**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**

* **C#:** é uma linguagem de tipos protegidos, orientada a objeto e que permite aos desenvolvedores construírem uma variedade de aplicações compatíveis com o .NET Framework. [1]
* **.Net Core:** é um conjunto de ferramentas criado pela Microsoft que permite uma “comunicação” entre várias linguagens, graças a um poderoso conjunto de bibliotecas que fazem com que os programas .NET rodem em um ambiente chamado CLR (Common Language Runtime), ao invés de rodar diretamente no hardware, como em C, C++ ou Assembly. [1]
* **MsTest:** possui a finalidade de verificar pequenas porções de código como classes e métodos. Testes unitários realizam a validação das estruturas que compõem um software, procurando garantir que as estruturas criadas estão em conformidade com aquilo que foi desejado pelo programador. Um dos benefícios diretos da adoção de tal prática, está na obtenção de sistemas mais confiáveis e a capacidade de se verificar com maior velocidade os impactos decorrentes de mudanças. [2]
* **Visual Studio 2017:** É a edição mais recente do ambiente de desenvolvimento integrado da Microsoft especialmente dedicado ao .NET, capaz de oferecer ao desenvolvedor suporte completo para a criação de software para Windows, Mac, Linux, além de estrutura para desenvolvimento web e de aplicativos para Android e iOS. [3]
* **Trello:** É um sistema de quadro virtual para gerenciamento de tarefas que segue o método "kanban", muito usado no desenvolvimento com Scrum. Ele permite a criação de diversos quadros, nos quais podemos criar quantas colunas quisermos. Dentro de cada coluna é possível adicionar um ou mais "cards" (que, propriamente ditas, são as tarefas), as quais contêm o que o usuário desejar. [4]
* **GitHub:** é um site que permite a hospedagem de códigos fontes e sistema de gerenciamento de versões dos mesmos por meio do Git. Através deles podemos desenvolver projetos onde diversas pessoas podem contribuir simultaneamente. Os contribuintes podem criar novos arquivos e editar arquivos já existentes sem o risco de suas alterações serem sobrescritas indevidamente. [5]
* **SourceTree:** é um programa que fornece uma interface visual entre o cliente e o GitHub. Assim, o desenvolvedor pode realizar operações do Git sem a necessidade direta de linhas de comando e obter uma visualização da fluidez do código.

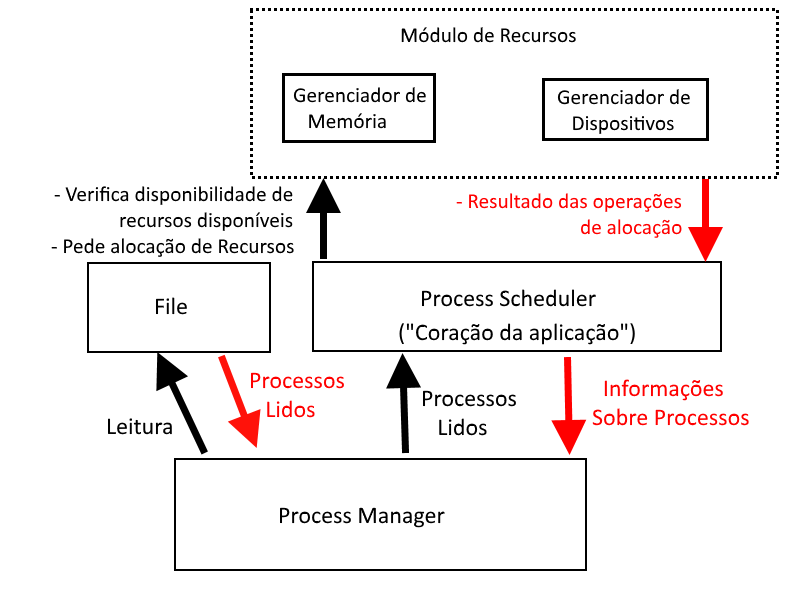
1. **Descrição Teórica e Prática da solução dada**

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.

A solução desenvolvida para o Process Manager segue a seguinte arquitetura:

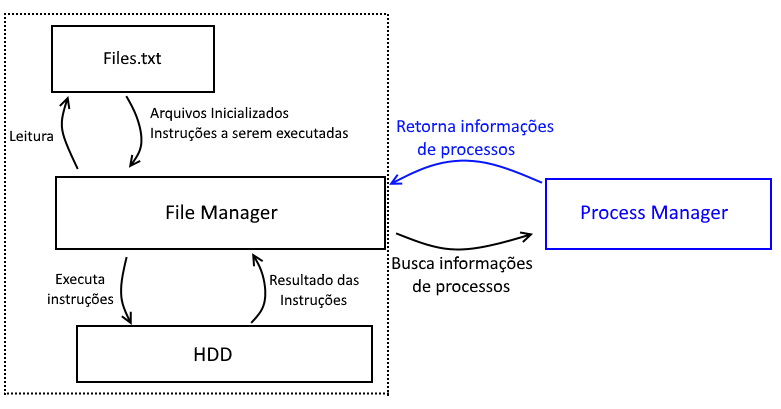
**Arquitetura do Process Manager**



* **Process Manager:**
  + **Função:** recebe como entrada um arquivo contendo informações dos processos e realiza uma simulação da execução dos mesmos até que todos os processos especificados no arquivo tenham sido executados 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) –** realiza 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:** funciona como o “coração” do programa. É 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. Para evitar *starvation* de algum processo, o simulador do sistema operacional utiliza uma técnica de envelhecimento (aging), em que processos bloqueados que já rodaram em algum momento e foram bloqueados por outro de maior prioridade, começam a ser atribuídos uma prioridade de menor número a cada segundo (o que corresponde a 1 batida de *clock*), quanto menor o valor de prioridade, maior é a prioridade de execução daquele processo. O valor máximo de prioridade que os processos podem chegar é 1, pois processos de prioridade 0 são considerados de tempo real.
  + **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 os 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 a memória quanto os dispositivos foram considerados como **recursos do sistema**, mas isso apenas no âmbito da nomenclatura. O funcionamento de cada módulo **permanece em conformidade com a especificação.**

Por sua vez, a solução desenvolvida para o File Manager segue a seguinte arquitetura:

**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. O disco é inicializado com tais dados, para que logo em seguida realize a execução das instruções.

O módulo também é responsável pela verificação de erros na sintaxe (definidas pela especificação), verificação de erros lógicos nos arquivos de inicialização (como a inicialização de dois arquivos que ocupam os mesmos setores) e a realização de operações no disco (por meio da busca de dados dos processos e subsequente verificação de permissões para tais operações por meio de uma interface de comunicação com o Gerenciador de Processos).

* + **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**

Conseguir dividir o problema em módulos bem definidos com responsabilidades claras, conceito conhecido como *Separation of Concerns)* é um dos principais pilares que visam uma boa qualidade de código, desenvolvimento ágil e desenvolvimento paralelo. Como muitos dos módulos dependiam de outros, a solução foi estabelecer uma interface de comunicação com os pontos de contato das funções que seriam usadas para ligar os módulos. Logo, não houve a necessidade da implementação concreta de um módulo para que o outro funcionasse, bastava-se criar uma implementação provisória que seria substituída pelo módulo de fato assim que o seu desenvolvimento 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, realizar mudanças em um código já funcional gerava incertezas, uma vez que seria necessário revalidar toda a implementação já feita para 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 estabelecidos em um módulo. No entanto, como os módulos do sistema desenvolvidos estão interligados, foi necessário o desenvolvimento de implementações de teste que realizassem a comunicação 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 efetuar uma validação do input do usuário, antes das devidas conversões serem realizadas. Assim, o programa era interrompido e exibia detalhadamente qual foi o erro encontrado, para que o usuário realizasse uma nova validação do arquivo antes que o enviasse novamente para o sistema.

1. **Função dos Integrantes**

* Arquitetura do codigo: Pedro Yan
* Desenvolvimento do código: Todos
* Testes automatizados: Pietro Bertarini
* Correção de bugs: Vitor Satoru

1. **Bibliografia**

**[1]** [**www.programacaoprogressiva.net/2012/08/comece-programar-linguagem-de\_31.html**](http://www.programacaoprogressiva.net/2012/08/comece-programar-linguagem-de_31.html)

**[2] [https://www.mundotibrasil.com.br/testes-unitarios-visual-studio-2015/#more-5773](https://www.mundotibrasil.com.br/testes-unitarios-visual-studio-2015/" \l "more-5773)**

**[3]** [**http://www.techtudo.com.br/tudo-sobre/visual-studio.html**](http://www.techtudo.com.br/tudo-sobre/visual-studio.html)

**[4]** [**https://canaltech.com.br/utilitarios/gerencie-equipes-e-tarefas-com-o-trello-e-de-adeus-aos-post-its/**](https://canaltech.com.br/utilitarios/gerencie-equipes-e-tarefas-com-o-trello-e-de-adeus-aos-post-its/)

**[5]** [**https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar/**](https://tableless.com.br/tudo-que-voce-queria-saber-sobre-git-e-github-mas-tinha-vergonha-de-perguntar/)