DEZEMBRO DE 2024

SISTEMAS OPERATIVOS

ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA

Universidade do Minho

José Novais - A105056
Miguel Machado - A103668
Tiago Diogo - A103665

02

INTRODUÇÃO

Este relatório apresenta a implementação de uma solução em C para o processamento e análise de dados de sensores de temperatura localizados em diferentes regiões do país. A solução é composta por três programas:

- **sort: Ordena os registos de temperatura de uma região.**
- **stats: Calcula estatísticas como média, mediana, valor máximo e mínimo para uma região.**
- **report: Agrega e apresenta as estatísticas de todas as regiões, identificando as que se destacam em cada categoria.**

A implementação segue os requisitos descritos no enunciado do trabalho, incluindo a valorização de funcionalidades adicionais.

ARQUITETURA DE PROCESSOS

A solução utiliza três programas que interagem por meio de pipes anônimos e chamadas ao sistema para garantir a eficiência.
Estrutura Geral:

- **O programa Sort ordena os registos de uma região diretamente no ficheiro binário. Este programa utiliza buffers para evitar carregar todos os dados em memória, atendendo à restrição de memória limitada.**
- **O programa Stats invoca o Sort, calcula as estatísticas da região e envia-as para o stdout ou grava-as em ficheiros binários.**
- **O programa report paraleliza a execução de várias instâncias do stats, recebendo as estatísticas por um pipe anônimo para geração do relatório final.**

03

MOTIVAÇÃO E CONTEXTO

Com o aumento da quantidade de dados gerados por sensores, especialmente em cenários climáticos, torna-se necessário desenvolver soluções eficientes para processar e analisar essas informações.

Este projeto visa proporcionar:

- **Eficiência em Memória: Processamento de grandes volumes de dados com restrição de memória.**
- **Paralelismo: Utilização de múltiplos processos para reduzir o tempo total de execução.**
- **Modularidade: Desenvolvimento de programas independentes que possam ser reutilizados e escalados para outras aplicações.**

A escolha de comunicação por pipes anônimos e o uso de chamadas ao sistema permitem que o projeto explore as funcionalidades nativas do sistema operacional Linux, garantindo alta performance.

04

CÓDIGO SORT

Objetivo:

Ordenar os registos de temperatura de uma região diretamente no ficheiro binário, utilizando buffers para leitura e escrita.

Funcionalidades Implementadas:

- **Leitura em blocos: Utiliza um buffer de tamanho BUFFER_SIZE para processar os registos por partes, respeitando a restrição de memória.**
- **Ordenação por contagem: Implementa um algoritmo eficiente de contagem para ordenar os registos sem alocações extras.**
- **Escrita direta no ficheiro: Os dados ordenados são gravados diretamente no ficheiro binário, eliminando a necessidade de ficheiros temporários.**

Principais Chamadas ao Sistema:

- **open, read, write, pread, pwrite: Para manipulação do ficheiro binário.**
- **close: Para liberar os recursos alocados.**

```
// Inicializar o array de contagem para ordenar os dados
int counts[RANGE];
for (int i = 0; i < RANGE; i++) {
    counts[i] = 0;
}

// Ler os dados da região em blocos e atualizar as contagens
while (records_read < total_records) {
    int to_read = BUFFER_SIZE;
    if (total_records - records_read < BUFFER_SIZE) {
        to_read = total_records - records_read;
    }

    // Ler os dados para o buffer
    ssize_t bytes_read = pread(fd, buffer, to_read * sizeof(int), current_offset);
    for (int i = 0; i < to_read; i++) {
        counts[buffer[i] - MIN_TEMP]++;
    }
    records_read += to_read;
}

// Escrever os dados ordenados de volta ao ficheiro
for (int i = 0; i < RANGE; i++) {
    for (int j = 0; j < counts[i]; j++) {
        int temp = i + MIN_TEMP;
        pwrite(fd, &temp, sizeof(int), current_offset);
        current_offset += sizeof(int);
    }
}
```

05

STATS

Objetivo:

Calcular estatísticas de uma região e enviá-las para o stdout ou gravá-las em ficheiros binários.
Funcionalidades Implementadas:

- **Invocação do sort: Garante que os registos da região estejam ordenados antes do cálculo.**

Cálculo de estatísticas:

- **Média: Soma dos valores dividida pelo número de registos.**
- **Mediana: Determinada com base nos registos ordenados.**
- **Máximo e Mínimo: Identificados durante o processamento dos dados.**

Saída adaptável:

- **Envia as estatísticas diretamente para o stdout.**
- **Grava as estatísticas em ficheiros binários nomeados por região.**

Principais Chamadas ao Sistema:

- **fork e execlp: Para criar um processo filho e invocar o sort.**
- **read, write: Para manipulação de ficheiros e comunicação.**
- **waitpid: Para aguardar a conclusão do processo filho.**

```c
// Criar um processo filho para executar o sort
pid_t pid = fork();
if (pid == 0) {
    // Processo filho executa o sort
    execlp("./sort", "./sort", sensor_data_file, argv[2], NULL);
    _exit(1); // Sai caso o execlp falhe
} else {
    // Processo pai aguarda o término do filho
    int status;
    waitpid(pid, &status, 0);
    if (WEXITSTATUS(status) != 0) {
        write(STDERR_FILENO, "Erro ao executar sort.\n", 23);
    }
}
```

06

REPORT

- **Objetivo:**

Agregá-las estatísticas de todas as regiões e identificar os destaques em cada categoria.

Funcionalidades Implementadas:

- **Execução paralela:**
- **Cria processos filhos para executar o stats simultaneamente, limitado pelo número máximo de processos.**

Comunicação via pipe:

- **Utiliza pipes anônimos para receber as estatísticas geradas pelo stats.**

Agregação de dados:

- **Identifica regiões com valores máximos, mínimos, médios máximos e mínimos.**

Geração de relatório:

- **Imprime as estatísticas detalhadas de cada região e os destaques no stdout.**

Principais Chamadas ao Sistema:

- **fork, execlp: Para criar processos filhos e invocar o stats.**
- **pipe, read, write: Para comunicação via pipe anônimo.**
- **waitpid: Para sincronização dos processos filhos.**

```c
// Criar um pipe anônimo para comunicação
int pipefd[2];
pipe(pipefd);

// Criar processos filhos para cada região
for (int region = 1; region <= num_regions; region++) {
    pid_t pid = fork();
    if (pid == 0) {
        // Redirecionar o stdout para o lado de escrita do pipe
        dup2(pipefd[1], STDOUT_FILENO);
        close(pipefd[0]);
        close(pipefd[1]);

        // Executar o programa stats
        execlp("./stats", "./stats", sensor_data_file, region_str, "stdout", NULL);
        _exit(1); // Sai caso o execlp falhe
    }
}

// Ler os dados das regiões a partir do pipe
close(pipefd[1]);
for (int i = 0; i < num_regions; i++) {
    read(pipefd[0], &stats, sizeof(region_stats));
    // Processar estatísticas recebidas...
}
```

07

VALORIZAÇÃO

Saída pelo stdout no stats:

- **Implementado um argumento adicional para enviar as estatísticas diretamente para o stdout.**

Controle de Processos no report:

- **Foi adicionada a capacidade de limitar o número de processos simultâneos para execução do stats.**
- **Implementado um mecanismo de controle que utiliza waitpid para gerenciar os processos ativos.**

Escalabilidade:

- **Os testes demonstraram que a arquitetura suporta eficientemente até 180 regiões com 200.000 registos cada. O uso de buffers e comunicação via pipes minimizou o consumo de memória e otimizou a velocidade de execução.**

08 TESTES E RESULTADOS

Cenários Testados:

Ficheiros Pequenos:

- **Número de regiões: 5.**
- **Registos por região: 100.**

Ficheiros Grandes:

- **Número de regiões: 180.**
- **Registos por região: 200.000.**

Desempenho:

- **O programa sort demonstrou eficiência na ordenação em ficheiros grandes, mesmo sob restrições de memória.**
- **O programa report paralelizou eficientemente a execução do stats, respeitando o limite de processos.**

Resultados Gerados:
Os resultados corresponderam às expectativas, com destaque correto para as regiões com valores máximos e mínimos.

09

CONCLUSÃO

A implementação apresentada alcançou com sucesso os objetivos definidos, oferecendo uma solução eficiente e escalável para o processamento e análise de dados de sensores de temperatura. A arquitetura baseada em processos, combinada com o uso de comunicação via pipes e chamadas ao sistema, mostrou-se robusta e alinhada às boas práticas em sistemas operativos.

Destaques:

- **A modularidade dos programas sort, stats e report permite uma manutenção e extensibilidade simples.**
- **O uso de pipes anônimos assegurou uma comunicação eficiente e direta entre processos, eliminando dependências de ficheiros temporários.**
- **A implementação das funcionalidades valorizadas, como o controle de processos no report e a saída pelo stdout no stats, incrementou a flexibilidade e a eficiência da solução.**

Limitações e Melhorias Futuras:

- **Desempenho em cenários extremos:**
- **Embora o sistema seja escalável, testes em cenários com milhões de registos podem identificar oportunidades de otimização adicional.**
- **Monitoramento e Logs:**
- **Adicionar logs detalhados para cada etapa do processamento ajudaria na depuração e monitoramento em tempo real.**

Automação de Testes:

- **Criar um conjunto de testes automatizados para validar a funcionalidade em diferentes cenários e configurações.**
- **A abordagem adotada demonstra um equilíbrio eficaz entre simplicidade e desempenho, tornando-a ideal para aplicações em larga escala de análise de dados sensoriais.**