**Sistemas Operativos**

**Projeto - Shell Simples para o Linux**

Hugo Alexandre Silva

Beja, aos 30 de Janeiro de 2020

Índice

[Índice 2](#_Toc31310616)

[1. Introdução 3](#_Toc31310617)

[2. void GetCommand(char command[]) 4](#_Toc31310618)

[3. int PrintArguments(char command[]) 5](#_Toc31310619)

[4. makeArgVector(char command[], char \*argVector[]) 6](#_Toc31310620)

[5. execvp, ss - exeCommand 7](#_Toc31310621)

[6. funcionalidades avançadas na shell 7](#_Toc31310622)

[7. print demonstrando a execução e resultados 10](#_Toc31310623)

[8. Conclusões Finais 14](#_Toc31310624)

[9. Referências bibliográficas 15](#_Toc31310625)

Imagem1 – Imagem1 – código getcommand..........................................................................................4

[Imagem2 – código main..........................................................................................................................](#_Toc5608514)5

Imagem3 – código printarguments.........................................................................................................5

[Imagem4 – código makeargvector.........................................................................................................](#_Toc5608516)6

[Imagem5 – código execommand............................................................................................................](#_Toc5608517)7

[Imagem6 – código execução de listas de comandos............................................................................](#_Toc5608518). 9

# Introdução

Este projeto trabalha com desenvolvimento de funções para a Sistemas Operacionais onde nos foi dado como tarefa criar um conjunto de shell simples denominada ss (simple shell). Além disso as funções devem fornecer informações ao utilizador durente o processo. Utilizamos um terminal virtual com Linux Ubuntu. O ponto a cumprir, deve aceitar um comando introduzido pelo utilizador e devolvê-lo, através de parâmetros command; estes comandos devem ser armazenados em um vetor; construindo assim um vetor que contenha em cada posição um dos argumentos do comando e na posição seguinte ao último argumento deve ser colocada a constante null como indicação do fim dos argumentos estabelecidos; tendo em conta uma shell filho onde construímos um vetor de argumentos para executar o comando através da função execvp e deve-se esperar a conclusão da shell filho para então ficar disponível para a introdução de um outro comando.

Para o desenvolvimento do projeto, utilizamos a linguagem de programação C, uma linguagem de alto nível onde podemos trabalhar próximo da máquina, tendo disponível diversas características de portabilidade, compilação, eficiência e regularidade.

Para trabalhos com os códigos utilizamos o editor “gedit”, onde temos uma facilidade de percepção do código, uma vez que o mesmo identifica a linguagem, além das variáveis e funções, separando as mesmas por cores e sequência, o que ajuda nas identificações e facilidade de uso para o devido trabalho.

Para testar o funcionamento da shell ao longo do desenvolvimento, é necessário utilizar um compilador para traduzir linguagem de alto nivel em linguagem de máquina, no nosso caso utilzamos o compilador Gnu Compiler Collection, abreviado por G.C.C.

Utilizamos juntamente com esse desenvolvimento a ferramenta GitHub para compartilhar o desenvolvimento e partes de documentação encontradas por ambos componentes do grupo, ao que poderá ser encontrado no seguindo url: “https://github.com/MistahJorge/SO-TG2”, a qual o docente já se encontra com o convite enviado para se juntar ao compartilhamento e com isso ter acesso aos commits, comentários e participação do desenvolvimento.

Concluímos que se tudo ocorrer devidamente bem teremos sempre em ecrã as informações e ações executadas durante todo o processo, juntamente com os processos de excução da shell pai e filho.

# void GetCommand(char command[])

Nesta primeira etapa pretende-se criar um desenvolvimento onde essa função deverá escrever na ecrâ o comando inserido pelo utilizador dentro de um ciclo, até que o utilizador digite ‘quit’ para sair do programa.

No código utilizamos o strcmp para comparar a string comando, com “quit”, enquanto não for igual, dando resultado diferente de 0, continua o ciclo. “fgets” para receber a entrada de string a qual recebe os caracteres e armazena no buffer apontado para o “s”, até a interrupção após a EOF ou uma nova linha. Completamos a ação com o comando “strlen”, que calcula o comprimento da string, excluindo o byte de terminação nula.

Em relação as bibliotecas que estamos usando, são as suas características e ações conforme suas rotinas carregadas: “stdio”, funções de biblioteca para entrada e saída padrão, “string”, executa operações em cadeia terminado em nulos, “stdlib”, é um nível superior ao stdio, a qual faz parte da stdlib, controle de rotinas automáticas, “unistd.h”, composta por funções wrapper de chamadas do sistema, apoia ações do fork, pipe e primitivas de entrada e saída, “errno”, fornece [macros](https://pt.wikipedia.org/wiki/Macro) para identificar e relatar erros de execução através de [códigos de erro](https://pt.wikipedia.org/w/index.php?title=Códigos_de_erro&action=edit&redlink=1), “sys/wait”, emite chamadas de espera sistema, que suspende a execução de determinados processos, até o status de saída para o sistema operacional e liberação para outro processo, utilizado para controlo de processos entre pai e filho.

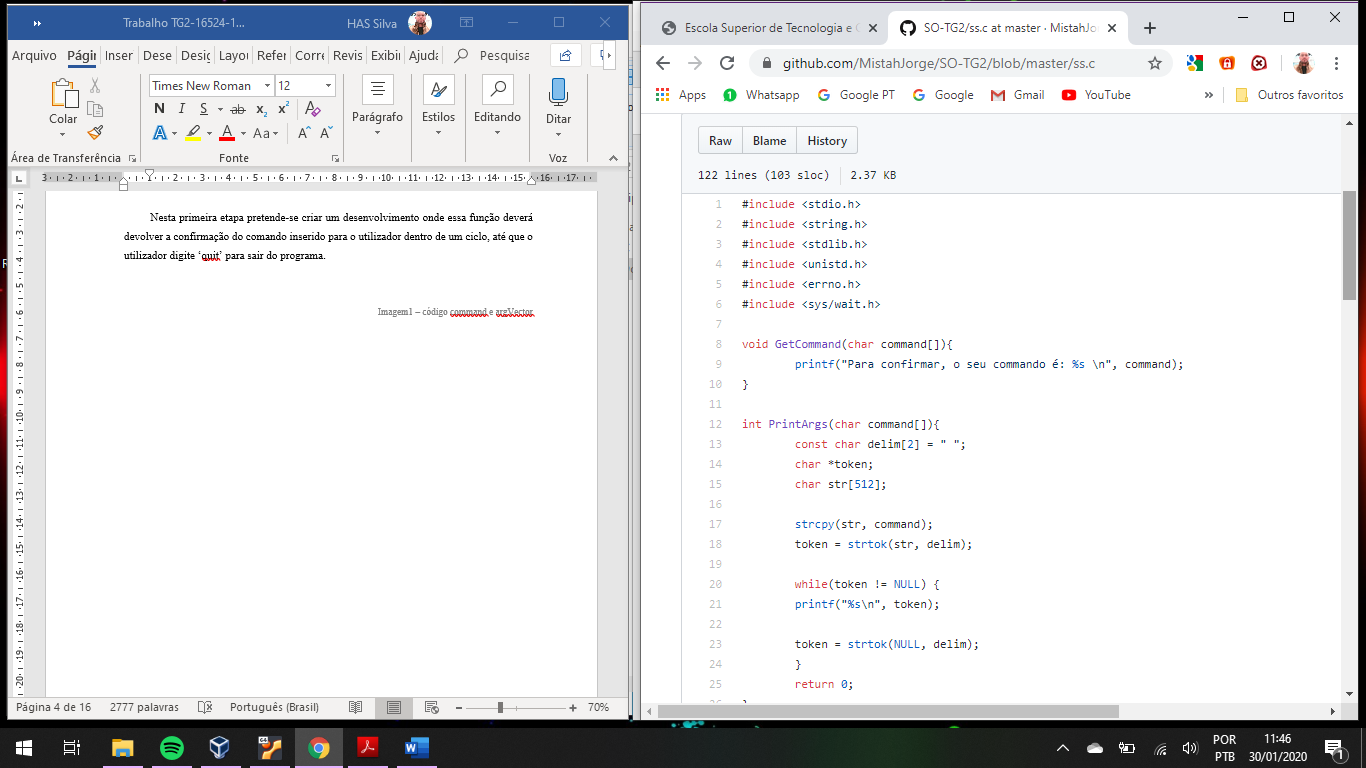


Imagem1 – código getcommand

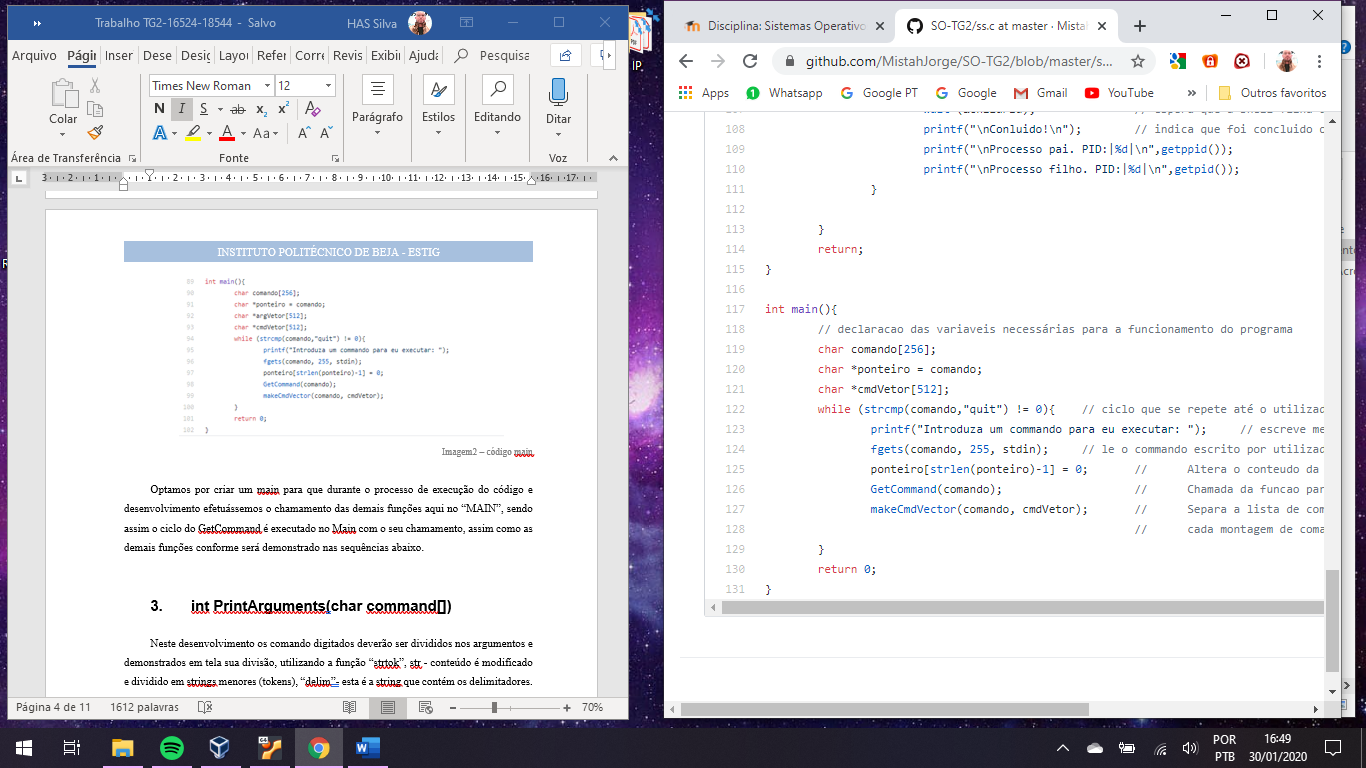


Imagem2 – código main

Optamos por criar um main para que durante o processo de execução do código e desenvolvimento efetuássemos o chamamento das demais funções aqui no “MAIN”, sendo assim o ciclo do GetCommand é executado no Main com o seu chamamento, assim como as demais funções conforme será demonstrado nas sequências abaixo.

# int PrintArguments(char command[])

Neste desenvolvimento os comando digitados deverão ser divididos nos argumentos e demonstrados em tela sua divisão, utilizando a função “strtok”, str - conteúdo é modificado e dividido em strings menores (tokens), “delim”- esta é a string que contém os delimitadores.

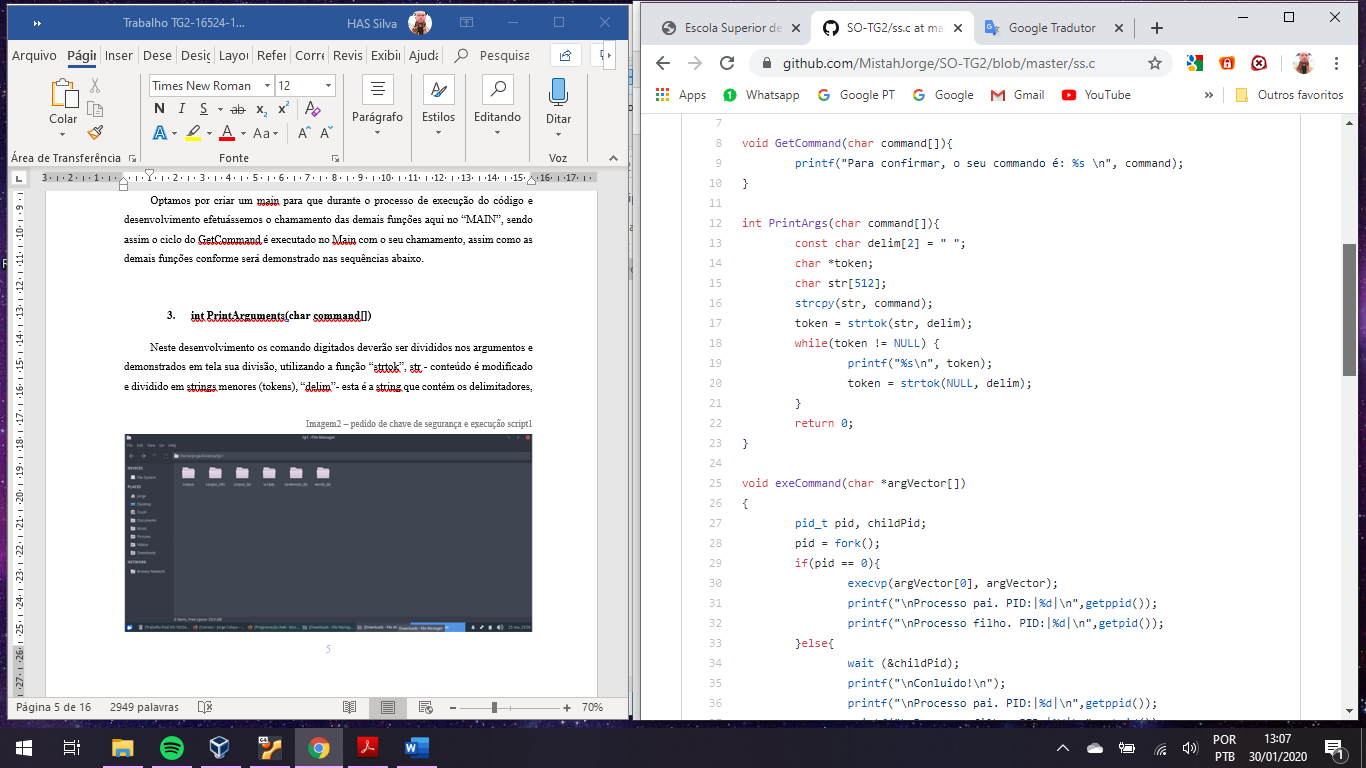


Imagem3 – código printarguments

# makeArgVector(char command[], char \*argVector[])

Para o desenvolvimento destas função foi solicitado para que a mesma execute a construção de um vetor que contenha os argumentos em suas posições, ao preencher todos as posições o ultimo deve conter a constante NULL para indicar o fim, deve ser usado a função “malloc”, ou seja, alocação dinâmica de memória, aloca espaço para um bloco de bytes consecutivos na memoria RAM e devolve o endereço desse determinado bloco. Por isso a utilização do ponteiro e void, pois é uma função do tipo genérico e armazenamos o endereço num ponteiro do tipo apropriado, no nosso caso “char”.

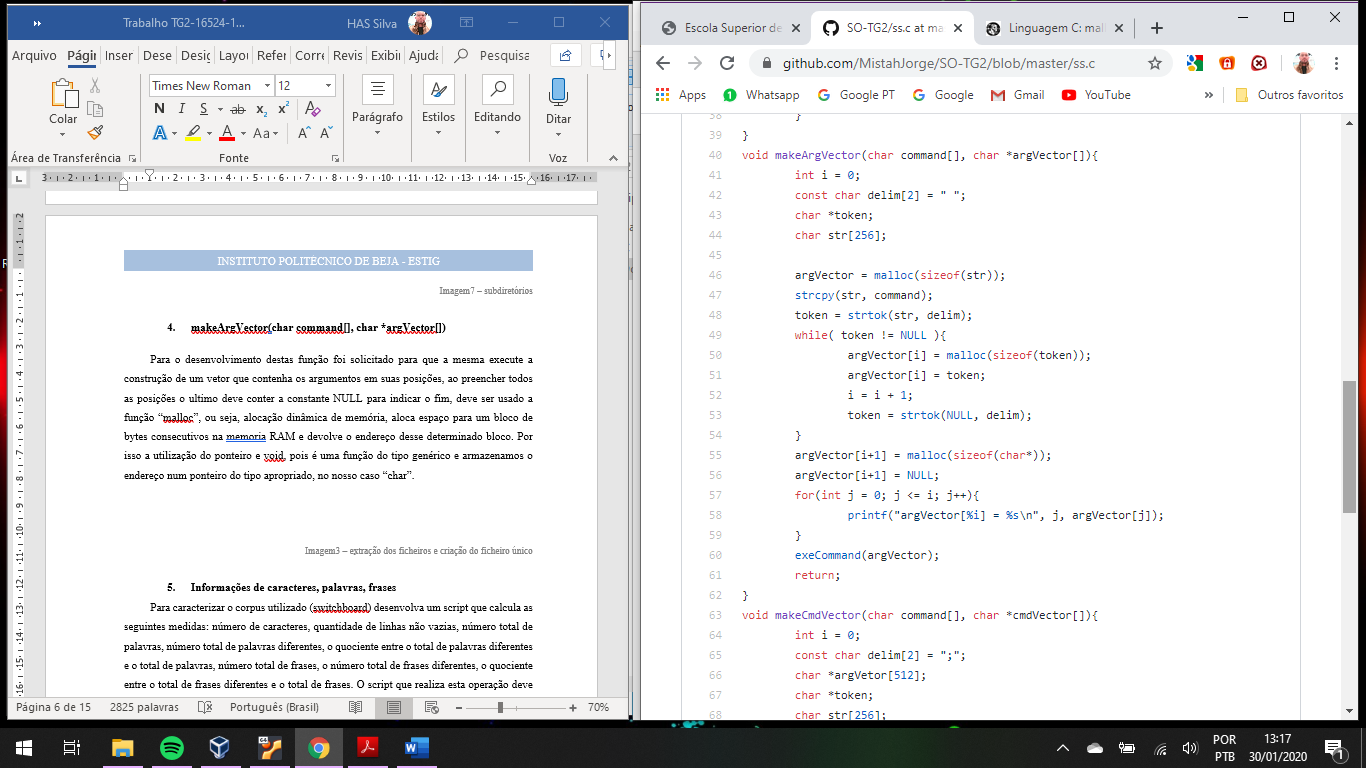


Imagem4 – código makeargvector

# execvp, ss - exeCommand

Nesta etapa cinco concluímos a primeira parte do trabalho com o caso para desenvolver a versão da “ss”, para que o comando recebido pela shell deva lançar uma shell filha e constuir um vetor de argumentos com suas devidas execuções através da função “execvp”, que é da família execução, cuja ação é substituir o processo atual por uma nova imagem do processo, onde no caso do “vp”, fornece uma matriz ou vetor com ponteiros para sequenciar as ações e terminar em nulo, que representa o fim da lista de argumento para disposição de um novo programa.

Para isso criamos uma função “exeCommand()” que recebe os valores de argVector e temos o chamamento do pai e filho e as diretrizes.

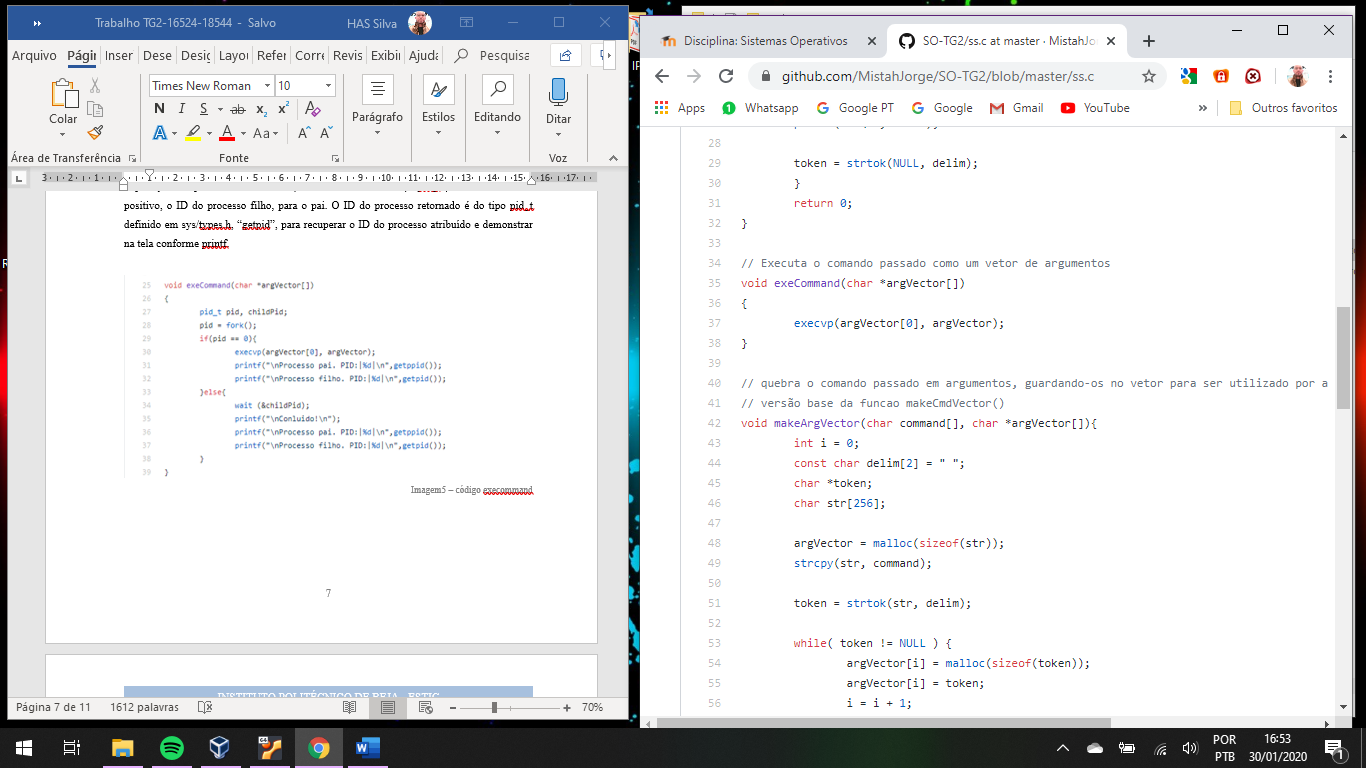


Imagem5 – código execommand

# funcionalidades avançadas na shell

Nesta etapa iniciamos a parte dois do projeto, onde devemos implementar usos de possibilidades de execução com determinadas funcionalidades dispostas em projeto e solicitada pelo professor.

“Fork”, cria um processo através da duplicação do processo de chamada, referido como o filho, “pid\_t”, retorna um valor positivo, o ID do processo filho, para o pai. O ID do processo retornado é do tipo pid\_t definido em sys/types.h, “getpid”, para recuperar o ID do processo atribuído e demonstrar na tela conforme printf.

Exemplos: implementar a possibilidade de execução de listas de comandos, por exemplo: sleep 5; date; sleep5; date

implementar a possibilidade de utilização da redireção, por exemplo: ls -l > file\_list.txt

implementar a possibilidade de utilização de pipelines, por exemplo: ls -l | grep ‘^d’ | wc -l

Aqui temos disponível o código para a primeira tarefa, sequencial ao “;”, onde será o nosso ponto de separação, para execução dos comandos, armazenamento dos argumentos. Usamos como base de processo o código da primeira parte onde, usamos o “strtok” para a dividir as string, mas antes com o “cpy” copiamos a string original para efetuarmos os trabalhos numa string nova, o “delim” nesse caso continua recebendo o valor [2], devido a receber a identificação da string “;”, que corresponde a 1(um) e “/0”, que corresponde a 2(dois), que significa o fim da string e deve efetuar a separação, conduzindo para as demais operações de argumentos, comandos e execução separadas.

Com a criação da nova função makecmdvector, buscamos aperfeiçoar na função a utilização de quase todas as funções solicitadas durante o trabalho, para assim garantirmos resultados perfeitos, usando além das funções acima já citadas, também fazendo recurso do “malloc” e utilização de um contador para as posições no vetor, e mais dois “for”, um para impressão dos resultados e acompanhamento em tela dos argumentos atribuídos e o segundo para chamar as demais funções para o ciclo de funcionamento conforme ainda existe processos ou aguardo de processos para serem executados.

A saber a execução visada para o projeto é em cascata, onde uma função recebe da outra função os comandos, efetua as suas atribuições de argumento separadas e efetuando suas comunicações e chamamento pelo main, até que o utilizador solicite a sua interrupção ou saída do programa.

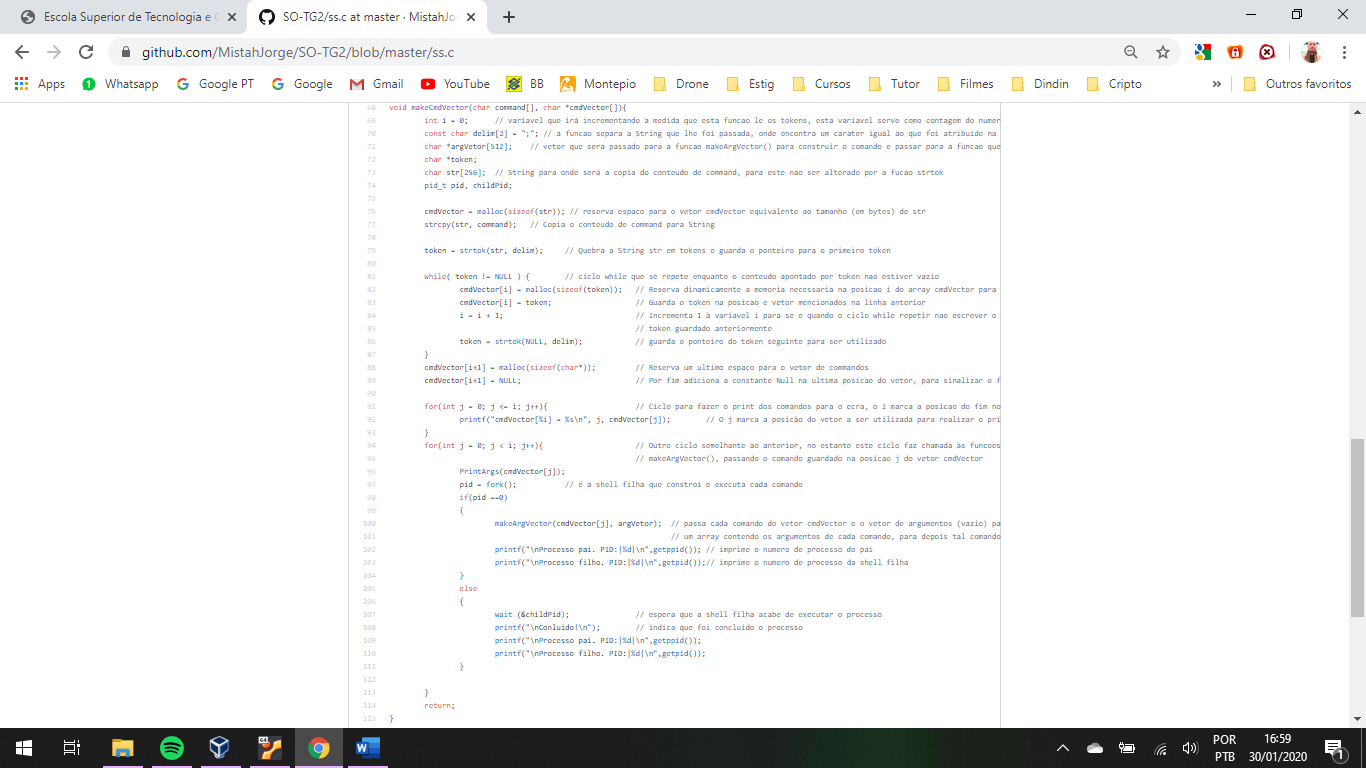
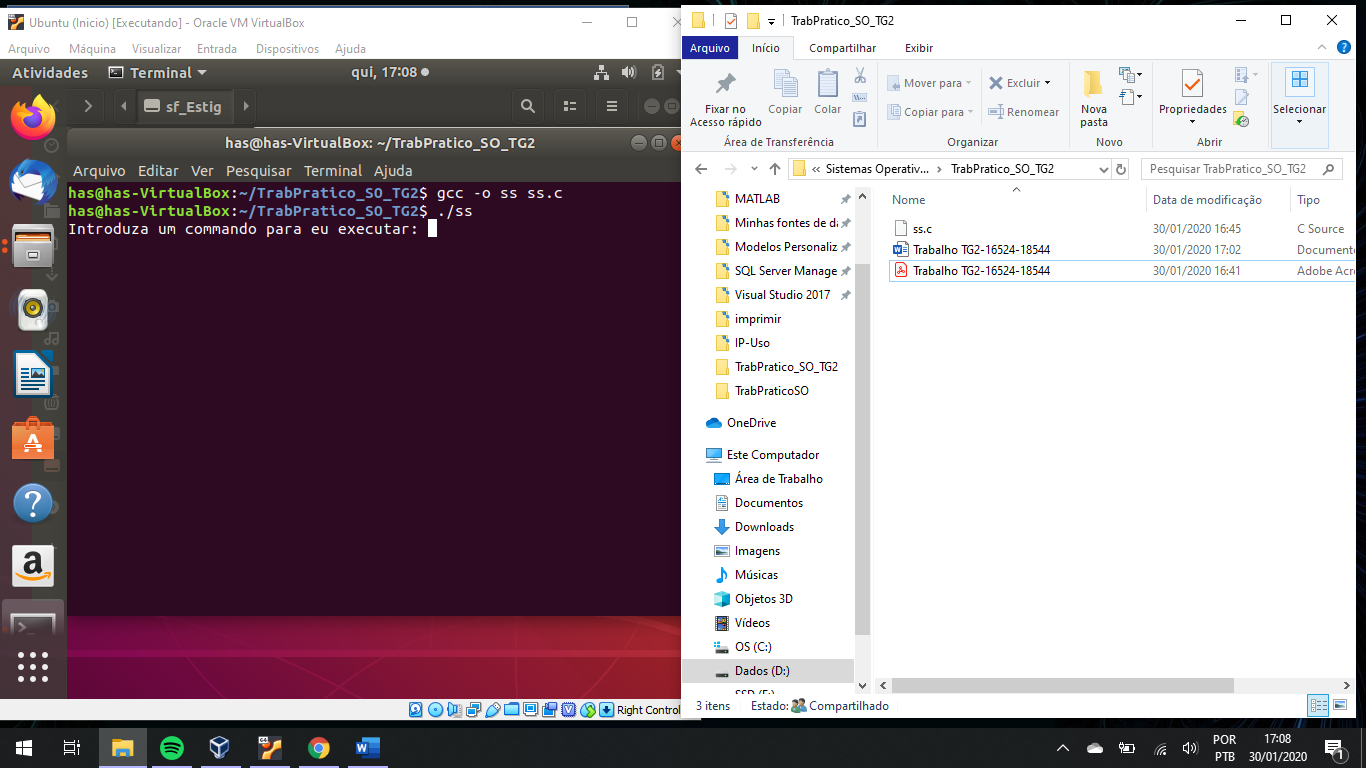
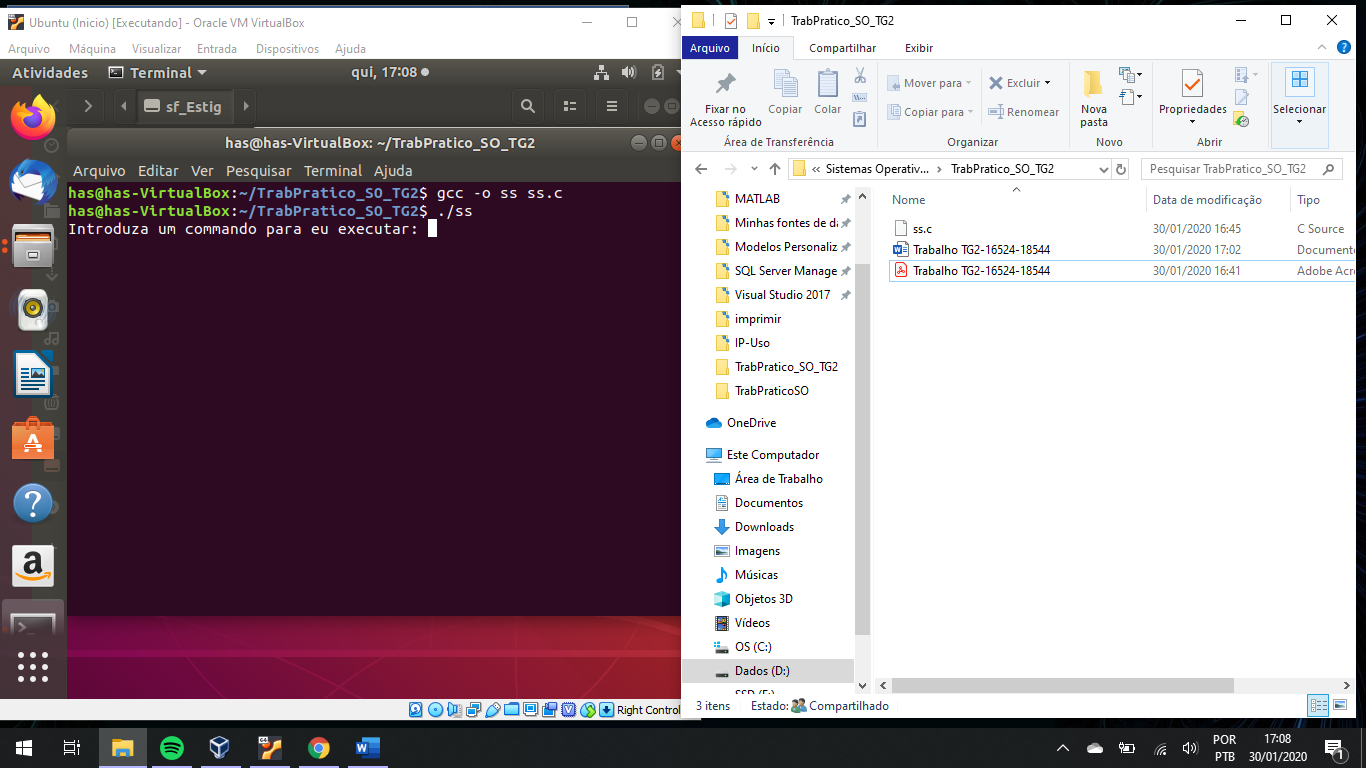
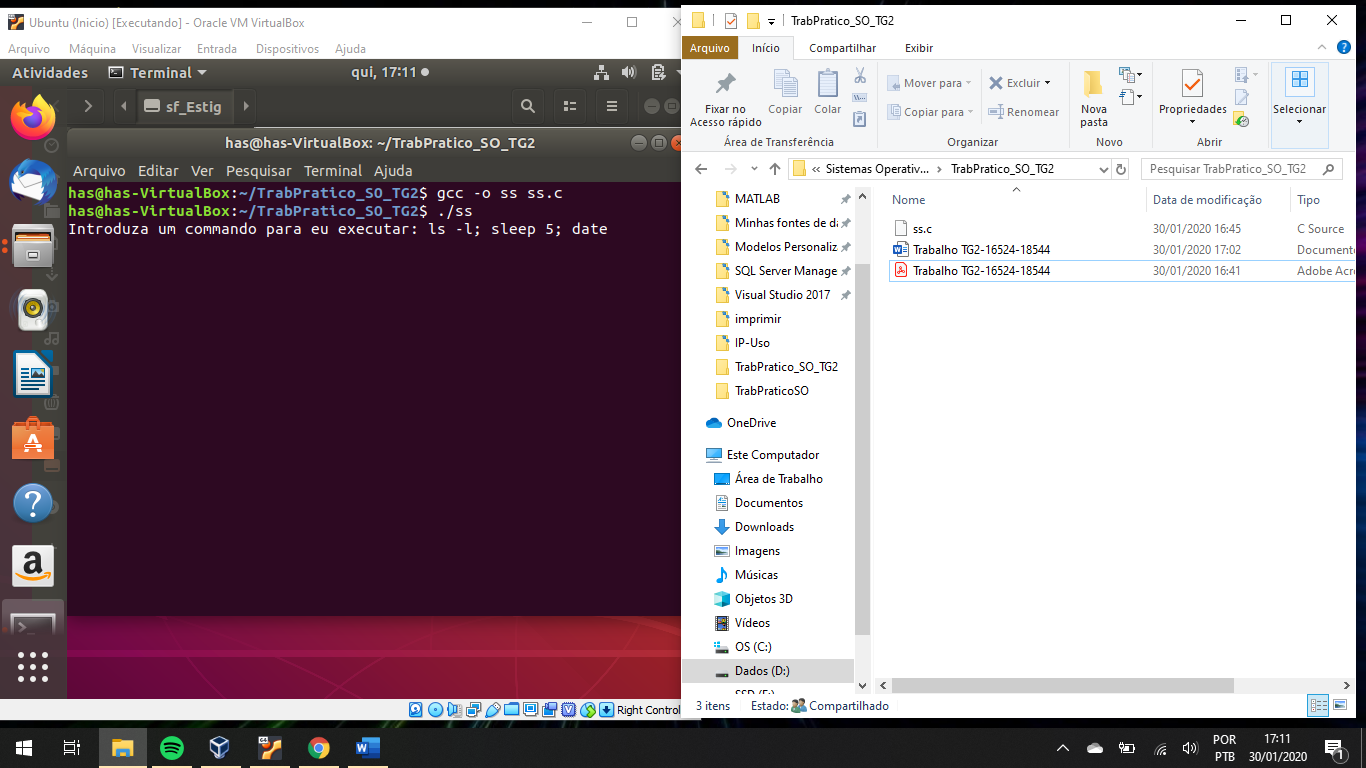


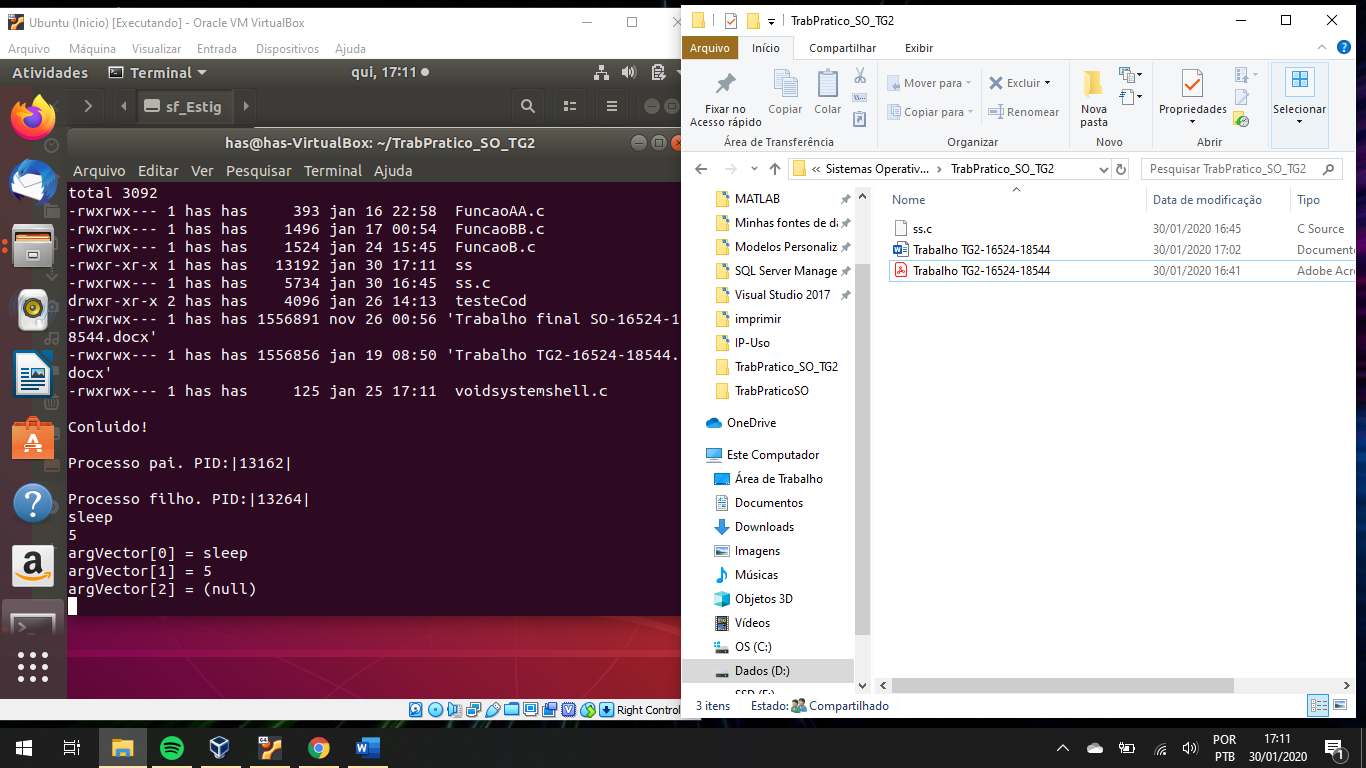
Imagem6 – código execução de listas de comandos

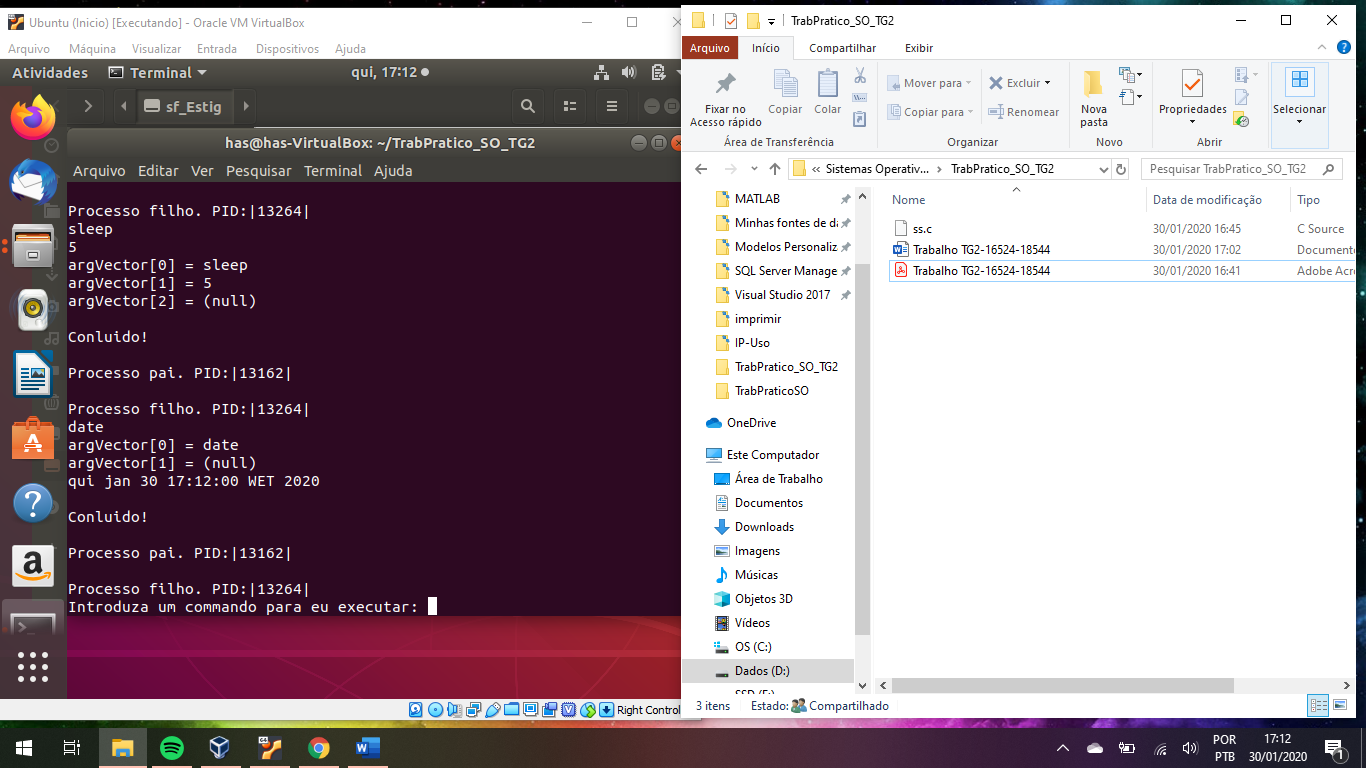
# print demonstrando a execução e resultados

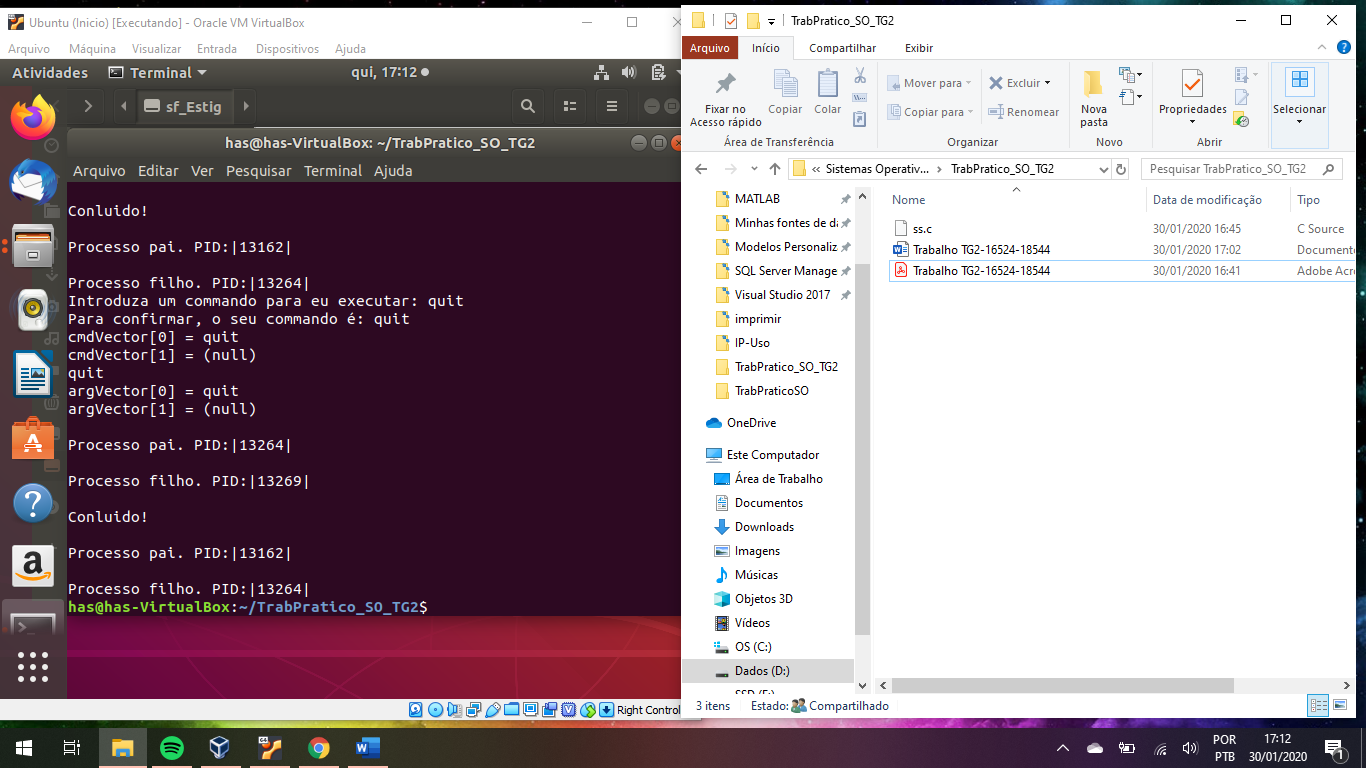






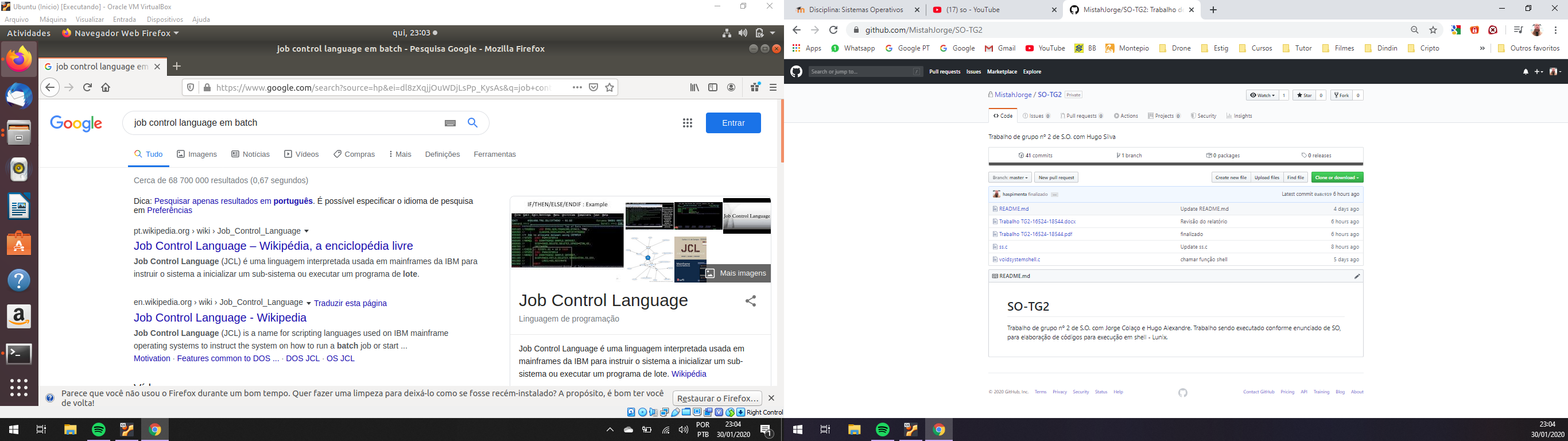


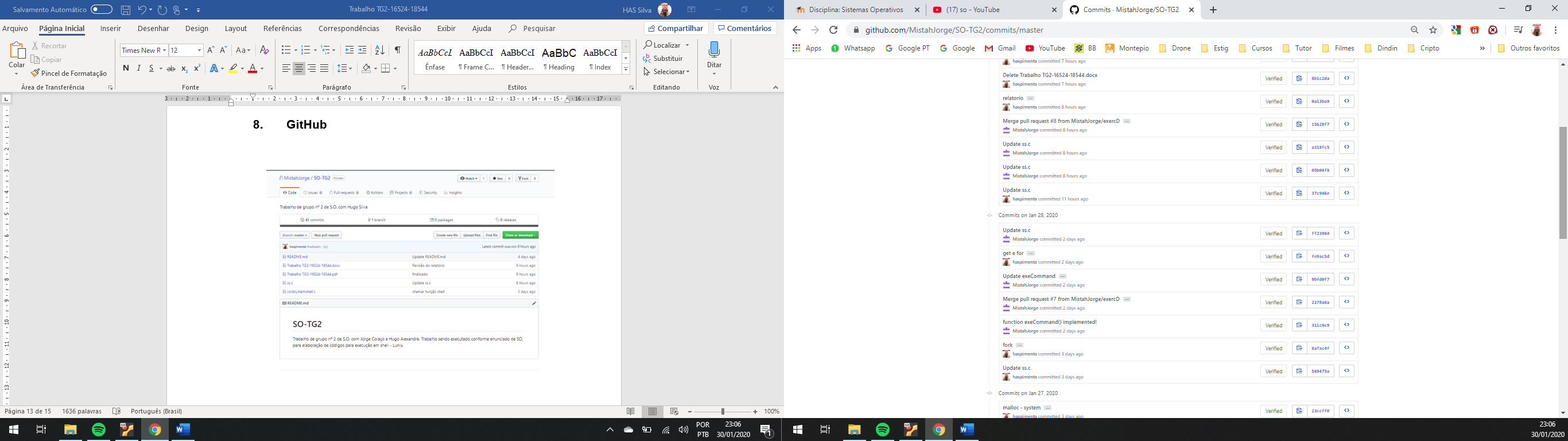




# GitHub

O github, foi utilizado conforme solicitação do projeto para demonstrar os commits realizados pelos colaboradores. Utilizamos o github, para armazenar todas as alterações do código, incluindo as positivas, funcionais, como as negativas, “não funcionais”, adicionando também pesquisa de trechos e códigos de uso e consulta, demonstrando do início ao fim toda a pesquisa, realização e modificações do código, relatório e projeto num todo, usando os commits.





# Conclusões Finais

O trabalho foi apresentado e elaborado dentro da proposta de vários funções para ação e execução das tarefas baseadas a partir de um desenvolvimento para shell do linux.

A elaboração de criação do programa demanda principalmente da performance do sistema e utilização das funções solicitadas em cada tarefa, tendo em consideração os comandos digitado pelo utilizador e permissões de acesso e escrita, adaptando e eliminando falhas para não prejudicar a execução das tarefas.

Baseado no proposto, podemos dizer que este trabalho foi um bom começo para o aprendizado sobre como trabalhar com shell, comandos, funções, segurança no fornecimento do processo, o que nos incentivou este desenvolvimento sequencial foi a possibilidade de ser aplicado em algo real e que pode ser utilizado para outros modelos e parâmetros.

Concluindo este projeto podemos dizer que a necessidade de um processo com respostas de confirmação em cada etapa das funções foi necessária para garantir a integridade e os utilizadores poderem ver cada etapa e as funções executadas, conforme código.

Estamos satisfeitos com os resultados e conhecimento adquirido.

# Referências bibliográficas

Moodle

https://cms.ipbeja.pt/pluginfile.php/227883/mod\_resource/content/1/tarefa4.pdf

https://cms.ipbeja.pt/pluginfile.php/228485/mod\_resource/content/1/Introdu%C3%A7%C3%A3o%20%C3%A0%20Gest%C3%A3o%20de%20Processos.pdf

https://cms.ipbeja.pt/pluginfile.php/229019/mod\_resource/content/1/Escalonamento%20Processos.pdf

https://cms.ipbeja.pt/pluginfile.php/229766/mod\_resource/content/1/Comunica%C3%A7%C3%A3o%20e%20Sincroniza%C3%A7%C3%A3o%20Processos.pdf

Youtube

https://www.youtube.com/watch?v=tF0Qau7zcsw&list=PLS1QulWo1RIYmaxcEqw5JhK3b-6rgdWO\_&index=27

https://www.youtube.com/watch?v=EOLPUc6oo-w&list=PLucm8g\_ezqNrYgjXC8\_CgbvHbvI7dDfhs

Tutoriais

https://www.ime.usp.br/~pf/algoritmos/aulas/aloca.html

http://www.br-c.org/doku.php?id=fork

https://linux.die.net

https://terminalroot.com.br/2015/07/30-exemplos-do-comando-sed-com-regex.html

https://www.livrosdelinux.com.br/respostas-livro-shell-linux/