Departamento de Informática

Licenciatura em Engenharia Informática

Unidade Curricular: Sistemas Operativos

Docente: Paul Andrew Crocker

ProbSched

Um Simulador para Algoritmos de Escalonamento Probabilístico

Elementos do Grupo:

Alessandra Delgado – 51713
Matilde Sequeira – 51752

13 de junho de 2025

Conteúdo

1. Modelos Probabilísticos Utilizados para a Geração de Processos

No simulador foram implementados três modelos probabilísticos para a geração de processo.

- ***Arrival Times:*** Foram implementadas duas distribuições:
  - Distribuição de Poisson : usada para modelar a probabilidade de um número de processos ocorrer num intervalo fixo de tempo
  - Distribuição Exponencial : foi usada para os modelar tempos
- ***Burst Times:*** Foram implementadas duas distribuições:
  - Distribuição Normal : usada para modelar os processos que se agrupam em torno de uma média
  - Distribuição Exponencial : usada para modelar os procesos com curto burst time
- ***Priorities:*** Foram implementadas dois métodos:
  - Distribuição Uniforme : usado para quando as prioridades têm todas a mesma igualdade.
  - Weighted Random : usado quando certas prioridades têm mais probabilidade que outras

2. Algoritmos de Escalonamento Implementados

2.1 Algoritmos Convencionais

2.1.1 *First Come First Served*

O algoritmo mais simples deste projeto, implementado com o auxílio de uma *priority queue*, ordenada por tempo de chegada dos processos, de forma ascendente.

2.1.2 *Priority Scheduling*

Foram implementadas duas versões do escalonamento por prioridade – não-preemptivo e preemptivo. Para ambas as implementações foram utilizadas *priority queues*, ordenadas por prioridade e tempo de chegada (para desempate de prioridades).

2.1.3 *Shortest Job First*

Foram implementadas duas versões do escalonamento por tempo de execução mais curto – não-preemptivo e preemptivo. Para ambas as implementações foram utilizadas *priority queues*, ordenadas por tempo de execução e tempo de chegada (para desempate dos tempos de execução). A sua versão preemptiva pode ser até considerada como *Shortest Remaining Time*, de desempenho preferível.

2.1.4 *Round Robin*

Este algoritmos foi implementado usando um fila FIFO (std::queue), onde o 1º processo a chegar é o 1º a ser executado. Ele aloca a CPU em intervalos de tempo fixos de forma circular, ou seja, cada processo recebe um quantum, e ao expirar esse quantum, volta para o final da fila; o processo continua a ter esse tratamento até se encontrar como terminado, momento em que é retirado da fila. Este algoritmo tem como forma de execução.

2.2 Algoritmos para Sistemas de Tempo Real

2.2.1 *Rate Monotonic*

Este algoritmo de tempo real foi implementado com auxílio um vetor simples, contendo os blocos de controlo de processos de todas as tarefas a simular, que são executados periodicamente.

2.2.2 *Earliest Deadline First*

Este algoritmo preemptivo que dá preferencia a processos com o deadline mais baixo, independentemente da ordem de chegada dos processos. Foi implementado com o auxilio de um vetor de PCB's com ordenação por deadline.

3. Execução do Programa

Este simulador de algoritmos de escalonamento, é executado usando o comando make run. Toda a interface para o utilizador foi implementada com a biblioteca FTXUI, que é baseada em texto. Quando o programa é executado, aparece o menu inicial.

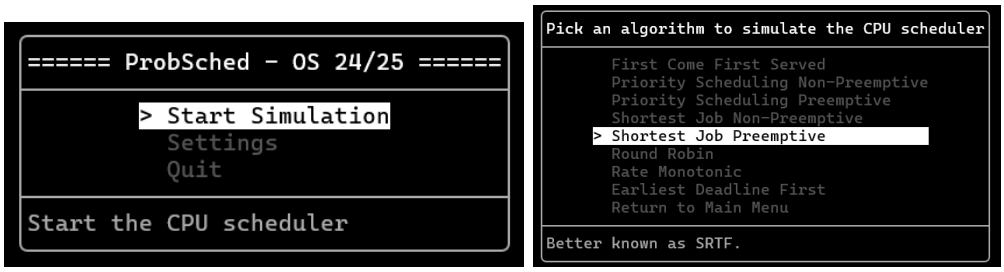

Figura 3.1: Menus: inicial e de algoritmos.

O programa usa 3 métodos de geração de processos:

- Geração Dinâmica (usa distribuições e limite dinâmico)
- Geração por Arquivos (lê os arquivos que se encontram na pasta inputs)
- Geração por Tempo Fixo (usa a geração dinâmica mas limita a execução do algoritmo até ao tempo pedido)

As estatísticas e a geração de processos exibem:

- Diagrama de Gantt (mostra os processos , nos últimos 60 segundos)
- Tabelas (Os que estão em espera na *ready queue*, e os terminados, e lista as tarefas existentes (*real time*)

O programa dispõe ainda de um menu de configurações, onde o utilizador pode fazer alguns ajustes da configuração de processos:

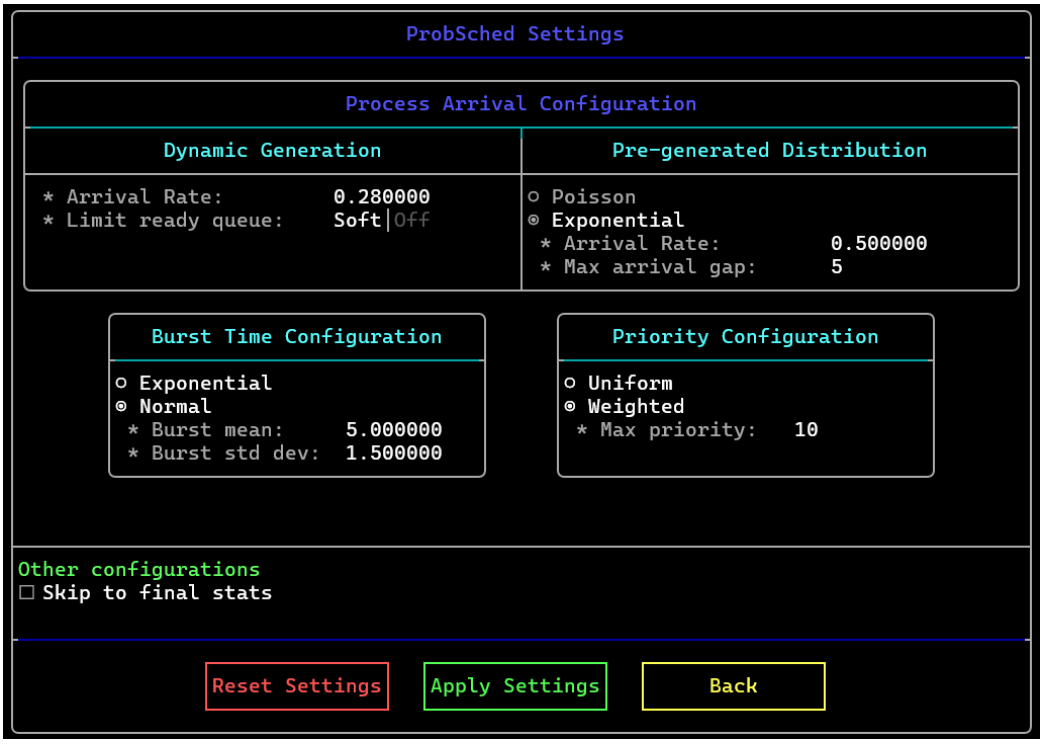

Figura 3.2: Menu de configurações.

4. Conjunto de entradas e saídas de amostra que demonstram os recursos do simulador

4.1 Geração Infinita de Processos

Por exemplo, com o algoritmo de *Round Robin*, com time quantum de 2, sem limite na *ready queue*, e com rácio de 0.28 na chegada.

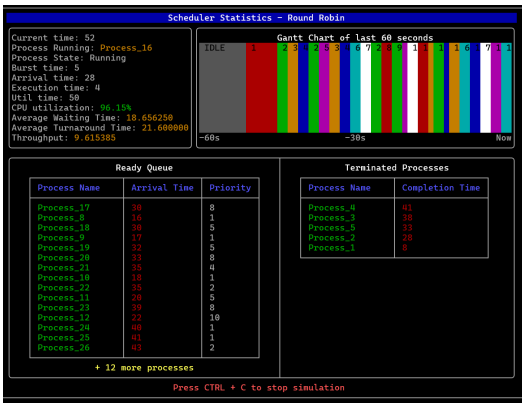

Figura 4.1: Geração infinita de processos com *Round Robin*.

4.2 Execução Até Determinado Instante de Tempo

Por exemplo, com o algoritmo de *Shortest Job First* (preemptivo), com tempo de execução de 60 segundos, com as definições padrão.

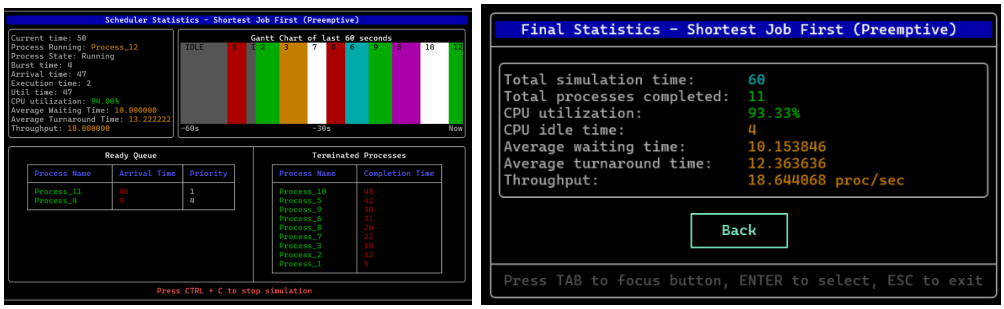

Figura 4.2: Demonstração da execução do algoritmo SRTF.

4.3 Carregamento a Partir de Ficheiro

Por exemplo, com o algoritmo de *First Come First Served*.

Figura 4.3: Demonstração da execução do algoritmo FCFS.

4.4 Geração de um Determinado Número de Processos

Por exemplo, com o algoritmo *Earliest Deadline First*.

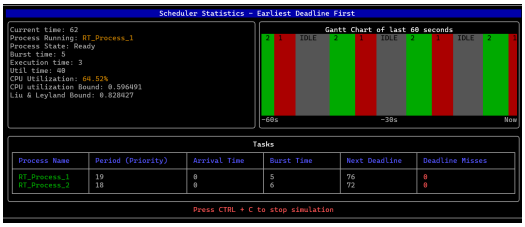

Figura 4.4: Demonstração da execução do algoritmo EDF.

5. Resultados da comparação de desempenho de algoritmos sob diferentes distribuições

5.1 Teste I

Até 10 processos, burst time gerado com exponencial de lambda 0.4 e máximo 2, com prioridade escolhida de forma pesada (máximo 10), e tempos de chegada em Poisson (lambda 0.5). Com estes parâmetros, todos os algoritmos correram de forma semelhante. No entanto, o SJF (preemptivo) foi o que teve melhor desempenho, com maior utilização de CPU e menor tempo de espera.

5.2 Teste II

Até 10 processos, burst time gerado com distribuição uniforme de lambda média de cinco e desvio máximo de um e meio, com prioridade escolhida de forma uniforme (máximo 10), e tempos de chegada em exponencial (lambda 0.5 e máximo 10). Tal como no teste anterior, e como esperado, o melhor algoritmo foi o SJF preemptivo. É de notar que, por mais eficiente que este algoritmo seja, existem certos casos que não consegue tratar por si só, tais como eventos de maior prioridade num sistema operativo, que tivessem de ser resolvidos o mais cedo possível.

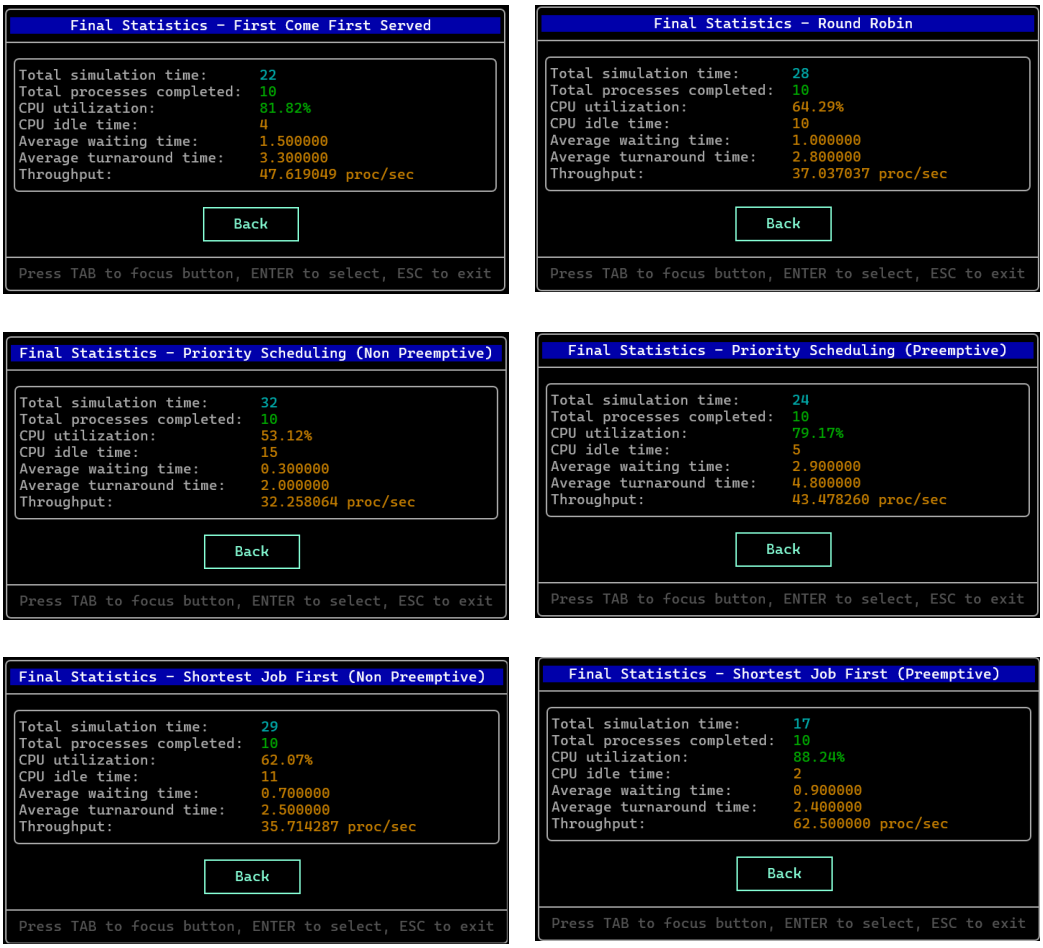

Figura 5.1: Resultados do primeiro teste dos algoritmos gerais.

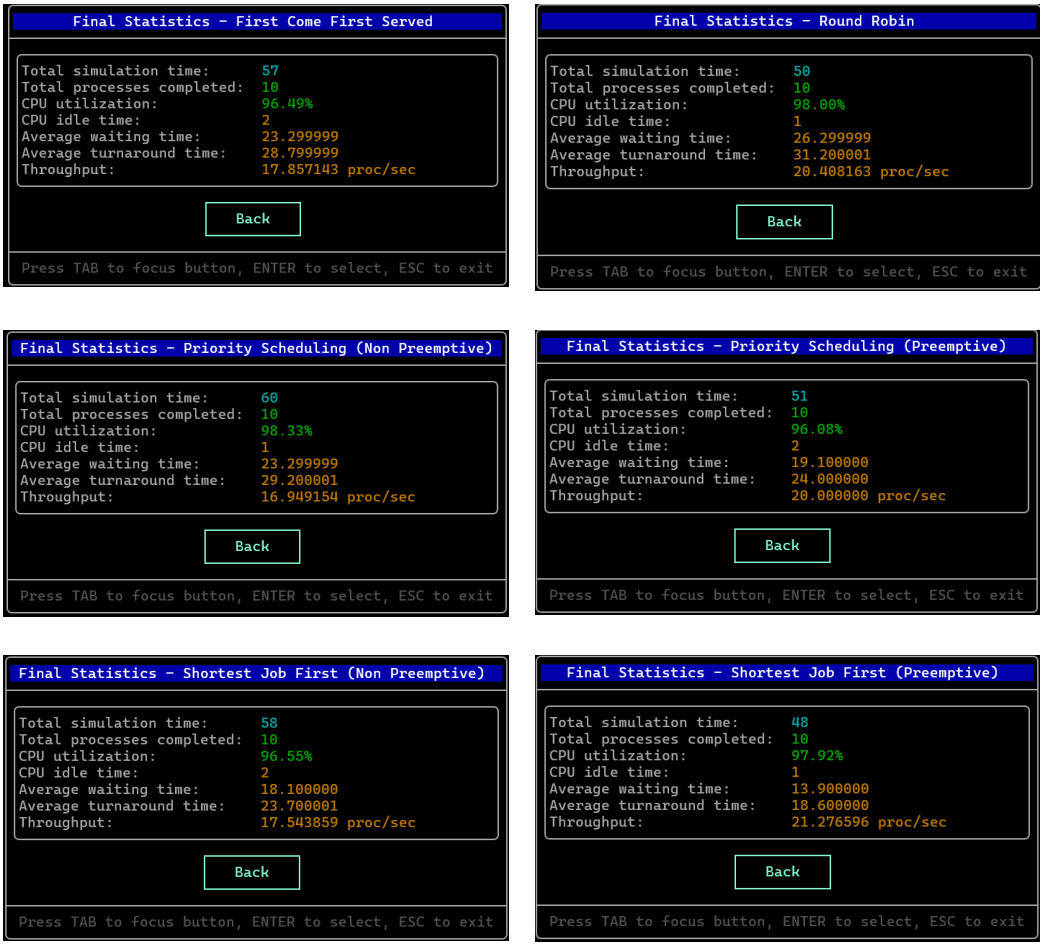

Figura 5.2: Resultados do segundo teste dos algoritmos gerais.