UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Atividade 1

Professores: Giovani Gracioli e Márcio Castro

Alunos: Augusto de Hollanda Vieira Guerner (22102192) e
Micael Angelo Sabadin Presotto (22104063)

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

1. INTRODUÇÃO

Este documento tem por objetivo delinear sobre a solução encontrada pelo grupo referente à primeira atividade prática da matéria de sistemas operacionais. No que concerne a tarefa, foi exigido que se fizesse um simulador de algoritmos de escalonamento (First come, First Served; Shortest Job First; Por prioridade, sem preempção; Por prioridade, com preempção por prioridade; Round-Robin com quantum = 2s, sem prioridade). No mais, a estrutura deste relatório será bem simples: primeiro se fala das 3 classes bases (as que são imprescindíveis para o sistema); em seguida se comenta acerca das implementações dos algoritmos; depois é abordado as métricas; por fim, algumas considerações a respeito de classes utilitárias e outras coisas.

2. Kernel, CPU E Escalonador

Este tópico discute sobre as três principais classes do projeto: kernel; cpu e escalonador.

2.1. Kernel

Desde o início nosso grupo pensou que seria interessante uma classe que integrasse todo o sistema. Desse modo, foi criado o kernel. Ele tem as seguintes funções: controlar o fluxo de trabalho; administrar os recursos; e fazer a ponte de comunicação entre escalonador e a cpu.

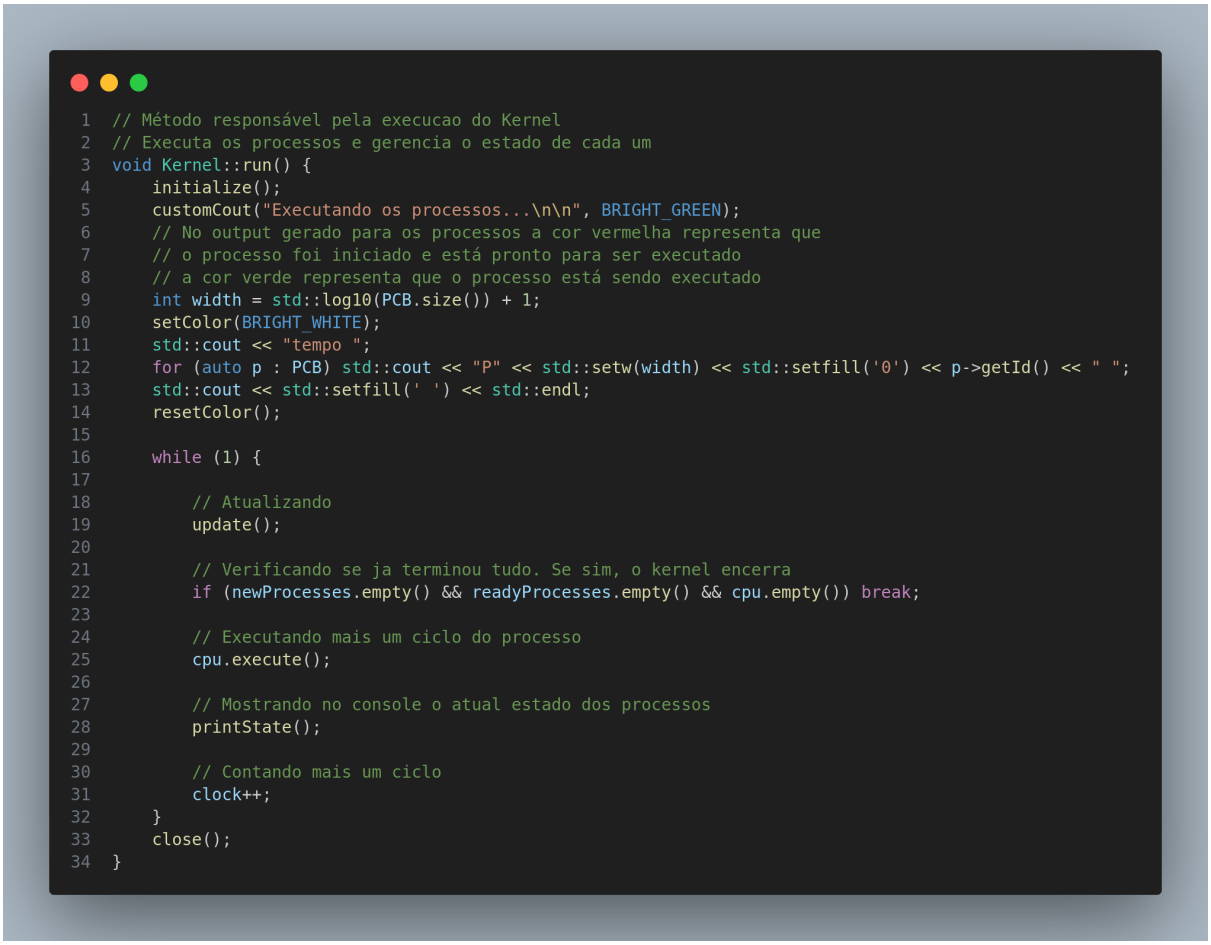
Primeiramente, no que se refere ao controle de fluxo de trabalho, o kernel, como ilustrado na figura 1, controla cada ciclo de execução. Para tanto, existe um while dentro de seu método run, que executa passo a passo e termina só quando não tem mais processos a serem executados. Além disso, é o kernel quem define quando ler os arquivos de entrada.

Ademais, o kernel também faz o gerenciamento dos recursos. Ele controla a criação dos processos e seus estados; controla o clock do sistema também e os objetos como file_reader, scheduler e cpu

Por último, para o escalonador e para a cpu, o kernel foi construído de tal forma que ele é a ponte de comunicação entre os dois. Como se vê na figura 2, o escalonador indica para o kernel que é hora de trocar de contexto, o kernel novamente requisita o escalonador, só que dessa vez para saber qual é o próximo processo a ser executado, e finalmente, com o novo processo em mãos, o kernel manda a cpu trocar de contexto, descarregando o antigo e carregando o novo. O kernel também faz uma comunicação indireta com o FileReader e ProcessFactory e os outros objetos, uma vez que é ele que manda o leitor de arquivo ler os parâmetros e enviá-os para a ProcessFactory.

Portanto, como fica evidente, o kernel é quem faz toda a integração do sistema, controlando o fluxo de trabalho, gerenciando recursos e sendo a ponte de comunicação entre os elementos integrantes do software. Poderia dar-se outro nome a classe, como System ou Manager, mas tendo em vista os conceitos em aula, optou-se por kernel.

Figura 1 - Fluxo de trabalho definido no método run do kernel.



```cpp
// Método responsável pela execucao do Kernel
// Executa os processos e gerencia o estado de cada um
void Kernel::run() {
    initialize();
    customCout("Executando os processos...\n\n", BRIGHT_GREEN);
    // No output gerado para os processos a cor vermelha representa que
    // o processo foi iniciado e está pronto para ser executado
    // a cor verde representa que o processo está sendo executado
    int width = std::log10(PCB.size()) + 1;
    setColor(BRIGHT_WHITE);
    std::cout << "tempo ";
    for (auto p : PCB) std::cout << "P" << std::setw(width) << std::setfill('0') << p->getId() << " ";
    std::cout << std::setfill(' ') << std::endl;
    resetColor();

    while (1) {

        // Atualizando
        update();

        // Verificando se ja terminou tudo. Se sim, o kernel encerra
        if (newProcesses.empty() && readyProcesses.empty() && cpu.empty()) break;

        // Executando mais um ciclo do processo
        cpu.execute();

        // Mostrando no console o atual estado dos processos
        printState();

        // Contando mais um ciclo
        clock++;
    }
    close();
}
```

Fonte: Acervo dos autores.

Figura 2 -Método que atualiza o estado dos processos e gerencia a troca de contexto.

```cpp
// Método responsável por atualizar o estado dos processsos
// Verifica se já é hora de trocar de processo
// Se for, troca de processo e gerencia o estado do processo atual
void Kernel::update() {
    // Atualizando lista de processos
    updateReadyProcesses();
    // Se for a hora de trocar, troca o processo e faz o gerenciamento do estado do processo atual
    if (scheduler->isItTimeToSwitch(&cpu, readyProcesses)) {
        if (currentProcessRunning) {
            if (currentProcessRunning->finished()) {
                executedProcesses.push_back(currentProcessRunning);
                currentProcessRunning->setEnd(clock);
                currentProcessRunning->setCurrentState(TERMINADO);
            } else {
                readyProcesses.push_back(currentProcessRunning);
                currentProcessRunning->setCurrentState(PRONTO);
            }
        }
        currentProcessRunning = scheduler->getNextProcess(readyProcesses);
        if (currentProcessRunning) {
            readyProcesses.erase(readyProcesses.begin());
            currentProcessRunning->setCurrentState(EXECUTANDO);
        }
        cpu.switchProcess(currentProcessRunning);
        contextSwitches++;
    }
}
```

Fonte: Acervo dos autores.

2.2. CPU

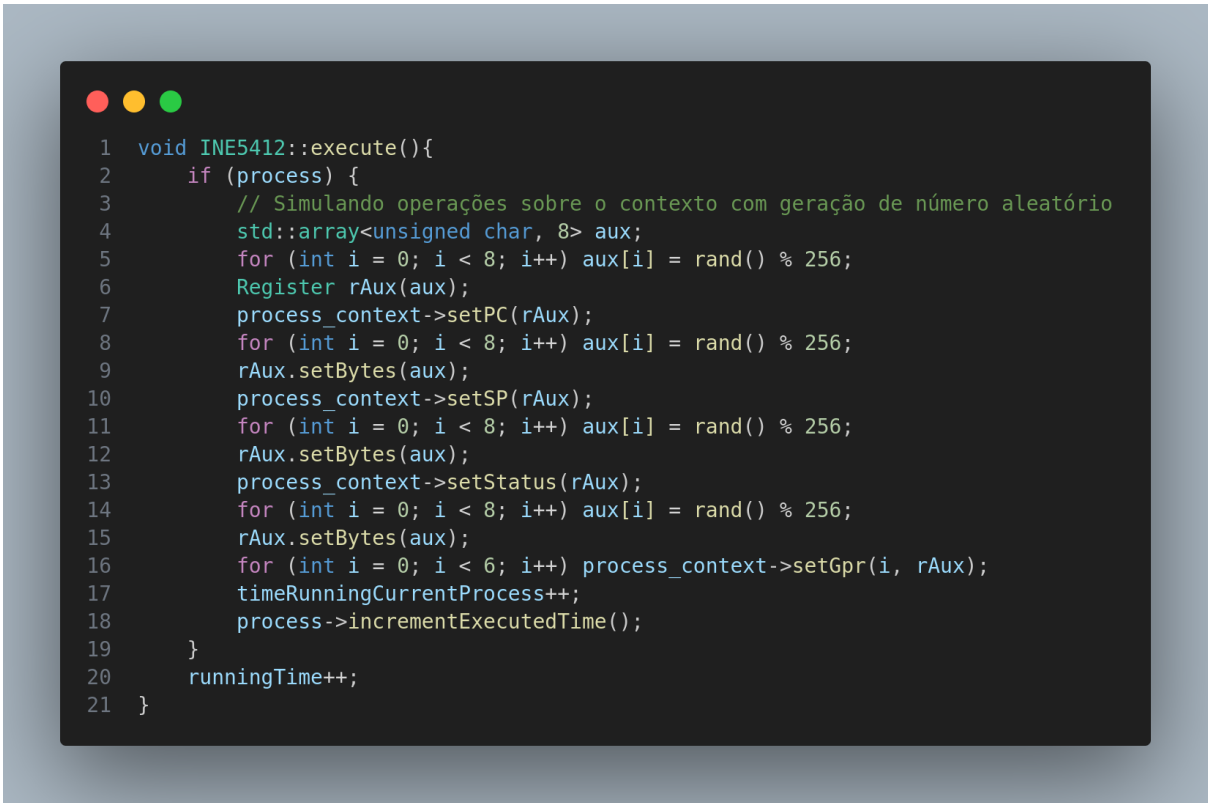
Apesar de ser um elemento indispensável para o hardware, a cpu no nossa atividade tem um caráter muito mais ilustrativo do que necessário. Isso se deve ao fato de que o objeto principal da tarefa não é simular a operação de uma cpu, mas sim simular algoritmos de escalonamento. Assim, as operações que a cpu faz de acordo com cada processo são totalmente arbitrárias, isto é, não tem sentido lógico, é apenas para “consumir” clock do sistema e simular uma execução.

Primeiramente, foi feita uma cpu genérica e uma concreta, chamada de INE5412. A concreta implementa os métodos execute, load e unload para especialmente suportar o contexto definido pela atividade de um registrador para SP, PC, status e 6 registradores de propósito geral.

No mais, no que tange às operações feitas sobre o contexto do processo, faz-se apenas uma atribuição a cada um de valores aleatórios. A figura 3 mostra como ficou.

Conclui-se que a cpu, embora seja muito mais de caráter ilustrativo, ela atende os requisitos da atividade, já que tem possui uma natureza abstrata capaz de suportar vários tipos de contexto.

Figura 3 - Operações aleatórias para simular um processamento real dos processo.



```
1  void INE5412::execute(){
2      if (process) {
3          // Simulando operações sobre o contexto com geração de número aleatório
4          std::array<unsigned char, 8> aux;
5          for (int i = 0; i < 8; i++) aux[i] = rand() % 256;
6          Register rAux(aux);
7          process_context->setPC(rAux);
8          for (int i = 0; i < 8; i++) aux[i] = rand() % 256;
9          rAux.setBytes(aux);
10         process_context->setSP(rAux);
11         for (int i = 0; i < 8; i++) aux[i] = rand() % 256;
12         rAux.setBytes(aux);
13         process_context->setStatus(rAux);
14         for (int i = 0; i < 8; i++) aux[i] = rand() % 256;
15         rAux.setBytes(aux);
16         for (int i = 0; i < 6; i++) process_context->setGpr(i, rAux);
17         timeRunningCurrentProcess++;
18         process->incrementExecutedTime();
19     }
20     runningTime++;
21 }
```

Fonte: Acervo dos autores.

2.3. Escalonador

Certamente o escalonador é um elemento importantíssimo para o trabalho, uma vez que é ele quem vai garantir o escalonamento dos processos de forma correta e é quem vai fazer a interface com os algoritmos de escalonamento. Ele funciona utilizando dois métodos: isItTimeToSwitch e getNextProcess, ambos são essencialmente chamadas para o algoritmo de escalonamento.

Referentemente ao primeiro método exposto (isItTimeToSwitch), como o próprio nome induz, ele é responsável por indicar ao kernel se é hora ou não de trocar de processo em execução. Esse método, como já dito, é acionado todo ciclo de execução pelo kernel e, quando ele retornar verdadeiro, o kernel se comunica novamente com o escalonador, chamando getNextProcess (próxima discussão), e enviando o novo processo para a cpu.

Já no que concerne o segundo método, ou seja, getNextProcess, quando requisitado com os devidos parâmetros (a lista de processos prontos), ele extrai da lista o próximo processo a ser executado. Extrai, pois ele altera a lista de tal forma que retira o processo selecionado dela e o retorna. Como comentado, ele é chamado pelo kernel, após o próprio kernel ser indicado que é necessário a troca de contexto.

A partir do supracitado, torna-se evidente que o escalonador é alma do simulador, tendo em vista que é ele quem garante o correto funcionamento do programa. Claramente, a corretude dos algoritmos também é imprescindível para que o simulador funcione.

2.4. Relação entre eles

A relação entre esses três componentes principais, como já descrito sucintamente, é bastante simples. O kernel fica responsável por ser a ponte entre escalonador e cpu; já a cpu será quem executa o processo que o escalonador escolhe, por meio da requisição do kernel; por fim, o escalonador, além de indicar o próximo processo, também define a hora que se deve trocá-lo para kernel assim mandar a cpu trocar.

3. ALGORITMOS

Os algoritmos são a verdadeira essência da atividade, pois é onde está a lógica do escalonamento de processos. Cada algoritmo possui dois métodos básicos: isItTimeToSwitch e getNextProcess

Como descrito sucintamente no tópico 2.3, ambos os métodos são chamados pelo escalonador, ou seja, a descrição do método é muito parecida.

O isItTimeToSwitch fala se é a hora de trocar de contexto. Para, por exemplo, o método Robin-Round, verdadeiro será retornado para quando a cpu atingir o tempo de 1 quantum de execução do mesmo processo ou se o processo terminar antes ou ainda se não tem nenhum processo na cpu e tem processo pronto. Poderia-se falar dos outros, mas a lógica é muito parecida, mudando apenas alguns pormenores.

O getNextProcess obtém, sem extrair da lista passada por parâmetro, o próximo processo a ser executado. Por exemplo, para o algoritmo de prioridade sem preempção, ele retornará o processo com o maior valor de prioridade. Assim, quando se tem 4 processos com 1, 2, 3 e 4 de prioridade na lista de prontos para serem executados, o próximo escalonado será o com prioridade de 4; possivelmente, o último, caso nenhum seja inserido na lista de prontos, a ser executado será o de prioridade 1.

No mais, é interessante destacar que, a fim de respeitar uma das mais importantes exigências tanto na produção de software quanto para a atividade, foi utilizado o padrão strategy para aumentar a extensibilidade do projeto.

Resumidamente, percebe-se a notoriedade dos algoritmos para a simulação. Além disso, a utilização do padrão de projeto strategy torna mais extensível o projeto bem como vai de acordo com os princípios da orientação a objeto.

4. MÉTRICAS

No geral foi muito simples a implementação dos cálculos das métricas, já que de antemão foi dada a fórmula para cada um delas.

A primeira métrica exigida é o tempo de resposta para cada processo, ou seja, tempo de término menos o tempo do momento da criação. Assim para obter o resultado em código, percorre-se o vetor de processos e adiciona o resultado da diferença do tempo final e do tempo inicial a um outro vetor, em que o primeiro elemento do vetor resultante é o tempo de resposta do primeiro processo, o segundo elemento, o tempo de resposta do segundo processo e assim por diante.

A segunda métrica é referente ao tempo médio de respostas do processo. Com o vetor de tempo de resposta de cada processo calculado anteriormente, torna-se fácil o cálculo da média, visto que é só somar todos os elementos dele e dividir pelo número de processos.

A terceira métrica é o tempo médio de espera. Para calculá-la em código usou-se a mesma lógica da métrica anterior: somar todos elementos de um vetor com o tempo de espera de cada processo e dividir pelo número de processo.

A última métrica é o número de trocas de um contexto. Isso já é calculado à medida que a simulação acontece, assim a classe Analyzer apenas externaliza isso por meio do console.

Na figura 4 está como fica visível no console os resultados. Também na mesma figura tem-se o diagrama de tempo exigido.

Figura 4 -Saída do console.

```
SIMULANDO ROUND ROBIN

Inicializando kernel...

Arquivo entrada.txt

Quantidade de processos lidos do arquivo: 4
Creation time = 0 duration = 5 priority = 2
Creation time = 0 duration = 2 priority = 3
Creation time = 1 duration = 4 priority = 1
Creation time = 3 duration = 3 priority = 4

Processos

ID: 1, Start: 0, Duration: 5, Priority: 2, Executed Time: 0, Current State: 0
ID: 2, Start: 0, Duration: 2, Priority: 3, Executed Time: 0, Current State: 0
ID: 3, Start: 1, Duration: 4, Priority: 1, Executed Time: 0, Current State: 0
ID: 4, Start: 3, Duration: 3, Priority: 4, Executed Time: 0, Current State: 0

Executando os processos...


Encerrando kernel...

METRICAS

tempo P1 P2 P3 P4
  0-1
  1-2
  2-3
  3-4
  4-5
  5-6
  6-7
  7-8
  8-9
 9-10
10-11
11-12
12-13
13-14
14-15
      Tempo de resposta por processo: P1 - 13 ciclos P2 - 4 ciclos P3 - 11 ciclos P4 - 11 ciclos
      Tempo médio de resposta:        9.75 ciclos
      Tempo médio de espera:          6.25 ciclos
      Número de trocas de contexto:   9 trocas

FIM DA SIMULACAO
```

Fonte: Acervo dos autores.

5. CONSIDERAÇÕES

Agora saindo um pouco da parte teórica e indo para uma parte que é mais simples e que não é o foco, tem-se as classes leitor de arquivo e fábrica de processo bem como o utils para deixar a saída do console mais atrativa.

O FileReader foi baseado no dado na atividade, no entanto sofreu grandes mudanças. A primeira é que agora ele apenas faz leitura do arquivo e não cria nem processo ou parâmetros de processo; a segunda é que ele retorna as linhas do arquivo para depois outro

objeto construir com os dados lidos os processos; e a terceira e última é que o caminho do arquivo é passado por parâmetro no método de ler arquivo. A intenção dessas mudanças era desacoplar funcionalidades que não cabem para um leitor de arquivo, como criação de parâmetros de processo.

Já no que se refere ao ProcessFactory, temos efetivamente nele a criação dos processos a partir do arquivo txt, que foi lido pelo FileReader. É bastante trivial a construção de cada processo. Passa-se ou as linhas do arquivo ou os parâmetros para a fábrica construir. Se passar as linhas, a fábrica retorna uma lista de processos, precisamente, uma lista de ponteiros para os processos, ou seja, percorre-se linha por linha criando um processo correspondente. Se passar apenas os parâmetros, ele chama o construtor diretamente e retorna um único processo, ou melhor, um ponteiro para o processo. Vale destacar que o id de cada processo é gerenciado pela factory, embora não sendo a melhor forma de fazê-lo.

Por fim, o utils criado tem por objetivo tornar a saída do terminal mais agradável e intuitiva, basicamente foi implementado no utils funções auxiliares para permitir colorir letras e fundos das letras do terminal. Sabe-se que não é nem perto de uma interface sofisticada como QT, mas é algo melhor que o terminal cru. Assim pede-se que avalie com carinho isso.

CONCLUSÃO

Portanto, como se pode observar, além de atividade ter exigido conhecimentos técnicos aprendidos em aula também foi requerido entendimento de programação orientada a objeto e da linguagem C++. Os conhecimentos envolvidos da parte teórica foram o entendimento dos algoritmos de escalonamento e a estrutura de hardware dos computadores. Já os conceitos de OO, trabalhou-se sempre priorizando os princípios SOLID e buscou-se usar padrões de projetos quando oportuno, como foi o caso do padrão strategy. Resumidamente, com atividade, aprendeu-se muito mais do que apenas o escopo da matéria.

REFERÊNCIAS

- Refactoring Guru, Strategy: 2014-2023. Disponível em: <https://refactoring.guru/design-patterns/strategy>. Acesso em setembro de 2023.