**Relatório Impresso – Implementação SO**

Pedro Yan Ornelas de Oliveira – 14/0158995

Pietro Bertarini Motta – 14/0159118

Vitor Satoru Machi Matsumine – 14/0033262

1. **Descrição das ferramentas/linguagens utilizadas**

Todo (Pietro)

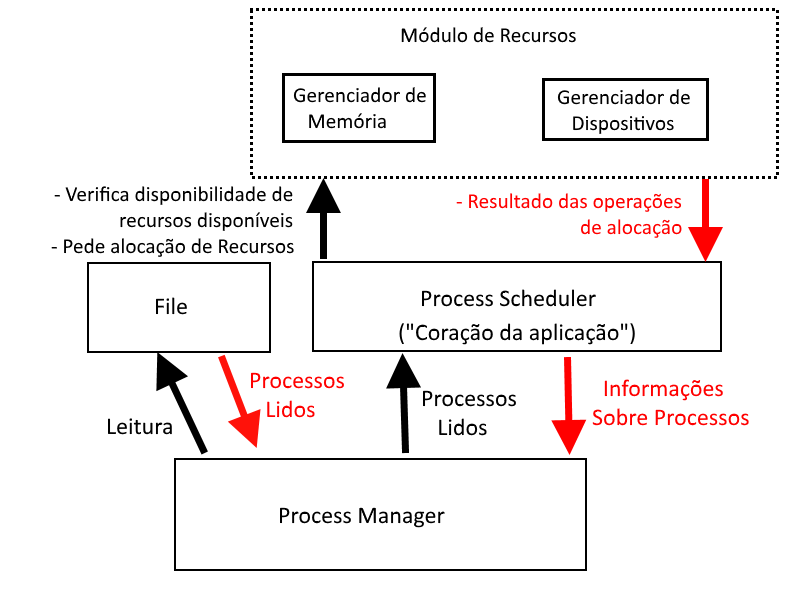
1. **Descrição Teórica e Prática da solução dada**

Para resolver o problema pedido, foi desenvolvido um sistema dividido em dois grandes módulos:

* 1. **Process Manager:** Módulo responsável por todas as operações de processo
  2. **File Manager:** Módulo responsável por todas as operações de arquivos

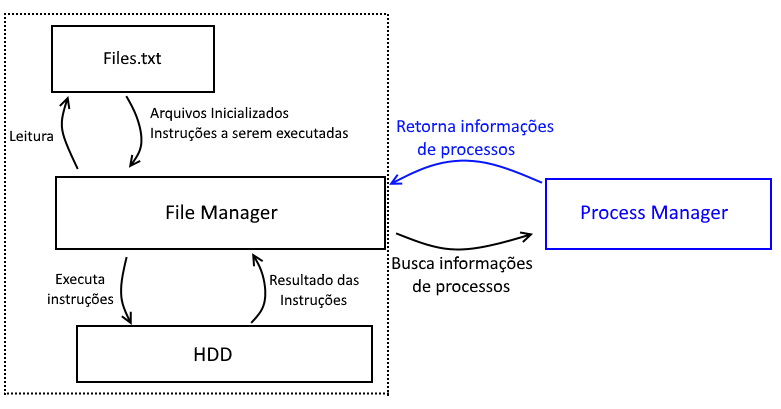
Antes de entrarmos nos detalhes da solução em si, vamos dar uma olhada na arquitetura utilizada para a resolução do problema:

**Arquitetura do Process Manager**



* **Process Manager:**
  + **Função:** Recebe como entrada um arquivo contendo informações dos processos e realiza uma simulação da execução de tais processos até que todos os processos especificados no arquivo tenham executado por completo. Fornece também uma interface de comunicação com módulos externos que precisam buscar informações sobre os processos.
  + **Equivale aos seguintes módulos da especificação:**
    - **Módulo de Processos** – Contém as estruturas de dados utilizadas para abstrair o funcionamento de um processo.
    - **Módulo de Leitura (módulo extra) –** Faz a leitura do arquivo de processos, validando-o e convertendo as linhas para as estruturas de dados definidas no módulo de processos.
* **Process Scheduler:**
  + **Função:** É o “Coração” do programa. Responsável por montar a abstração das estruturas de fila e escalonar quais processos devem ser executados em determinado momento, levando em conta as prioridades de cada processo e os recursos disponíveis para uso. [Pietro: Explicar algoritmo utilizado para evitar starvation]
  + **Equivale ao módulo de filas da especificação**
* **Gerenciador de Memória**
  + **Função:** Responsável por verificar a disponibilidade de memória, tanto para processos de tempo real quanto para demais processos, alocar blocos contíguos de memória utilizando o algoritmo First-Fit e liberar a memória quando um processo encerrar.
  + **Equivale ao módulo de memória da especificação.**
* **Gerenciador de Dispositivos**
  + **Função:** Responsável por gerenciar todos os dispositivos de E/S, realizando alocação e liberação desses recursos.
  + **Equivale ao módulo de recurso da especificação.**
  + **Obs:** No algoritmo construído, tanto memória quanto dispositivos foram considerados como **recursos do sistema**, mas isso somente no âmbito da nomenclatura. O funcionamento de cada módulo **permanece em conformidade com a especificação.**

**Arquitetura do File Manager**

****

* **File Manager**
  + **Função:** Recebe como entrada um arquivo contendo o estado inicial do disco e as instruções à serem executadas pelo sistema de arquivos e inicializa o disco com tais dados, para logo em seguida realizar a execução das instruções. O módulo também é responsável por verificar erros na sintaxe definida pela especificação, verificar erros lógicos nos arquivos de inicialização (como por exemplo inicializar 2 arquivos que ocupam os mesmos setores) e buscar dados dos processos realizando operações no disco no Gerenciador de Processos por meio de uma interface de comunicação.
  + **Equivale aos seguintes módulos da especificação:**
    - **Módulo de Arquivos –** Responsável por operações de Create e Delete
* **HDD**
  + **Função:** Abstrair as funcionalidades de um disco rígido conforme a especificação

1. **Principais Dificuldades e suas soluções**
2. **Comunicação entre módulos trabalhando em paralelo**

Um dos principais pilares visando uma boa qualidade do código desenvolvido e um desenvolvimento ágil e paralelo foi conseguir dividir o problema em módulos bem definidos com responsabilidades claras, conceito conhecido como *Separation of Concerns*. Como muitos dos módulos dependiam dos outros, a solução foi estabelecer uma interface de comunicação com os contratos das funções que seriam usadas para ligar os módulos. Assim não era necessária uma implementação concreta de um módulo para que o outro pudesse funcionar, bastava criar uma implementação provisória que seria substituída pelo módulo de fato assim que fosse concluído.

1. **Problemas de refatoração de código**

Durante o desenvolvimento do programa, muitas vezes percebia-se que o código poderia ser feito de uma forma mais genérica para melhorar a reutilização de código. Porém, fazer mudanças num código que já funcionava gerava uma incerteza, uma vez que seria necessário revalidar toda a implementação já feita a fins de conferir que o código refatorado atende a todos os requisitos que o código antigo atendia. Para facilitar esse processo, foram desenvolvidos testes de unidade automatizados, que validam automaticamente os requisitos estabelecidos.

1. **Testes de unidades**

Testes de unidade tem como característica principal a checagem de requisitos estabelecida em um módulo. Porém, como os módulos do sistema desenvolvidos estão interligados, foi necessário o desenvolvimento de implementações de teste que comunicavam com os módulos testados por meio da interface estabelecida. Desta forma, o desenvolvimento de um módulo X não impactava nos resultados do teste de um módulo Y.

1. **Erros de validações de arquivo**

Foi necessário, antes de fazer as devidas conversões, validar o input do usuário, interrompendo o programa e exibindo detalhadamente qual foi o erro encontrado, para que o usuário revalidasse o arquivo antes que entrasse com ele novamente no sistema.

1. **Função dos Integrantes**

**[Decisao em grupo]**

1. **Bibliografia**

**[Pietro]**