UFRJ
UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

Trabalho 1 - Relatório Técnico

Desenvolvimento de um Escalonador de Processos utilizando o Algoritmo Round-Robin com Feedback

Autores
Pedro Saito: 122149392
Halison Souza:
Allan Kildare

Professor Orientador
Valeria Menezes Bastos

Arquitetura de Computadores e Sistemas Operacionais
Universidade Federal do Rio de Janeiro
2024

Sumário

1 Introdução 3
1.1 Objetivos 3
1.2 Premissas 4

2 Algoritmo 6
2.1 Funcionamento 6

3 Código 8
3.1 Organização do Projeto 8
3.2 Estruturas e Enumerações 9
3.2.1 Lista de Processos 9
3.2.2 Processo 9
3.2.3 Operações de Entrada e Saída 10
3.2.4 Tipo de Operação de Entrada e Saída 10
3.2.5 Status do Processo 10
3.2.6 Lista de Filas 11
3.2.7 Fila 11
3.2.8 Tipo da Fila 11
3.2.9 Nó 11
3.3 Funções 12
3.3.1 Processos 12
3.3.2 Interface 13
3.3.3 Escalonador 14
3.3.4 Utilitários 15
3.3.5 Macros e Constantes 16

4 Instruções 18

5 Saída 19

Bibliografia 28

1. Introdução

Nos sistemas operacionais modernos, o escalonamento de processos é responsável por determinar qual dos **processos**, ou programas em execução, receberá a próxima fatia de tempo da UCP (Unidade Central de Processamento). Esse mecanismo é primordial para a distribuição equitativa dos recursos do processador, evitando que processos monopolizem a UCP e, consequentemente, prevenindo a *starvation* — situação em que um processo é continuamente negado os recursos necessários para sua execução, frequentemente devido a concorrência com processos mais prioritários. [1]

Para mitigar esse problema, utilizam-se algoritmos que determinam a alocação de curto prazo de todos os processos prontos para a execução. Tais abordagens visam maximizar a utilização da UCP, sendo fundamentais no contexto de multiprogramação. Nesse sentido, é necessário compreender as particularidades das diferentes metodologias de escalonamento e suas características. [2]

1.1 Objetivos

O presente estudo tem como objetivo simular o algoritmo de escalonamento Round-Robin com feedback, também conhecido como escalonamento circular, no contexto de realização de operações de entrada e saída em ambientes de tempo compartilhado.

Especificamente, estamos interessados em:

- Implementar um simulador do escalonador Round-Robin com feedback, incorporando filas de diferentes prioridades e operações de entrada e saída.
- Detalhar cada componente de código desenvolvido, explicando as estruturas e funções utilizadas na implementação.
- Promover fins didáticos, facilitando a compreensão do algoritmo e demonstrando sua aplicação na prática no gerenciamento de processos.

Ao final, espera-se demonstrar a eficácia e funcionamento desse algoritmo de escalonamento, demonstrando sua relevância na gestão de processos e otimização dos recursos do sistema.

1.2 Premissas

Para o desenvolvimento do simulador, foram estabelecidas algumas premissas, listadas a seguir:

- **Limite Máximo de Processos Criados** (`MAX_PROCESSOS`): Define-se um limite máximo de 5 processos que podem ser criados, independentemente de seu estado. Cada processo recebe um identificador único (`PID`), variando entre 0 e `MAX_PROCESSOS`−1.
- **Fatia de Tempo** (`QUANTUM`): Define-se uma fatia de tempo de 3 unidades de tempo para cada processo em execução.
- **Tempos de Serviço Máximo e Mínimo** (`TEMPO_CPU_MAX`, `TEMPO_CPU_MIN`): Estabelece o intervalo de duração para a execução de cada processo, com valores máximo e mínimo de 16 e 3 unidades de tempo, respectivamente.
- **Tempo de Chegada Máximo** (`TEMPO_CHEGADA_MAX`): Define o limite superior de 10 unidades de tempo para a chegada dos processos no sistema.

Especificação das Operações de Entrada e Saída: Implementamos dois tipos de dispositivos de I/O, especificados abaixo:

- **Disco** (`TEMPO_DISCO`): Define-se o tempo máximo para a operação de disco na variável `TEMPO_DISCO`, que corresponde a 2 unidades de tempo. Após essa operação, o processo retorna para a fila de baixa prioridade.
- **Fita magnética**: Define-se o tempo máximo para a operação de fita magnética na variável `TEMPO_FITA`, que corresponde a 3 unidades de tempo. Após essa operação, o processo retorna para a fila de alta prioridade.
- **Tempo de entrada e saída padrão** (`TEMPO_IO_PADRAO`): Inicializado como −1 para processos sem operações de entrada e saída.

Para assegurar a eficiência e o controle dos processos, adotamos as seguintes medidas de escalonamento:

- **Definição do PID** (`processo.pid`): O identificador único do processo, também conhecido como `PID`, pode ser atribuído de duas formas dependendo da escolha do usuário no menu:
  1. **Importação Manual de Dados**: Ao importar dados de um CSV, o usuário deve fornecer o `PID` manualmente, com validação para evitar duplicidade ou inconsistência.

2. **Geração Automática de Dados**: Na geração aleatória, o `PID` é atribuído de forma incremental à medida que cada processo é criado.

- **Filas do Escalonador**: O escalonador possui quatro filas, divididas em duas de prioridades — alta e baixa — e duas filas individuais para as operações de entrada e saída de disco e fita.

- **Ordem de Entrada na Fila de Prontos**: A ordem prevista para a entrada de processos está descrita abaixo:

  1. Processos recém-chegados entram na fila de alta prioridade.
  2. Processos que retornaram de operações de entrada/saída.
  3. Processos que sofreram preempção retornam para a fila de baixa prioridade.

Observação: Se um processo concluir uma operação de disco e retornar no mesmo instante em que outro processo sofreu preempção, excepcionalmente, o processo preemptado virá à frente na fila de baixa prioridade em relação ao processo que retornou do I/O.

Por fim, implementamos uma série de premissas adicionais que, embora não tenham sido especificadas na descrição original do trabalho, são importantes para o funcionamento do escalonador:

- **Requisições de Operações de Entrada e Saída**: Estabelecemos as seguintes limitações:

  - Um processo não pode realizar duas operações de entrada e saída simultaneamente.
  - Um processo não pode solicitar uma operação de entrada e saída no seu **último instante de execução**, apenas no penúltimo se for o caso.
  - Um processo não pode repetir a mesma operação de entrada e saída.

- **Leitura de Arquivo CSV**: Ao importar processos de um arquivo CSV, implementamos validações para garantir a consistência dos dados e evitar erros de execução. As verificações incluem:

  - Detecção de caracteres inválidos em campos numéricos.
  - Remoção de linhas em branco e caracteres especiais.
  - Identificação de valores fora dos limites permitidos para cada campo (por exemplo, instante de chegada negativo).
  - Número máximo de caracteres por linha (`MAX_LINHA`): Define-se o número máximo permitido de caracteres por linha do CSV como 100.

2. Algoritmo

O algoritmo de escalonamento Round-Robin é um dos métodos preemptivos[1] mais amplamente empregados no escalonamento de processos e redes na computação. Essa abordagem concede a cada processo uma fatia de tempo estática de forma cíclica, sendo considerado *starvation free*, impedindo que qualquer processo seja permanentemente preterido em favor de outros. É popular em sistemas de tempo compartilhado e multi-tarefa, onde vários processos precisam ser executados simultaneamente. [3]

2.1 Funcionamento

Antes de descrever o processo de escalonamento, precisamos realizar uma série de hipóteses, listadas a seguir:

1. O conjunto de tarefas consistem em processos executáveis aguardando a UCP.
2. Todos os processos são independentes e competem por recursos.
3. A função do escalonador é alocar de forma justa os recursos limitados da UCP aos diferentes processos de modo a otimizar algum critério de desempenho. [4]

Com base nisso, passamos a detalhar os fundamentos do algoritmo de escalonamento Round-Robin:

- Os processos são organizados em uma fila circular seguindo a ordem de chegada.
- Cada processos recebe uma fatia de tempo fixa (*Time Slice*) para execução.
- Ao término da fatia de tempo atribuída, o processo é interrompido, retornando para o fim da fila, permitindo que os demais processos sejam executados.

No entanto, o objetivo desse projeto é simular o Round-Robin com *feedback*, isto é, uma variação do algoritmo Round-Robin que ajusta dinamicamente os processos segundo o tempo solicitado de UCP. Nesse sentido, a fila de *feedback* multinível estende o escalonamento padrão com os seguintes requisitos:

- Classificar os processos em múltiplas filas de prontos conforme a demanda da UCP.
- Priorizar processos de curta duração (*CPU-Bound*).
- Priorizar processos com elevados períodos de E/S (*IO-Bound*) .

Na prática, isso implica que, ao esgotar sua fatia de tempo, um processo é removido fila atual e transferido para uma fila de prioridade mais baixa. Em nossa implementação, utilizamos apenas duas filas: uma de alta prioridade, onde os processos novos são

[1] O sistema operacional pode interromper um processo em execução para alocar a UCP a outro.

inseridos, e outra de baixa prioridade, na qual os processos aguardam a conclusão dos anteriores para sua execução.

Um processo pode realizar operações de entrada e saída, interrompendo-se até sua conclusão. Nesse período, ele é movido para a fila de espera, liberando a UCP para outros processos. Após a operação, o processo retorna à fila de alta ou baixa prioridade. Foram implementadas duas filas de entrada e saída: Fita magnética e Disco.

Aqui está um exemplo prático de funcionamento do algoritmo, que veremos posteriormente na impressão do escalonamento. Considere a seguinte tabela de processos:

PID	Tempo de Início	Tempo de Serviço	E/S (Tempo de Início)
P0	3	3	Vazio
P1	1	5	Fita (1)
P2	2	3	Disco (1)
P3	0	5	Fita (3)

Aqui está o gráfico correspondente, criado com base no nosso escalonador:

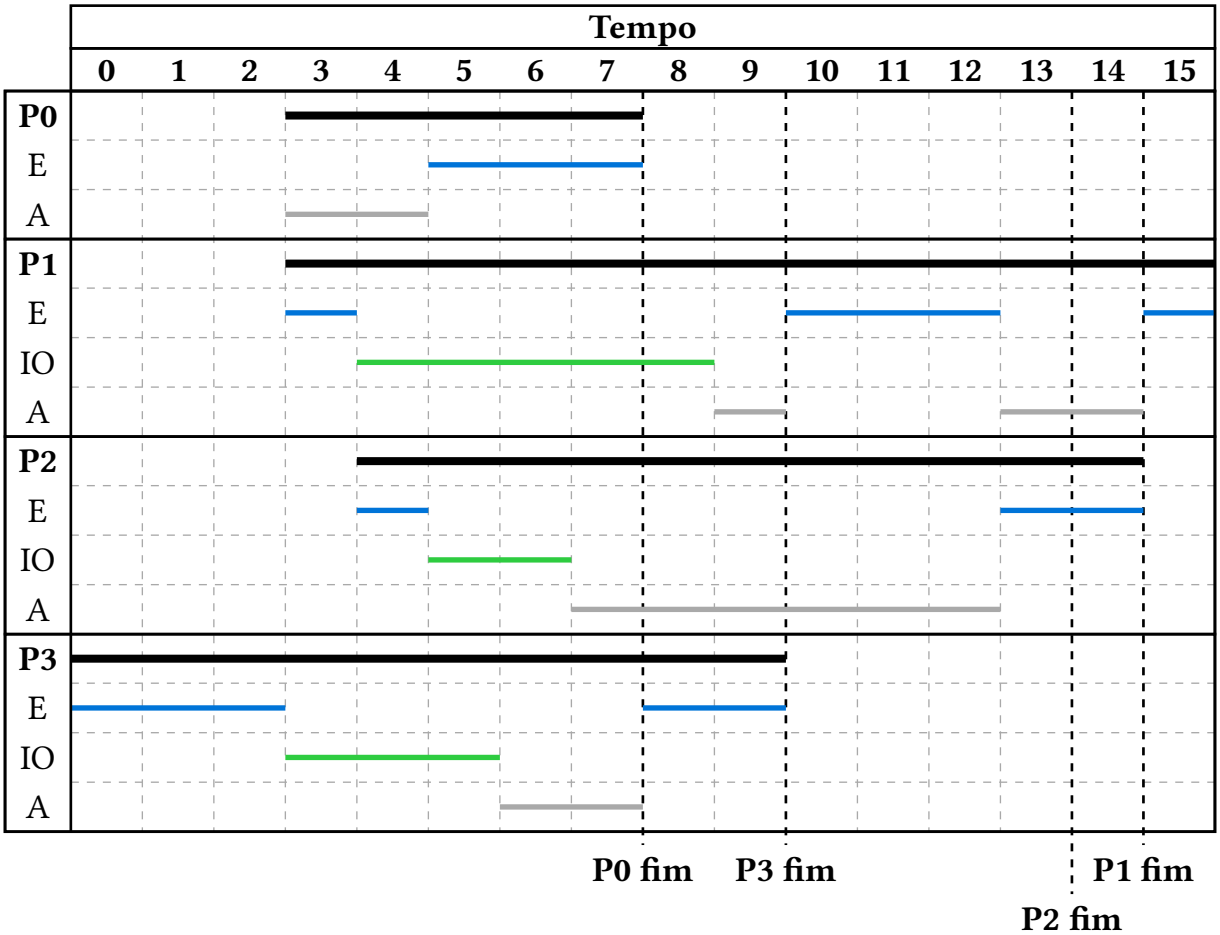

Figura 1: "Linha azul: Execução (E); linha verde: E/S (IO); Linha cinza: Aguardando."

3. Código

Nesta seção, descrevemos a implementação do simulador do escalonador Round--Robin com feedback, incluindo estruturas de dados, arquivos de cabeçalho e funções principais. O código em C segue práticas de modularidade e simplicidade.

3.1 Organização do Projeto

O projeto tem quatro arquivos de cabeçalho, cada um com um arquivo `.c` correspondente, exceto o `main.c`. As funções estão declaradas nos arquivos de cabeçalho. Abaixo está a estrutura do projeto:

```
├── LICENSE
├── Makefile
├── README.md
├── include
│   ├── escalonador.h
│   ├── fila.h
│   ├── interface.h
│   ├── processo.h
│   └── utilitarios.h
├── main
├── processos.csv
└── src
    ├── escalonador.c
    ├── fila.c
    ├── interface-pretty.c
    ├── interface.c
    ├── main.c
    ├── processo.c
    └── utilitarios.c
```

Os diretórios e arquivos do projeto estão organizados da seguinte forma:

- **`include/`**: Arquivos de cabeçalho (`.h`).
  - `utilitarios.h`: Funções auxiliares.
  - `escalonador.h`: Funções do escalonador.
  - `fila.h`: Funções para filas.
  - `interface.h`: Interface com o usuário.
  - `processo.h`: Funções dos processos.
- **`src/`**: Arquivos de implementação (`.c`).

- `utilitarios.c`: Implementação das funções auxiliares.
- `escalonador.c`: Implementação do escalonador.
- `fila.c`: Implementação das filas.
- `interface.c`: Implementação da interface.
- `interface-pretty.c`: Implementação da interface com caracteres Unicode.
- `processo.c`: Implementação dos processos.
- `main.c`: Função principal.

3.2 Estruturas e Enumerações

Durante a implementação do escalonador, foram definidas estruturas de dados que representam os processos, filas e operações. Além disso, enumerações foram aplicadas para padronizar estados e tipos, facilitando a manipulação e leitura de código.

Abaixo, entro em detalhes sobre cada uma das estruturas e enumerações utilizadas.

3.2.1 Lista de Processos

A estrutura `ListaProcessos` engloba as estruturas de processos e aloca uma quantidade fixa de processos, isto é, `MAXIMO_PROCESSOS`. Contém as seguintes variáveis:

Variável	Descrição
`processos`	Vetor de `Processos` com até `MAXIMO_PROCESSOS`.
`quantidade`	Nº de processos alocados.

3.2.2 Processo

A estrutura `Processo` representa um processo no escalonamento, contendo as seguintes variáveis:

Variável	Descrição
`pid`	ID do processo.
`instante_chegada`	Instante de criação.
`tempo_cpu`	Duração requisitada.
`tempo_turnaround`	Tempo total no sistema.
`tempo_quantum_restante`	Tempo até preempção.
`tempo_cpu_restante`	Processamento restante.
`tempo_cpu_atual`	Processamento atual.
`num_operacoes_io`	Nº de operações I/O.
`tipo_io_atual`	Tipo de I/O (0: Disco, 1: Fita).
`operacoes_io`	Vetor de I/O.

Variável	Descrição
status_processo	Status do processo.

3.2.3 Operações de Entrada e Saída

A estrutura `OperacaoIO` representa uma operação de entrada, possuindo as seguintes variáveis:

Variável	Descrição
tipo_io	Enum com tipos de I/O.
presente	Flag: 1 para presente, 0 para ausente.
tempo_inicio	Início relativo à UCP.
tempo_restante	Tempo para finalizar o I/O.

3.2.4 Tipo de Operação de Entrada e Saída

A enumeração `OperacaoIO` indica o tipo de operação de entrada, possuindo as seguintes variáveis:

Enumeração	Descrição	Valor
DISCO	Entrada/Saída em disco.	0
FITA	Entrada/Saída em fita.	1
QUANTIDADE_TIPOS_IO	Total de tipos de I/O.	2

A constante `QUANTIDADE_TIPOS_IO` é utilizada para representar o número total de tipos de I/O disponíveis, possuindo o valor 2.

3.2.5 Status do Processo

A enumeração `StatusProcesso` identifica o estado atual em que o processo se encontra.

Enumeração	Descrição	Valor
EXECUTANDO	Estado de Execução.	0
ENTRADA_SAIDA	Estado de Entrada/Saída.	1

Embora o status `PRONTO` exista, ele não foi relevante durante a implementação do escalonador.

3.2.6 Lista de Filas

A estrutura `ListaFila` engloba a estrutura de Filas, alocando um vetor fixo de tamanho `NUM_FILAS` para armazenar cada uma das filas.

Variável	Descrição
`filas`	Vetor de `Fila` com até `NUM_FILAS`.

3.2.7 Fila

A estrutura `Fila` é a implementação padrão de uma lista de encadeadas de nós, em que cada um deles contém um processo.

Variável	Descrição
`inicio`	Ponteiro para primeira estrutura `No` da fila.
`fim`	Ponteiro para última estrutura `No` da fila.

3.2.8 Tipo da Fila

A enumeração `TipoFila` indica o tipo das fila dentre as 4 filas possíveis.

Enumeração	Descrição	Valor
`FILA_ALTA_PRIORIDADE`	Fila de alta prioridade	0
`FILA_BAIXA_PRIORIDADE`	Fila de baixa prioridade.	1
`FILA_DISCO`	Fila do disco.	2
`FILA_FITA`	Fila da fita magnética.	3
`NUM_FILAS`	Nº máximo de filas.	4

3.2.9 Nó

A estrutura `No` é responsável por conter o processo atual e apontar para o próximo item da lista encadeada.

Enumeração	Descrição
`processo`	Ponteiro para o processo.
`prox`	Ponteiro para o próximo nó da fila.

3.3 Funções

Nesta seção, descrevemos detalhadamente as principais funções implementadas no simulador, responsáveis por executar as diversas operações essenciais ao funcionamento do escalonador Round-Robin com feedback. As funções estão organizadas em módulos conforme suas responsabilidades específicas, como gerenciamento de processos, interface com o usuário, operação do escalonador e utilitários auxiliares.

3.3.1 Processos

Iniciaremos a análise das funções de gerenciamento de processos, enfatizando as relacionadas à criação de processos.

1. `criar_processo_aleatorio(void)`
   - Gera um processo com variáveis aleatórias dentro os limites permitidos, utilizando a *seed* baseada no tempo atual.
   - Retorna um objeto da estrutura `Processo`.
2. `criar_lista_processo_aleatorio(void)`
   - Gera uma lista de processos por meio da função anterior. Define o `PID` de cada processo de forma incremental.
   - Retorna um objeto da estrutura `ListaProcessos`.
3. `criar_lista_processos_csv(const char *nome_arquivo)`
   - Gera uma lista de processos a partir de um arquivo CSV, tratando erros de leitura e arquivos vazios.
   - Retorna um objeto da estrutura `ListaProcessos`.
4. `trim_novalinha(char *str)`
   - Remove caracteres de **quebra de linha** e **retorno de carro**.
   - Não possui valor de retorno.
5. `executa_processo(Processo *processo)`
   - Simula a execução do processo em uma unidade de tempo.
   - Atualiza as variáveis `tempo_cpu_restante`, `tempo_cpu_atual` e `tempo_quantum_restante`.
   - Não possui valor de retorno.
6. `processo_finalizado(Processo *processo)`

- Verifica se o processo finalizou sua execução; Desenfileira o processo se for o caso.
- Não possui valor de retorno.

7. `tempo_inicio_io(Processo *processo)`

   - Verifica se o processo possui operação de entrada e saída pendente; Enfileira o processo na operação de entrada e saída correspondente.
   - Retorna 1 em caso de sucesso; 0, caso contrário.

8. `executa_io(Processo *processo)`

   - Simula e execução a operação de entrada e saída do processo.
   - Não possui valor de retorno.

9. `io_finalizada(Processo *processo)`

   - Verifica se a operação de entrada e saída foi concluída; Enfileira o processo na fila de prioridade.
   - Retorna 1 em caso de sucesso; 0, caso contrário.

10. `tempo_quantum_completo(Processo *processo)`

    - Verifica se o processo finalizou seu quantum; Enfileira o processo na fila de baixa prioridade.
    - Retorna 1 em caso de sucesso; 0, caso contrário.

Agora, vamos nos voltar para as funções cujo objetivo é disponibilizar uma interface ao usuário.

3.3.2 Interface

Vamos começar descrevendo funções destinadas à criação da interface gráfica para o usuário:

1. `imprime_menu(void)`

   - Exibe um menu contendo as opções: “Carregar dados de arquivo externo (CSV)”, “gerar dados aleatoriamente” e “Sair”.
   - Não possui valor de retorno.

2. `processa_menu(ListaProcessos *lista_processos)`

   - Convoca a função anterior e atualiza a lista processos com base na escolha do usuário (1: Extração do CSV) ou (2: Geração aleatória).

- Não possui valor de retorno.

3. `imprime_instante(int tempo_atual)`

   - Exibe um cabeçalho contendo o instante atual.
   - Não possui valor de retorno.

4. `imprime_fim_escalonador(void)`

   - Exibe um cabeçalho informando o fim do escalonamento.
   - Não possui valor de retorno.

5. `imprime_fila(const char *nome_fila, Fila *fila)`

   - Exibe o nome da fila juntamente de sua representação, começando do início até o fim.
   - Não possui valor de retorno.

6. `imprime_todas_filas(ListaFila *lista_filas)`

   - Engloba a função anterior, imprime todas as filas especificadas.
   - Não possui valor de retorno.

7. `imprime_tabela_processos(ListaProcessos *lista_processos)`

   - Exibe a tabela de processos contendo o `PID`, `instante_chegada`, `tempo_cpu` e operações de entrada e saída para cada um dos processos.
   - Não possui valor de retorno.

8. `imprime_turnaround_processos(ListaProcessos lista_processos)`

   - Imprime os tempos de turnaround e de espera para cada processo.
   - Não possui valor de retorno.

3.3.3 Escalonador

1. `processa_fila_io(Fila *fila_prioridade, ListaFila *lista_filas, ListaProcessos *lista_processos, int instante_atual, int *processos_finalizados)`

   - Executa a operação de entrada e saída atual, verifica finalização dessa operação ou do processo.
   - Não possui valor de retorno.

2. `processa_fila_prioridade(Fila *fila_io, Fila *fila_destino, const char *prioridade, int instante_atual, int *processos_finalizados)`

- Executa o processo atual, verifica finalização, operações de entrada e saída pendentes ou mudança de prioridade de filas.
- Não possui valor de retorno.

3. `escalonador(ListaFila *lista_filas, ListaProcessos *lista_processos)`

   - Função central do programa, gerencia o fluxo das filas de prioridade e de entrada e saída. Pode ser descrita nas seguintes etapas:

     1. Impressão das informações de cada processo descrito na tabela de processos.
     2. Solicitação da entrada do usuário para prosseguir a execução.
     3. Realização de um *loop* para cada instante de execução do escalonador.
     4. Verificação se novos processos chegaram.
     5. Função `processa_fila_prioridade` para `FILA_ALTA_PRIORIDADE`.
     6. Função `processa_fila_prioridade` para `FILA_BAIXA_PRIORIDADE`.
     7. Função `processa_fila_io` para `FILA_DISCO`.
     8. Função `processa_fila_io` para `FILA_FITA`.
     9. Verificação se todas as filas estão vazias com `todas_filas_vazias()`.
     10. Finalização do escalonamento se todas as filas estiverem vazias.

4. `todas_filas_vazias(ListaFila todas_filas)`

   - Verifica se todas as 4 estão vazias.
   - Retorna 1 em caso de sucesso; 0, caso contrário.

5. `verifica_novos_processos(Fila *fila, ListaProcessos *lista_processos, int instante_atual)`

   - Enfileira processos que chegaram no instante atual.
   - Não possui valor de retorno.

6. `envia_para_io(Processo *processo, ListaFila *lista_filas)`

   - Enfileira processo na fila de I/O correspondente no instante atual.
   - Não possui valor de retorno.

7. `atualiza_turnaround(Processo *processo_atual, int instante_fim)`

   - Atualiza o tempo de turnaround dos processos: Instante fim — Instante chegada.
   - Não possui valor de retorno.

3.3.4 Utilitários

1. `enviar_mensagem_erro(const char *mensagem)`

- Exibe uma mensagem de erro no `stderr` e encerra o programa.
- Não possui valor de retorno.

2. `obter_entrada_caractere(const char *mensagem, char* variavel, char min, char max)`

   - Recebe a entrada do usuário durante a escolha do menu.
   - Não possui valor de retorno.

3. `valida_entrada_char(const char *mensagem, char *variavel)`

   - Verifica se a entrada inserida é válida.
   - Não possui valor de retorno.

4. `gerar_dado_aleatorio(int min, int max)`

   - Retorna um inteiro aleatório dentro do limite especificado.

5. `seleciona_tempo_io(TipoIO tipo_io)`

   - Define a duração da operação de IO conforme o tipo de IO escolhido.
   - Retorna o valor inteiro correspondente à duração

6. `parse_linha_csv(char *linha, Processo *processo)`

   - Extrai variáveis dos processos separadas por vírgulas.
   - Retorna 1 em caso de sucesso; 0, caso contrário.

7. `seleciona_tipo_io(TipoIO tipo_io)`

   - Define a duração da operação de IO conforme o tipo de IO escolhido.
   - Retorna o valor inteiro correspondente à duração

3.3.5 Macros e Constantes

A descrição e o valor de todas as constantes e macros empregadas se encontram abaixo:

Constante	Descrição	Valor
QUANTUM	Fatia de tempo	3
MAXIMO_PROCESSOS	Máximo de processos	5
TEMPO_MIN_CPU	Serviço mínimo	3
TEMPO_MAX_CPU	Serviço máximo	16
TEMPO_MAX_CHEGADA	Chegada máxima	10
TEMPO_DISCO	I/O do disco	2

Constante	Descrição	Valor
TEMPO_FITA	I/O da fita	3
TEMPO_IO_PADRAO	Sem operação de I/O	−1
NUM_FILAS	Máximo de filas	4
MAX_LINHAS	Máximo de caracteres por linha	100

4. Instruções

Para realizar a configuração e a compilação do projeto, é necessário que o ambiente do usuário atenda a alguns pré-requisitos. Primeiramente, certifique-se de que o utilitário `make` está instalado na máquina, assim como o compilador `gcc`.

A utilização do `make` permite automatizar a compilação por meio de um arquivo Makefile, que organiza as dependências e simplifica o processo. O compilador `gcc`, por sua vez, será responsável por traduzir os arquivos de código fonte em binários executáveis.

Existem duas opções disponíveis para compilar o projeto, dependendo da interface desejada:

1. **Interface Padrão**: Utiliza caracteres `ASCII` para visualizar as filas. Para compilar, execute:

```bash
make clean
make
```

1. **Interface com Caracteres Unicode**: Utiliza caracteres Unicode para uma visualização mais sofisticada. Para compilar, execute:

```bash
make clean
make pretty
```

Observações: O comando `make clean` permite remover os binários antigos para a nova compilação. Após a compilação, o programa é executado automaticamente.

Após a execução, o usuário visualizará o menu do simulador com as seguintes opções:

1. “Carregar dados de um arquivo externo (CSV)”
2. “Gerar dados aleatoriamente”
3. “Encerrar o programa”

O usuário deve selecionar uma opção inserindo um valor numérico entre 1 e 3. Em seguida, será exibida a tabela de processos, permitindo decidir se deseja prosseguir com o escalonamento ou não. A saída será apresentada conforme o formato descrito na próxima seção.

5. Saída

```text
╔═══════════════════════════════════════════════════╗
║      SIMULADOR ROUND ROBIN COM FEEDBACK           ║
╚═══════════════════════════════════════════════════╝
│  1. Carregar dados de arquivo externo (CSV). │
│  2. Gerar dados aleatoriamente.              │
│  3. Sair.                                    │
└──────────────────────────────────────────────┘
Escolha uma opção (1 - 3): 1
═══════════════════════════ PROCESSOS ═══════════════════════════
 PID  Tempo de Início   Tempo de Serviço  E/S (Tempo de Inicio)
──────────────────────────────────────────────────────────────────
 P0         3                  3          Nenhuma operacao de E/S.
 P1         1                  5          Fita (1)
 P2         2                  3          Disco (1)
 P3         0                  5          Fita (3)
══════════════════════════════════════════════════════════════════
Aqui estão os dados dos processos, deseja prosseguir? (s/N): s
┌──────────────────────────────────────┐
│         >>> INSTANTE 00 <<<          │
└──────────────────────────────────────┘
 P3 entrou na fila de alta prioridade.
 P3 executou por 1 u.t.

 ALTA PRIORIDADE  :
   ┌──────────┐
   │   P3     │
   └──────────┘
 BAIXA PRIORIDADE : Vazia

 DISCO : Vazia

 FITA : Vazia

┌──────────────────────────────────────┐
│         >>> INSTANTE 01 <<<          │
```

```
└──────────────────────────────────────────┘
 P1 entrou na fila de alta prioridade.
 P3 executou por 1 u.t.

 ALTA PRIORIDADE  :
   ┌──────────┬──────────┐
   │   P3     │   P1     │
   └──────────┴──────────┘
 BAIXA PRIORIDADE : Vazia

 DISCO : Vazia

 FITA : Vazia

┌──────────────────────────────────────────┐
│         >>> INSTANTE 02 <<<              │
└──────────────────────────────────────────┘
 P2 entrou na fila de alta prioridade.
 P3 executou por 1 u.t.
 P3 foi para a fila de E/S (Fita).

 ALTA PRIORIDADE  :
   ┌──────────┬──────────┐
   │   P1     │   P2     │
   └──────────┴──────────┘
 BAIXA PRIORIDADE : Vazia

 DISCO : Vazia

 FITA :
   ┌──────────┐
   │   P3     │
   └──────────┘

┌──────────────────────────────────────────┐
│         >>> INSTANTE 03 <<<              │
└──────────────────────────────────────────┘
 P0 entrou na fila de alta prioridade.
 P1 executou por 1 u.t.
```

```
75   P1 foi para a fila de E/S (Fita).
76   P3 executou 1 u.t. da sua E/S de Fita, faltam 2 u.t.
77
78   ALTA PRIORIDADE  :
79     ┌──────────┬──────────┐
80     │   P2     │   P0     │
81     └──────────┴──────────┘
82   BAIXA PRIORIDADE : Vazia
83
84   DISCO : Vazia
85
86   FITA :
87     ┌──────────┬──────────┐
88     │   P3     │   P1     │
89     └──────────┴──────────┘
90
91  ┌──────────────────────────────────────┐
92  │         >>> INSTANTE 04 <<<          │
93  └──────────────────────────────────────┘
94   P2 executou por 1 u.t.
95   P2 foi para a fila de E/S (Disco).
96   P3 executou 1 u.t. da sua E/S de Fita, faltam 1 u.t.
97
98   ALTA PRIORIDADE  :
99     ┌──────────┐
100    │   P0     │
101    └──────────┘
102  BAIXA PRIORIDADE : Vazia
103
104  DISCO :
105    ┌──────────┐
106    │   P2     │
107    └──────────┘
108  FITA :
109    ┌──────────┬──────────┐
110    │   P3     │   P1     │
111    └──────────┴──────────┘
112
113 ┌──────────────────────────────────────┐
```

```
114 │         >>> INSTANTE 05 <<<         │
115 └──────────────────────────────────────┘
116  P0 executou por 1 u.t.
117  P2 executou 1 u.t. da sua E/S de Disco, faltam 1 u.t.
118  P3 executou 1 u.t. da sua E/S de Fita, faltam 0 u.t.
119  P3 finalizou sua E/S de Fita, P3 completou I/O e vai para a fila
     de alta prioridade.
120
121  ALTA PRIORIDADE  :
122    ┌──────────┬──────────┐
123    │   P0     │   P3     │
124    └──────────┴──────────┘
125  BAIXA PRIORIDADE : Vazia
126
127  DISCO :
128    ┌──────────┐
129    │   P2     │
130    └──────────┘
131  FITA :
132    ┌──────────┐
133    │   P1     │
134    └──────────┘
135
136 ┌──────────────────────────────────────┐
137 │         >>> INSTANTE 06 <<<         │
138 └──────────────────────────────────────┘
139  P0 executou por 1 u.t.
140  P2 executou 1 u.t. da sua E/S de Disco, faltam 0 u.t.
141  P2 finalizou sua E/S de Disco, P2 completou I/O e vai para a fila
     de baixa prioridade.
142  P1 executou 1 u.t. da sua E/S de Fita, faltam 2 u.t.
143
144  ALTA PRIORIDADE  :
145    ┌──────────┬──────────┐
146    │   P0     │   P3     │
147    └──────────┴──────────┘
148  BAIXA PRIORIDADE :
149    ┌──────────┐
150    │   P2     │
```

```
    └──────────┘
  DISCO : Vazia

  FITA :
    ┌──────────┐
    │   P1     │
    └──────────┘


┌──────────────────────────────────────┐
│         >>> INSTANTE 07 <<<          │
└──────────────────────────────────────┘
  P0 executou por 1 u.t.
  P0 finalizou sua execucao.

[ TURNAROUND DE P0 ] : 5

  P1 executou 1 u.t. da sua E/S de Fita, faltam 1 u.t.

  ALTA PRIORIDADE  :
    ┌──────────┐
    │   P3     │
    └──────────┘
  BAIXA PRIORIDADE :
    ┌──────────┐
    │   P2     │
    └──────────┘
  DISCO : Vazia

  FITA :
    ┌──────────┐
    │   P1     │
    └──────────┘


┌──────────────────────────────────────┐
│         >>> INSTANTE 08 <<<          │
└──────────────────────────────────────┘
  P3 executou por 1 u.t.
  P1 executou 1 u.t. da sua E/S de Fita, faltam 0 u.t.
```

```
 P1 finalizou sua E/S de Fita, P1 completou I/O e vai para a fila de alta prioridade.

 ALTA PRIORIDADE  :
  ┌──────────┬──────────┐
  │   P3     │   P1     │
  └──────────┴──────────┘
 BAIXA PRIORIDADE :
  ┌──────────┐
  │   P2     │
  └──────────┘
 DISCO : Vazia

 FITA : Vazia

┌──────────────────────────────────────┐
│         >>> INSTANTE 09 <<<          │
└──────────────────────────────────────┘
 P3 executou por 1 u.t.
 P3 finalizou sua execucao.

[ TURNAROUND DE P3 ] : 10

 ALTA PRIORIDADE  :
  ┌──────────┐
  │   P1     │
  └──────────┘
 BAIXA PRIORIDADE :
  ┌──────────┐
  │   P2     │
  └──────────┘
 DISCO : Vazia

 FITA : Vazia

┌──────────────────────────────────────┐
│         >>> INSTANTE 10 <<<          │
└──────────────────────────────────────┘
 P1 executou por 1 u.t.
```

```
227
228   ALTA PRIORIDADE  :
229     ┌──────────┐
230     │   P1     │
231     └──────────┘
232   BAIXA PRIORIDADE :
233     ┌──────────┐
234     │   P2     │
235     └──────────┘
236   DISCO : Vazia
237
238   FITA : Vazia
239
240 ┌──────────────────────────────────────────┐
241 │         >>> INSTANTE 11 <<<             │
242 └──────────────────────────────────────────┘
243   P1 executou por 1 u.t.
244
245   ALTA PRIORIDADE  :
246     ┌──────────┐
247     │   P1     │
248     └──────────┘
249   BAIXA PRIORIDADE :
250     ┌──────────┐
251     │   P2     │
252     └──────────┘
253   DISCO : Vazia
254
255   FITA : Vazia
256
257 ┌──────────────────────────────────────────┐
258 │         >>> INSTANTE 12 <<<             │
259 └──────────────────────────────────────────┘
260   P1 executou por 1 u.t.
261   P1 sofreu preempcao, vai pra fila de baixa prioridade.
262
263   ALTA PRIORIDADE  : Vazia
264
265   BAIXA PRIORIDADE :
```

```
266    ┌──────────┬──────────┐
267    │   P2     │   P1     │
268    └──────────┴──────────┘
269   DISCO : Vazia
270
271   FITA : Vazia
272
273 ┌──────────────────────────────────────┐
274 │         >>> INSTANTE 13 <<<          │
275 └──────────────────────────────────────┘
276   P2 executou por 1 u.t.
277
278   ALTA PRIORIDADE  : Vazia
279
280   BAIXA PRIORIDADE :
281    ┌──────────┬──────────┐
282    │   P2     │   P1     │
283    └──────────┴──────────┘
284   DISCO : Vazia
285
286   FITA : Vazia
287
288 ┌──────────────────────────────────────┐
289 │         >>> INSTANTE 14 <<<          │
290 └──────────────────────────────────────┘
291   P2 executou por 1 u.t.
292   P2 finalizou sua execucao.
293
294 [ TURNAROUND DE P2 ] : 13
295
296   ALTA PRIORIDADE  : Vazia
297
298   BAIXA PRIORIDADE :
299    ┌──────────┐
300    │   P1     │
301    └──────────┘
302   DISCO : Vazia
303
304   FITA : Vazia
```

```
305
306 ┌──────────────────────────────────────┐
307 │         >>> INSTANTE 15 <<<          │
308 └──────────────────────────────────────┘
309  P1 executou por 1 u.t.
310  P1 finalizou sua execucao.
311
312 [ TURNAROUND DE P1 ] : 15
313
314 Nenhuma fila com processos, o processador está ocioso.
315
316 [+] Todos os processos foram finalizados com sucesso.
317
318 ┌──────────────────────────────────────────┐
319 │            FIM DO ESCALONAMENTO          │
320 └──────────────────────────────────────────┘
321
322 Tempos de turnaround dos processos:
323 ═══════════════════════════ TURNAROUND ═══════════════════════════
324  PID  Tempo de Turnaround   Tempo de Espera
325 ──────────────────────────────────────────────────────────────────
326  P0         5 u.t.            2 u.t.
327  P1         15 u.t.           10 u.t.
328  P2         13 u.t.           10 u.t.
329  P3         10 u.t.           5 u.t.
330 ══════════════════════════════════════════════════════════════════
331  Turnaround médio: 10.75 u.t.
332  Tempo de espera médio: 6.75 u.t.
```

Bibliografia

[1] W. Stallings, *Arquitetura e Organização de Computadores*, 8.º ed. Pearson, 2010.

[2] A. Kumar e A. Tripathy, «CPU Scheduling Techniques: A Review on Novel Approaches, Strategy, and Performance Assessment», *International Journal of Advanced Computer Science and Applications*, vol. 14, n.º 7, pp. 12–24, 2023, doi: 10.14569/IJACSA.2023.0140720.

[3] A. Singh, P. Goyal, e S. Batra, «An Optimized Round Robin Scheduling Algorithm for CPU Scheduling», *International Journal on Computer Science and Engineering*, vol. 2, n.º 7, pp. 2383–2385, 2010, [Online]. Disponível em: https://www.researchgate.net/publication/50194216_An_Optimized_Round_Robin_Scheduling_Algorithm_for_CPU_Scheduling

[4] S. Zouaoui, L. Boussaid, e A. Mtibaa, «Priority based round robin (PBRR) CPU scheduling algorithm», *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, n.º 1, pp. 190–202, 2019, doi: 10.11591/ijece.v9i1.pp190-202.