Relatório do Trabalho Prático II (TP II)

Filas e Métodos de Ordenação

Nome dos alunos:
Maurício de Oliveira Santos Rodrigues
João Pedro Seabra Nogueira

Professora: Karla A S Joriatti

3 de fevereiro de 2026

Resumo

Este relatório descreve a implementação de um simulador de escalonador de processos para um Kernel de sistema operacional. O sistema utiliza uma Fila Dinâmica Encadeada e métodos híbridos de ordenação (*Merge Sort* para a carga inicial e *Insertion Sort* para a manutenção da fila). O objetivo é garantir que processos com maior prioridade e menor tempo de chegada sejam atendidos respeitando o limite de ciclos de CPU.

Sumário

1 Implementação

1.1 Visão geral

O programa simula um escalonador que opera sob o regime de *Time Sharing*. Cada processo possui um identificador (ID), tempo de chegada, prioridade e o número total de ciclos necessários. O escalonador processa cada item da fila concedendo até 100 ciclos por vez.

1.2 Arquitetura e TADs

A implementação foi modularizada para facilitar a manutenção e legibilidade:

- **TAD Processo:** Estrutura que armazena os dados vitais de cada tarefa.
- **TAD FilaProcessos:** Uma lista encadeada dinâmica composta por `Celulas`. A estrutura da fila mantém ponteiros para o `topo` e para o `ultimo` elemento, permitindo manipulações eficientes.

1.3 Funcionamento das principais funções

- `criarFila`**:** Responsável por alocar a fila e realizar a ordenação inicial dos processos via *Merge Sort*.
- `mergeSort`**:** Implementado de forma recursiva para ordenar a lista encadeada com complexidade $O(n \log n)$. A ordenação prioriza o menor valor de prioridade e, em caso de empate, o menor tempo de chegada.
- `adicionaFila`**:** Utiliza o algoritmo *Insertion Sort* para reinserir processos que não terminaram sua execução, garantindo que a fila permaneça sempre ordenada por prioridade após o reescalonamento.
- `escalonador`**:** Controla o ciclo de vida dos processos, decrementando os ciclos e incrementando a prioridade caso o processo exceda o limite de 100 ciclos.

2 Impressões gerais

A implementação consolidou nosso conhecimento sobre estruturas de dados dinâmicas e algoritmos de ordenação aplicados a cenários reais, como o escalonamento de processos em um núcleo de sistema operacional. Migrar da lógica de vetores estáticos para uma fila encadeada dinâmica trouxe desafios técnicos consideráveis, exigindo um controle rigoroso da memória e uma manipulação precisa de ponteiros em estruturas que mudam de tamanho e ordem o tempo todo.

Um dos pontos altos foi o desenvolvimento da função *merge* para listas encadeadas. Diferente dos vetores, onde o acesso é direto por índices, na lista encadeada precisamos de um cuidado redobrado com os ponteiros *prox* para não perdermos referências de nós nem criarmos ciclos infinitos. Essa etapa foi essencial para entendermos como algoritmos clássicos de "divisão e conquista"se comportam em diferentes formas de armazenamento.

Além disso, trabalhar com uma ordenação híbrida — usando *Merge Sort* para a carga inicial e *Insertion Sort* para as reinserções — mostrou que a escolha do algoritmo deve depender do estado atual dos dados. Percebemos na prática como o *Insertion Sort* é eficiente para manter a fila organizada após pequenas mudanças, o que reforça que nem sempre o algoritmo com a melhor complexidade teórica global é a melhor escolha para todas as situações de um software.

No fim, nossa maior motivação foi aplicar conceitos teóricos de Sistemas Operacionais e Estruturas de Dados em algo prático. Garantir que o programa estivesse livre de vazamentos de memória (*memory leaks*), validando tudo com o *Valgrind*, aproximou este trabalho acadêmico das práticas reais de desenvolvimento de baixo nível, onde eficiência e estabilidade são fundamentais.

3 Análise de Complexidade

3.1 Tempo

A ordenação inicial via *Merge Sort* apresenta complexidade $O(n \log n)$. A manutenção da fila através do *Insertion Sort* possui complexidade $O(n)$ no pior caso. O escalonamento total depende da soma dos ciclos de todos os processos divididos pelo *quantum* de tempo.

3.2 Espaço

A complexidade de espaço é $O(n)$, onde n é o número de processos, devido à alocação dinâmica de uma célula para cada processo na memória.

4 Conclusão

O trabalho foi concluído com sucesso, atendendo a todos os requisitos do enunciado. A maior dificuldade encontrada foi garantir a estabilidade da ordenação na lista encadeada e o tratamento correto dos ponteiros para evitar vazamentos de memória, os quais foram verificados e eliminados com o auxílio da ferramenta *Valgrind*.

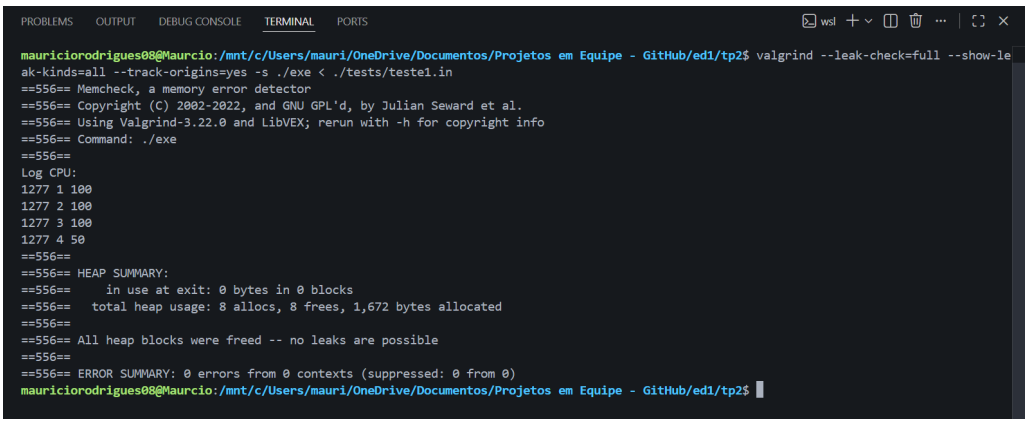

Figura 1: Print Valgrind