Título do Trabalho : "ProbSched: Um Simulador para Algoritmos de Escalonamento Probabilístico para Sistemas Operativos"

Versão 1.0 Date 12-3-2025

Grupos : 3-4 Alunos.

Objetivos

O objetivo deste projeto é projetar e implementar uma ferramenta de simulação que demonstre e compare o comportamento de vários algoritmos de escalonamento da CPU e assim simule o funcionamento das operações de gestão e execução de processos por um sistema operativo incluindo as operações de escalonamento do CPU, criação e terminação de processos e a comutação de contexto.

Os tempos de chegada dos processos e os tempos de execução da CPU (burst time) serão gerados usando distribuições probabilísticas, proporcionando uma simulação mais realista do escalonamento em sistemas operativos.

O trabalho será descrito dum modo geral para que haja uma grande variedade de interpretações e implementações possíveis! Faz parte do seu trabalho inventar falhas na descrição do simulador e esclarecê-los no relatório do trabalho e apresentação.

Descrição do Projeto

Crie um simulador de processos em uma linguagem de programação de sua escolha (Escolha entre: C, C++, Java, Python, Ocaml ou Rust) que modele uma CPU simples (única) e uma fila de processos. A sua aplicação deve considerar (a) listas estaticas de processos a executar e (b) ser capaz de gerar processos dinamicamente com base em distribuições probabilísticas especificadas e (c) deverá simular vários algoritmos de escalonamento.

- Deverá desenvolver o seu código duma forma modular e usar um Gestor de Compilação (build tool): make ou gmake para C/C++, Ant/Maven para Java, cargo para Rust etc.
- Cada elemento do grupo deve criar uma conta individual no gitlab/github. Um membro do grupo poderá criara o projeto inicial e partilhar com os outros membros do seu grupo e com os professores.

Characteristics

1. **Input**:
    - Scheduling Algorithm to simulate.
    - Maximum Time to Execute or the Number of processes to simulate.
    - Time quantum (for Round Robin scheduling).
    - For a static list of processes - the list of processes
        - For General a periodic processes: List: id, start time, burst time, priority
        - For Periodic Real Time Processes List : id, start time, burst time, period
    - For random process generation
        - Parameters for process generation, including:
        - **Arrival Times**:
            - Generated using a probabilistic distribution (e.g., Poisson distribution for inter-arrival times).
        - **CPU Burst Times**:
            - Generated using a probabilistic distribution (e.g., Normal or Exponential distribution).
2. **Process Generation**:
    - Process IDs should be created sequentially.
    - Processes can be generated from a statically given list
    - Processes can be generated randomly
        - Use probabilistic distributions to dynamically generate:
            - Arrival times (Use a Poisson/Exponential distribution to model realistic arrival times).
            - CPU burst times (Use an Exponential distribution for short bursts or Normal distribution for more variable bursts).
            - Priorities (e.g., uniformly distributed or based on a weighted random selection).

3. **Supported Scheduling Algorithms**:
    - **First-Come, First-Served (FCFS)**.
    - **Shortest Job (SJ)**.
    - **Priority Scheduling** (Preemptive and Non-Preemptive).
    - **Round Robin (RR)**.
    - **For Real time**
        - **Rate Monotonic**
        - **EDF (Earliest Deadline First)**
    - **Multilevel Queue Scheduling** (optional).
4. **Simulation Output**:
    - Statistics:
        - Average waiting time.
        - Average turnaround time.
        - CPU utilization.
        - Throughput (number of processes completed per x unit time).
        - Dead Line Misses (Real Time Algorithms)

Probabilistic Distributions

1. **Arrival Times**:
    - Use a **Poisson distribution** or **Exponential distribution** for modeling realistic inter-arrival times.
    - Example: Arrival time for a new process = Previous arrival time + Random value from distribution.
2. **CPU Burst Times**:
    - Use **Exponential distribution** for shorter bursts in general with occasional long bursts.
    - **Use a Normal distribution** for burst times centered around an average with some variance.
    - Example: Burst time = Random value from distribution.
3. **Priorities**:
    - Assign priorities randomly using:
        - Uniform distribution.
        - Weighted random sampling (e.g., lower numbers are more likely).

Implementation Steps

1. **Process Data Representation**:
    - Use data structures (e.g., a list or queue of "Process Control Blocks") to represent processes and their attributes, dynamically generated based on distributions.
2. **Process Generator**: Implement a module that:
    - Simulates process arrivals using probabilistic distributions.
    - Assigns CPU burst times and priorities to processes.
    - Feeds the generated processes into the scheduling queue.
3. **Algorithm Implementation**:
    - Implement each scheduling algorithm as a separate function or module.
    - Use the generated processes as input for simulation.
    - **Dynamic Priority Adjustment (optional)**
        - Implement aging to prevent starvation in Priority Scheduling.
4. **Simulation Logic**:
    - Maintain a simulated "clock" to track time.
    - Use queues to manage ready and waiting processes.
    - Update statistics dynamically as the simulation progresses.
5. **Visualization**:
    - Text-based: Display a timeline and process statistics in the console.
    - GUI-based (optional): Use libraries like gnuplot, matplotlib etc. for Gantt charts and performance graphs.
6. **User Interface**:
    - Command-line interface for inputting simulation parameters and selecting algorithms. That is to create an input file of parameters to be read
    - Advanced (optional): Build a GUI for easier interaction.
7. **Random** Number Libraries

    (i) Numerical Recipes in C chapter 7 – you can just copy the C code
    https://s3.amazonaws.com/nrbook.com/book_C210.html

    (ii) Mix and Match c++ with c
    a. https://en.cppreference.com/w/cpp/numeric/random
    b. Use the boost library or gsl etc.

NOTA Importante – Geradores de números (psuedo) aleatórios geram a mesma sequência de números aleatórios a partir do mesmo semente (seed). Isto pode ser importante quando quer comparara algoritmos de escalonamento

Deliverables

1. Source code for the scheduling simulator, including process generation.
2. A report
    - (3–5 pages) explaining:
        1. The probabilistic models used for process generation.
        2. Which scheduling algorithms were implemented.
        3. How the program is executed
        4. A set of sample inputs and outputs demonstrating the simulator's capabilities.
    - (2-5 pages)
    - Results for the Performance comparison of algorithms under different distributions.

Evaluation Criteria

1. **Correctness**:
    - The simulator correctly implements the scheduling algorithms and process generation.
2. **Probabilistic Modeling**:
    - Accurate use of probabilistic distributions for process generation.
3. **Performance Metrics**:
    - Accurate calculation of waiting time, turnaround time, CPU utilization, and throughput.
4. **Report** :
    - The report is clear, concise, and covers all required elements.

Extension Ideas

1. **Simulating Multi-Core Systems**:
    - Extend the simulator to support multi-core scheduling and load balancing.
2. **I/O-Bound Processes**:
    - Include a model for I/O operations, where processes spend part (probabilistic) of their time in I/O waiting queues.
3. **Visualization** (optional)

Gantt charts or other visualizations that are clear and informative

Example: A Gantt chart showing process execution order and times. This can be created manually for specific examples or better automatically and even interactively

Interactive Visualization (Optional):

- A graphical or textual representation of the execution timeline.
- Highlight differences in outcomes for each scheduling algorithm.

Comparison Mode (optional)

- Allow users to compare the performance of different algorithms under different probabilistic process distributions.

ASPETOS IMPORTANTES

- O Projeto tem que compilar e executar corretamente num sistema operativo POSIX Linux/MacOS e pelo menos a partir da linha de comando.
- O trabalho consiste no código fonte, ficheiros de dados, Build Tool file (obrigatório) (Makefile ou Java Build File etc) e um relatório.
- O relatório deverá incluir um breve resumo e explicação do funcionamento do programa e os algoritmos implementados, exemplos da execução do programa e verificação com resultados teóricos. Deverá destacar o trabalho realizado por cada elemento do grupo e incluir o url do projeto no gitlab/github.
- Não deve ser necessário incluir muito código no relatório apenas os protótipos das funções usadas.
- Prazos
    - Entregar - Dia 27 de Abril de 2025.
    - Apresentações – a combinar a partir desta data

INSCRIÇÃO DO GRUPO

Feita usando um *form* de Micrososft/OneDrive – a disponibilizar via Moodle

PROGRAMAÇÃO - HINTS

Muitos livros e sites da Internet disponibilizam código para implementar estruturas de dados como **filas** (FIFO) e **filas de prioridades** etc. **utilizam à vontade** .. desde que inclua as referências !