocked têm seus tamanhos limitados a quantidade máxima de processos definida pela constante PROCESS_N.

3.5.2 Process

Tipo abstrato que armazena todas as instruções de um programa.

```
typedef struct instruction_t{
   char type;
   char* value;
} instruction_t;
```

- Type: char que armazena uma instrução do programa sendo executado pelo processo.
- Value: char que armazena o valor passado para a execução da instrução, caso ele exija. OBS: value tem seu tamanho limitado pela constante INSTRUCTION_SIZE.

Tipo abstrato de dados que armazena todas as informações sobre um determinado processo.

```
typedef struct process_t{
    int id, pid, start;         // from PM
    int pc, var, cpu_usage;     // initialized 0
    int priority, priority_i;  // priority, counter to schedule
    instruction_t** instruction;
} process_t;
```

- Id: tipo inteiro que armazena o ID do processo.
- Pid: tipo inteiro que armazena o ID do processo pai
- Start: tipo inteiro que armazena o tempo de execução da CPU em que o processo foi iniciado.
- Pc: tipo inteiro que armazena o contador de programa.
- Var: Tipo inteiro que armazena o valor da variável manipulada pelo processo.
- Cpu_usage: tipo inteiro que armazena o tempo que o processo permanece na CPU.
- Priority: tipo inteiro que armazena a prioridade que o processo tem durante sua execução.
- Priority_i: tipo inteiro usado como contador para saber se o processo ainda possui priotidade para ser executado.
- Instruction: vetor de ponteiro do tipo instuction_t que armazena as instruções do programa sendo executado pelo processo.

  OBS: instruction tem seu tamanho limitado pela quantidade máxima de instruções de um programa, definida pela constante INSTRUCTION_N

3.6 Funções

Para o funcionamento de todo o simulador de gerenciador de processos foram precisos desenvolver as funções citadas abaixo:

3.6.1 Commander

- Main:

  Declara as variáveis utilizadas pelo processo principal, realizando em seguida a chamada do pipe para possibilitar que o processo principal compartilhe informações com o processo filho e do fork para a criação do processo filho, armazenando seu id em uma variável.

  Ambas as chamadas retornam uma mensagem de erro caso não seja possível executá-las.

  No processo filho, a ponta emissora do pipe é fechada e sua entrada padrão é alterada para a ponta receptora do pipe e execv é chamada, mudando a imagem do processo para o programa "manager".

  No processo principal a ponta receptora do pipe é fechada iniciando-se então a leitura dos comandos retornados pela função read_command e enviando-as para o processo filho pela ponta emissora do pipe até que seja retornado "T", que imprime o estado atual do sistema e finaliza a execução do programa.

- Read_Command:

  Inicializa com o valor '' uma variável do tipo char para armazenamento do comando inserido pelo usuário. Enquanto o retorno da função verify_input que verifica se o char é um comando aceito pelo programa for FALSE, é solicitado que seja inserido um novo comando.

  A função retorna um char com um comando lido do terminal.

- Verify_input:

  Recebe como argumento uma variável do tipo char e retorna TRUE caso seja um comando aceito pelo programa, se não, retorna FALSE.

3.6.2 Manager

- Main:

  O programa declara uma variável do tipo abstrato process_manager_t para armazenar o estado do sistema e inicializa o Process Manager chamando a função ininitialize_process_manager. Em seguida, lê os comandos inseridos, executa suas respectivas funções e incrementa o contador de tempo, até que seja recebido o comando "T" que imprime o estado atual do sistema e finaliza o programa.

3.6.3 Process

- new_process:

  Aloca espaço na memória para um novo processo com seus valores iniciados com zero e suas instruções com as instruções retornadas pela função new_instruction.

  A função retorna o processo.

- set_priority:

  Recebe as instruções como argumento e retorna como prioridade a quantidade de instruções "B" o processo executa.

- new_instrution:

  Armazena em uma variável o caminho do arquivo com as instruções do processo, aloca uma variável de tipo abstrato instructions_t com tamanho suficiente para, no máximo, 100 instruções.

  Em seguida, a função realiza a leitura do arquivo com as instruções no caminho armazenado anteriormente para então salva-las na variável do tipo instructions_t e retorna-la.

- set_var:

  Recebe como argumento um inteiro e um processo e atualiza o número armazenado no processo para o inteiro recebido.

- add_var:

  Recebe como argumento um inteiro e um processo e soma o número armazenado no processo com o inteiro recebido.

- dec_var:

  Recebe como argumento um inteiro e um processo e subtrai do número armazenado no processo o inteiro recebido.

3.6.4 Manager-TAD

- reporter:

  Recebe o process manager como argumento e imprime o estado atual do sistema e cria um pipe para permitir a comunicação com o processo filho e um fork para iniciar outro processo.

  Ambas as chamadas retornam uma mensagem de erro caso não seja possível executá-las.

No processo filho, a ponta emissora do pipe é fechada e sua entrada padrão é alterada para a ponta receptora do pipe e execv é chamada, mudando a imagem do processo para o programa "reporter".

No processo pai, a ponta receptora do pipe é fechada e os valores do process manager, um sinal se a CPU está vazia ou se há algum processo sendo executado, o processo sendo executado (caso tenha), os demais processos presentes na tabela de processos e as filas de processos prontos e processos bloqueados são enviados através do pipe.

- initialize_process_manager:

  Declara uma variável do tipo abstrato process_manager_t alocada na memória e inicializa a tabela de processos, a fila de processos prontos, bloqueados e os demais valores em zero.

  A primeira posição da tabela de processos é preenchida com o primeiro processo, que é retornado pela função first_process e valor next_pcb que representa a quantidade de processo existentes na tabela, tem seu valor incrementado.

  O primeiro processo é adicionado ao valor cpu que armazena o processo que está em execução no momento e o process manager é retornado.

- first_process:

  Recebe como argumento um ID de processo, ID do pai do processo, tempo atual e o nome de um arquivo contendo instruções a serem executadas pelo processo.

  Uma variável do tipo abstrato process_t recebe um processo retornado pela função new_process, atualiza os valores para seus respectivos valores recebidos como argumento e retorna o processo já atualizado.

- new_pcb_table:

  Aloca um vetor de processos na memória e o retorna.

- Execulte:

  Recebe o process manager como argumento e imprime uma mensagem caso não haja processo sendo executado na cpu, se não, imprime o ID do processo sendo executado na CPU, o a instrução sendo executada e caso necessário, o valor da instrução.

  O tempo de uso da CPU é incrementado, é chamada a função que corresponde a instrução recebida e o processo é escalonado com a chamada da função scheduler.

- scheduler:

  Recebe um processo como argumento. Caso a CPU esteja ocupada é verificado se o processo tem a prioridade para ser executado.

  Se não houver mais prioridade para executar instruções, o contador de prioridade é resetado e o processo é inserido ao fim da fila de processos prontos para serem executado. Em seguida a CPU é liberada e é chamada a função ready_to_cpu que move o próximo processo pronto para ser executado para a CPU.

- ready_to_cpu:

  Recebe o process manager como argumento.

  Move o próximo processo da fila de processos prontos para a CPU ou altera a cpu para NULL caso não tenha processos prontos para serem executados.

- unblock_process:

  Recebe o process manager como argumento.

  Move o primeiro processo da fila de processos bloqueados para a fila de processos prontos para serem executados pela CPU.

  Se o primeiro ID da fila de processos bloqueados for diferente de -1 (indicando que a fila está vazia), é verificado se a CPU está vazia, se sim, o primeiro processo da fila de bloqueados é movido para a CPU, se não, o processo é inserido na fila de processos prontos pra serem executados

- exit_process:

  Recebe o process manager como argumento.

  Busca na tabela de processos o index do processo em execução na CPU, remove o processo da memória e move os processos seguintes para preencher o espaço deixado.

  Decrementa todos os índices armazenados nas filas de processo bloqueados e prontos, a partir do index que foi removido, decrementa o tamanho da tabela de processos e libera a CPU.

- fork_process:

  Recebe um inteiro e o process manager como argumento.

  Cria uma cópia do processo atual, define seus valores, incrementa no contador de programa o inteiro recebido como argumento e adiciona

o processo criado na fila de processo prontos para serem executados e na tabela de processos.

- run_image:

  Recebe o nome de um programa e um processo como argumento.

  Reseta o contador de programa e o valor da variável armazenada no processo e atualiza as instruções do processo para as instruções do programa recebido e a prioridade de execução do processo.

- new_queue:

  Cria uma fila vazia onde as posições vazias são representadas pelo inteiro -1 e retorna um ponteiro para a fila criada

- add_queue:

  Recebe um inteiro qualquer, um ponteiro para uma fila e o tamanho da fila como argumentos.

  Adiciona o inteiro no fim da fila e atualiza o tamanho da fila.

- move_queue:

  Recebe um ponteiro para uma fila e o seu tamanho como argumentos.

  Percorre a fila, movendo os elementos uma posição à frente.

3.6.5 Reporter

- main:

  Aloca uma variável do tipo abstrato process_manager_t e a inicializa chamando a função get_process_manager. Em seguida, a função imprime o estado atual do sistema e a tabela de processos.

- get_process_manager:

  Recebe o um ponteiro para o process manager como argumento.

  A função recebe pela ponta receptora do pipe, todas as informações que são passadas em reporter() no arquivo manager_TAD.c

3.7 Analise de Dados

Foram escritos cinco programas que estao locazado na pasta *$ cd caminhopasta/ProcessManager/programs* sendo eles:

3.8 programa 0

```
S 50
A 32
D 32
B
D 50
F 1
R program2.txt
B
A 1
E
```

3.9 programa 1

```
A 52
B
B
A 65
D 98
S 12
B
S 65
D 88
F 1
R program3.txt
B
E
```

3.10 programa 2

```
S 565
B
A 78
A 98
```

```
F 1
R program4.txt
B
D 54
D 78
A 2
B
S 3
E
```

3.11 programa 3

```
A 98
F 1
R program4.txt
B
A 55
F 1
R program4.txt
D 22
S 58
A 1
B
D 96
E
```

3.12 programa 4

```
D 23
B
A 58
D 65
S 12
B
D 65
A 28
S 99
D 21
B
D 53
A 2
```

E

Dados os programas acima fizemos alguns testes no simulador, lembrando que cada interrupção tem o tempo de 1s para o próximo passo, temos a seguir os dados obtidos na execução do simulador:

```
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   0
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |2      |0      |0      |0
|BLOCKED PROCESSES:
|READY PROCESSES:
q
_/> pid 0: S 1000
q
_/> pid 0: A 19
q
_/> pid 0: A 20
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   4
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |2      |1039   |0      |3
|BLOCKED PROCESSES:
|READY PROCESSES:
q
_/> pid 0: D 23
q
_/> pid 0: A 55
p
*****************************************************
                        Sys status:
*****************************************************
```

```
TIME RUNNING:   7
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |2      |1071   |0      |5
|BLOCKED PROCESSES:
|READY PROCESSES:
q
_/> pid 0: F 1 --> HOUVE UMA TROCA DE CONTEXTO
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   9
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|1      |0      |2      |1071   |8      |0
|BLOCKED PROCESSES:
|READY PROCESSES:
|0      |0      |2      |1071   |0      |6
q
_/> pid 1: R program0.txt
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   11
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|1      |0      |2      |0      |8      |1
|BLOCKED PROCESSES:
|READY PROCESSES:
|0      |0      |2      |1071   |0      |6
q
_/> pid 1: S 50
q
_/> pid 1: A 32
q
_/> pid 0: A 343
q
_/> pid 0: B --> HOUVE UM BLOQUEIO DO PROCESSO
p
```

```
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   16
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|1      |0      |2      |82     |8      |3
|BLOCKED PROCESSES:
|0      |0      |1      |1414   |0      |8
|READY PROCESSES:
q
_/> pid 1: D 32
q
_/> pid 1: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   20
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|0      |0      |1      |1414   |0      |8
|1      |0      |1      |50     |8      |5
|READY PROCESSES:
u
_/> pid 0 unblocked.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   22
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |1      |1414   |0      |8
|BLOCKED PROCESSES:
|1      |0      |1      |50     |8      |5
|READY PROCESSES:
u
```

```
_/> pid 1 unblocked.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   24
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |1      |1414   |0      |8
|BLOCKED PROCESSES:
|READY PROCESSES:
|1      |0      |1      |50     |8      |5
q
_/> pid 0: S 23
q
_/> pid 0: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> pid 1: D 50
q
_/> pid 1: F 1 --> HOUVE UMA TROCA DE CONTEXTO
q
_/> pid 2: R program2.txt
q
_/> pid 2: S 565
q
_/> pid 2: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> pid 1: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   34
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|0      |0      |0      |23     |0      |10
|2      |1      |2      |565    |28     |3
|1      |0      |0      |0      |8      |8
```

```
|READY PROCESSES:
u
_/> pid 0 unblocked.
u
_/> pid 2 unblocked.
u
_/> pid 1 unblocked.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   38
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|0      |0      |0      |23     |0      |10
|BLOCKED PROCESSES:
|READY PROCESSES:
|2      |1      |2      |565    |28     |3
|1      |0      |0      |0      |8      |8
q
_/> pid 0: E
q
_/> pid 2: A 78
q
_/> pid 2: A 98
q
_/> pid 2: F 1 --> HOUVE UMA TROCA DE CONTEXTO
q
_/> pid 1: A 1
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   44
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|3      |2      |2      |741    |42     |0
|BLOCKED PROCESSES:
|READY PROCESSES:
|2      |1      |2      |741    |28     |6
|1      |0      |0      |1      |8      |9
```

```
q
_/> pid 3: R program4.txt
q
_/> pid 3: D 23
q
_/> pid 3: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> pid 2: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> pid 1: E
q
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   51
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|3      |2      |2      |-23    |42     |3
|2      |1      |1      |741    |28     |7
|READY PROCESSES:
q
_/> Theres no process running now.
u
_/> pid 3 unblocked.
u
_/> pid 2 unblocked.
u
_/> No process to unblock.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   56
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|3      |2      |2      |-23    |42     |3
|BLOCKED PROCESSES:
|READY PROCESSES:
```

```
|2      |1      |1      |741    |28     |7
q
_/> pid 3: A 58
q
_/> pid 3: D 65
q
_/> pid 3: S 12
q
_/> pid 2: D 54
q
_/> pid 2: D 78
q
_/> pid 3: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> pid 2: A 2
q
_/> pid 2: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   66
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|3      |2      |1      |12     |42     |7
|2      |1      |0      |611    |28     |11
|READY PROCESSES:
u
_/> pid 3 unblocked.
u
_/> pid 2 unblocked.
u
_/> No process to unblock.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   70
```

```
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|3      |2      |1      |12     |42     |7
|BLOCKED PROCESSES:
|READY PROCESSES:
|2      |1      |0      |611    |28     |11
q
_/> pid 3: D 65
q
_/> pid 3: A 28
q
_/> pid 2: S 3
q
_/> pid 3: S 99
q
_/> pid 3: D 21
q
_/> pid 2: E
q
_/> pid 3: B --> HOUVE UM BLOQUEIO DO PROCESSO
q
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   79
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|3      |2      |0      |78     |42     |12
|READY PROCESSES:
u
_/> pid 3 unblocked.
q
_/> pid 3: D 53
q
_/> pid 3: A 2
q
_/> pid 3: E
q
```

```
_/> Theres no process running now.
p
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   85
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|READY PROCESSES:
t
*****************************************************
                        Sys status:
*****************************************************
TIME RUNNING:   86
|pid    |ppid   |prior. |var    |start  |cpu_usage
|RUNNING PROCESS:
|BLOCKED PROCESSES:
|READY PROCESSES:
```

4 Conclusão

O objetivo deste trabalho foi construir um algoritmo de gerenciamento de processos, porém houve etapas que a implementação na linguagem C, ficou confusa e um pouco complexa de ser resolvido mas no decorrer do desenvolvimento conseguimos atingir o objetivo e fazer um simulador de gerenciamento de processo, com isso nos possibilitando fazer uma analise critica de todo comportamento do simulador. Portanto, este trabalho abre uma uma oportunidade de simular e compreender os processos executados no Linux

5 Bibliografia

@URL readv(2) — Linux manual page, author = Michael Kerrisk, year = 2020, month = dec, crossref = https://man7.org/linux/man-pages/man2/readv.2.html

@URL C library function - atoi(), author = Tutorialspoint, crossref = https://www.tutorialspoint.com/c_standard_library/c_function_atoi.htm

@BOOK Sistemas Operacionais – Projeto e Implementação, author = Andrew S. Tanenbaum e Albert S. Woodhull, title = Sistemas Operacionais – Projeto e Implementação, publisher = Editora Bookman, year = 2006, edition = third