



UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

ARTHUR HENRIQUE BANDO UEDA RA:129406

GUSTAVO GLATZ ALVES RA:128592

DIOGO FELIPE SOARES DA SILVA RA:124771

Controle de Estoque de Supermercado em Assembly PARTE 1

Maringá

2025

Introdução

O presente trabalho tem como objetivo desenvolver um CRUD de um sistema de controle de estoque de um Supermercado. Esse sistema será responsável por executar operações básicas sobre os produtos existentes nesse Supermercado. As operações que foram implementadas do trabalho foram as seguintes:

- Inserção de produto
- Remoção de produto
- Atualização de produto
- Consulta de produto
- Consulta financeira
- Gravação de registros
- Recuperação de registros

A linguagem que foi utilizada para a realização deste trabalho foi Assembly GAS com sintaxe AT&T. É uma ferramenta poderosa para programação de baixo nível, otimização e desenvolvimento de sistemas mais robustos. A ordem de dos operandos distingue-se de outras sintaxes, como a da Intel, pois é da forma: “*origem, destino*”. As instruções carregam um sufixo que corresponde ao tamanho do operando, como, por exemplo, ***movl*** para 32 bits. No contexto deste trabalho, a totalidade das operações será de 32 bits.

A seguir será documentado as principais funções implementadas no trabalho que satisfazem as operações requisitadas que foram listadas acima.

Primeiramente, aqui estão os comandos para executar o programa. Note que foi usado o ***gcc*** para fazer a linkagem do código objeto.

- 1) Compilação: ***as -32 -g trabalho.s -o trabalho.o***
- 2) Linkagem: ***gcc -m32 trabalho.o -g***
- 3) Execução: ***./a.out***

Quando o código é executado, aparecerá no terminal um menu de opções para o usuário. O usuário deve selecionar as opções conforme a necessidade:

```
=====
=== CONTROLE DE ESTOQUE ===
=====
1. Inserir Novo Produto
2. Remover Produto
3. Atualizar Produto
4. Consultar Produto
5. Consulta Financeira
6. Gerar Relatório
7. Gravar Registros em Arquivo
8. Carregar Registros do Arquivo
0. Sair do Programa
-----
Escolha uma opção: █
```

A parte do código que é responsável por mostrar o menu inicial e verificar a resposta do usuário está presente em **main**:

```
main:

    # head = NULL
    movl    $0, head_ptr

    start_point:
        call    read_menu

        movl    option, %eax
        cmpl    $1, %eax
        je      case1
        cmpl    $2, %eax
        je      case2
        cmpl    $3, %eax
        je      case3
        cmpl    $4, %eax
        je      case4
        cmpl    $5, %eax
        je      case5
        cmpl    $6, %eax
        je      case6
        cmpl    $7, %eax
```

```

je      case7
cmpl    $8, %eax
je      case8
cmpl    $0, %eax
je      exit

# invalid option
pushl    $msg_error_invalid_option
call     printf
addl     $4, %esp
jmp      start_point

```

Como é possível observar, a função ***read_menu*** é responsável por mostrar o menu inicial para o usuário e ler a opção que ele digitou e guardar na variável ***option***. Depois, o programa compara o valor em ***option*** com os possíveis casos existentes de acordo com o menu inicial (sendo eles de 1 até 8 e 0 para encerrar o programa). Se o usuário digitar um valor de número inválido, uma mensagem de erro será exibida e o menu inicial aparecerá novamente. Se o valor digitado não for um número (caractere alfabético), a mensagem de erro aparecerá normalmente.

A variável ***head_ptr*** que aparece no início da ***main*** representa a cabeça da lista ligada que será criada com a inserção de produtos no estoque. Por isso, ela deve ser inicializada com o valor 0 (NULL) no início do programa.

Vamos explicar a implementação de cada um dos possíveis casos de escolha pelo usuário a seguir.

Caso 1 - Inserção de produtos

Quando o valor de *option* é 1, será chamada a função ***insert_product***:

```

case1:
    call    insert_product
    jmp     start_point

```

Essa função é responsável por inserir um produto na lista ligada. Para isso, é necessário fazer 4 operações principais, como é possível ver no trecho de código abaixo:

```

insert_product:
    call    create_node
    call    input_product_data
    call    fill_node_with_data
    call    insert_node_by_name

    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    ret

```

create_node: Função responsável por criar o nó. Ou seja, alocar memória para esse nó e, também, para todas as variáveis que necessitam de alocação de memória. Para alocar os espaços necessários foi utilizado a função **malloc**.

input_product_data: Função que lê todos os atributos do produto por entrada de dados do usuário e armazena seus valores em variáveis globais definidas na **.section .data**.

fill_node_with_data: Função destinada a inserir os valores das variáveis da etapa anterior nas suas respectivas posições dentro do nó alocado. Isto é, vamos preencher o nó com os valores presentes nas variáveis.

insert_node_by_name: Função encarregada de inserir o nó que foi criado de forma ordenada (por nome do produto) na lista ligada. Essa ordenação acontece por meio do uso da função **strcmp** que compara o nome do produto novo, com os nomes dos produtos presentes na lista.

print_all_nodes_by_name: Função que printa a lista ligada por meio dos nomes dos produtos. Como é possível ver na imagem abaixo:

```

----LISTA ORDENADA----
Arroz Mineiro -> Batata Doce -> Canela em pó ->

```

Abaixo está um exemplo da execução de inserção de um produto na lista:

```
Escolha uma opção: 1

Digite o nome do produto: Batata Doce
Digite o lote do produto: 1

-----TIPOS DE PRODUDOS-----
1) Carnes
2) Laticínios
3) Hortifrúti
4) Bebidas
5) Padaria
6) Congelados
7) Limpeza
8) Higiene Pessoal
9) Mercearia
10) Pet Shop
11) Perfumaria
12) Eletrônicos
13) Brinquedos
14) Utilidades Domésticas
15) Vestuário

Digite o tipo do produto (1-15): 3
Digite a data de validade (DD/MM/AAAA): 10/07/2026
Digite o nome do fornecedor: Diogo Felipe
Digite a quantidade em estoque: 500
Digite o valor de compra (em reais): 1
Digite o valor de venda (em reais): 2

[SUCESSO] Produto inserido na lista.

-----LISTA ORDENADA-----
Batata Doce ->
```

Caso 2 - Remoção de Produtos

```
remove_product:
    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    pushl   $msg_remove
    call    enter_product_name
    addl    $4, %esp

    call    remove_product_by_name

    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    ret
```

A função acima remove um produto (nó) da lista ligada de acordo com o nome do produto passado pelo usuário. Essa ação é composta pelas seguintes funções:

enter_product_name: Função printa uma mensagem de remoção para o usuário e lê o nome do produto que deve ser removido. Esse nome é armazenado na variável ***product_name***.

```
enter_product_name:
    pushl   %ebp
    movl    %esp, %ebp

    movl    8(%ebp), %edx

    pushl   %edx
    pushl   $msg_crud_name
    call    printf
    addl    $8, %esp

    call    getchar
    pushl   stdin
    pushl   $30
    pushl   $product_name
    call    fgets
    addl    $12, %esp

    movl    $product_name, %eax
    pushl   %eax
    call    remove_newline
    addl    $4, %esp
```

```
popl    %ebp
ret
```

remove_product_by_name: Função incubida de comparar o nome em `product_name` com os nomes dos nós da lista ligada. Essa comparação é feita a cada nó até os valores dos nomes coincidirem ou a iteração chegar ao final da lista. Caso os valores coincidam, removemos o nó da lista ligada por meio da função ***free_node***. Essa função desaloca a memória inicialmente alocada com ***create_node***.

```
free_node:
    pushl    %ebp
    movl     %esp, %ebp

    movl     8(%ebp), %edx        # first parameter (current node)
    movl     %edx, restore_ptr

    # free name
    movl     NAME_REF(%edx), %ebx
    pushl    %ebx
    call     free
    addl     $4, %esp

    # free type
    movl     restore_ptr, %edx
    movl     TYPE_REF(%edx), %ebx
    pushl    %ebx
    call     free
    addl     $4, %esp

    # free expiration date
    movl     restore_ptr, %edx
    movl     EXPIRATION_REF(%edx), %ebx
    pushl    %ebx
    call     free
    addl     $4, %esp

    # free supplier
    movl     restore_ptr, %edx
    movl     SUPPLIER_REF(%edx), %ebx
    pushl    %ebx
    call     free
    addl     $4, %esp

    # free node
    movl     restore_ptr, %edx
    pushl    %edx
    call     free
    addl     $4, %esp

    popl     %ebp
    ret
```


Aqui está um exemplo da execução de remoção de um produto da lista:

```
=====
=== CONTROLE DE ESTOQUE ===
=====
1. Inserir Novo Produto
2. Remover Produto
3. Atualizar Produto
4. Consultar Produto
5. Consulta Financeira
6. Gerar Relatório
7. Gravar Registros em Arquivo
8. Carregar Registros do Arquivo
0. Sair do Programa
-----
Escolha uma opção: 2

----LISTA ORDENADA----
Batata Doce ->

Digite o nome do produto a ser removido: Batata Doce

[SUCESSO] Produto removido da lista.

----LISTA ORDENADA----
```

Caso 3 - Atualizar Produto

```
update_product:

    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    pushl   $msg_update
    call    enter_product_name
    addl    $4, %esp

    call    update_product_by_name

    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    ret
```

Essa função é muito parecida com a de remoção de produtos, visto que ela pede o nome do produto a ser atualizado. Depois acha esse produto, caso esteja presente na lista, e atualiza os campos **quantidade em estoque** e **valor de venda** que é feita pela função **update_product_by_name**.

Aqui está um exemplo de sua execução:

```
Escolha uma opção: 3

----LISTA ORDENADA----
Batata Doce ->

Digite o nome do produto a ser atualizado: Batata Doce

-----Produto Info-----

Nome do produto: Batata Doce
Quantidade em estoque: 500
Preço de venda: 2

Digite a nova quantidade em estoque: 300
Digite o valor de venda (em reais): 3

[SUCESSO] Produto atualizado.

-----Produto Info-----

Nome do produto: Batata Doce
Quantidade em estoque: 300
Preço de venda: 3
```

Caso 4 - Consultar produto

```
product_consult:
    movl    head_ptr, %eax
    pushl   %eax
    call    print_all_nodes_by_name
    addl    $4, %esp

    pushl   $msg_consult_product
    call    enter_product_name
    addl    $4, %esp

    call    product_consult_by_name

    ret
```

A consulta do produto é feita por meio de seu nome. Então, usa-se a função **enter_product_name** para ler o nome do produto e depois faz-se as comparações dos nomes dos produtos existentes na lista (função **product_consult_by_name**). Caso seja encontrado o produto, será printado todas as informações relacionadas àquele produto através da função **print_product2**.

```
print_product2:
# print all values of a especific node
    pushl   %ebp
    movl    %esp, %ebp

    movl    8(%ebp), %edi        # 1st parameter (current node)

    movl    $7, %ecx
start_print_loop:
    cmpl    $0, %ecx
    jl      end_print_loop
    movl    (%edi, %ecx, 4), %eax
    pushl   %eax
    decl    %ecx
    jmp     start_print_loop
end_print_loop:
    pushl   $product_data_fmt
    call    printf
    addl    $36, %esp
    popl    %ebp
    ret
```

Aqui está um exemplo de sua execução:

```
Escolha uma opção: 4

----LISTA ORDENADA----
Arroz Mineiro -> Batata Doce ->

Digite o nome do produto a ser consultado: Arroz Mineiro

-----Produto Info-----

Nome do produto: Arroz Mineiro
Lote: 2
Tipo: Mercearia
Data de validade: 20/12/2027
Fornecedor: Guilherme
Quantidade em estoque: 1000
Preço de compra: 3
Preço de venda: 8
```

Caso 5 - Consulta Financeira

Na consulta financeira o usuário vai selecionar uma das 3 opções de consultas possíveis, sendo elas: Total de compra, total de venda e lucro esperado (considera que todos os itens no estoque foram vendidos).

O total de compra e de venda é feito por meio de um loop que agrega os valores de compra e venda de cada produto na lista ligada nas variáveis ***total_purchase*** e ***total_sale***, respectivamente. Sendo que o cálculo de cada um desses valores é feito por:

$$\text{Total Compra/Venda} = \text{quantidade} * (\text{Valor Compra/Venda})$$

Depois que o loop termina, calcula-se o lucro esperado por meio da subtração de ***total_sale*** com ***total_purchase***. Assim, de acordo com a opção que o usuário escolheu, essa informação será mostrada para ele.

Aqui está o código do loop que faz as operações acima:

```
financial_consult_loop:
    # verify if the end of the list has been reached
    cmpl    $0, %edi
    je      end_finalcial_consult_loop

    # accumulate total_purchase: qnt*purchase_price + total_purchase
    movl    PURCHASE_REF(%edi), %ebx
    movl    QUANTITY_REF(%edi), %eax
    mull    %ebx
    movl    %eax, %ebx
    movl    total_purchase, %eax
    addl    %ebx, %eax
    movl    %eax, total_purchase

    # accumulate total_sale
    movl    SALE_REF(%edi), %ebx
    movl    QUANTITY_REF(%edi), %eax
    mull    %ebx
    movl    %eax, %ebx
    movl    total_sale, %eax
    addl    %ebx, %eax
    movl    %eax, total_sale

    # update current node
    movl    NEXT_REF(%edi), %edi
    jmp     financial_consult_loop
end_finalcial_consult_loop:
    # calculate total profit = total_sale - total_purchase
    movl    total_sale, %eax
    subl    total_purchase, %eax
    movl    %eax, total_profit
    # print the financial information according to the user option
    movl    option, %eax
    cmpl    $1, %eax
    je      print_total_purchase
    cmpl    $2, %eax
    je      print_total_sale
    cmpl    $3, %eax
    je      print_total_profit
    jmp     financial_consult
```

Caso 6 - Gerar Relatório

Esse caso é responsável por mostrar um relatório dos produtos presentes no estoque. O que ele faz é percorrer a lista ligada (que está já está ordenada por nome) e para cada nó, printa todos seus atributos de forma formatada em uma linha. Assim, ao final, teremos todos produtos em ordem alfabética listados no terminal. Abaixo está a implementação deste caso:

```

records_report:
    pushl    $msg_records_report
    call     printf
    addl     $4, %esp

    # loop through the linked list
    movl     head_ptr, %edi        # current node

start_report_loop:
    cmpl     $0, %edi
    je       end_report_loop

    # print the records
    pushl    SALE_REF(%edi)
    pushl    PURCHASE_REF(%edi)
    pushl    QUANTITY_REF(%edi)
    pushl    SUPPLIER_REF(%edi)
    pushl    EXPIRATION_REF(%edi)
    pushl    TYPE_REF(%edi)
    pushl    LOT_REF(%edi)
    pushl    NAME_REF(%edi)
    pushl    $format_write_string
    call     printf
    addl     $36, %esp

    # update current node
    movl     NEXT_REF(%edi), %edi
    jmp      start_report_loop
end_report_loop:
    pushl    $msg_success_report
    call     printf
    addl     $4, %esp

    ret

```

Aqui está um exemplo de como fica o relatório no terminal:

Considere que está é a lista ligada dos produtos em estoque:

```

----LISTA ORDENADA----
Alface -> Arroz -> Batata -> Feijao -> Leite ->

```

A imagem abaixo está cortada devido ao número de atributos.

```

-----RELATÓRIO DE REGISTROS-----
Nome: Alface, Lote: 2, Tipo: Hortifrúti, Validade: 10/08/2025
Nome: Arroz, Lote: 1, Tipo: Mercearia, Validade: 10/07/2026,
Nome: Batata, Lote: 2, Tipo: Hortifrúti, Validade: 30/12/2025
Nome: Feijao, Lote: 1, Tipo: Mercearia, Validade: 10/12/2025,
Nome: Leite, Lote: 5, Tipo: Laticínios, Validade: 10/10/2025,

```

Caso 7 - Gravar Registro em Arquivo

Essa funcionalidade é responsável por gravar o relatório dos produtos em um arquivo separado, chamado de **estoque.txt**. Chamada de sistema (syscall) foi utilizada para implementar essa funcionalidade. Foram gravados todos os nós da lista ligada por meio de um loop. Como os nós estão ordenados por nome de produto, o resultado final no arquivo **estoque.txt** foi a listagem em ordem alfabética dos produtos.

As etapas para fazer essa gravação foram:

- Abrir o arquivo (com chamada de sistema);
- Para cada nó (produto), preencher um buffer (chamado de **node_buffer**) com os dados do produto atual (usou-se a função **sprintf** para isso);
- Gravar o buffer no arquivo **estoque.txt**;
- Ao final do loop, fechar o arquivo

Abaixo estão trechos de código, mostrando a implementação das etapas listadas acima:

Chamada de sistema para abrir o arquivo:

```
write_to_disk:
    # this function will deal with system calls to write in an external file
    # open file (system open)
    movl    $5, %eax
    movl    $filename, %ebx
    movl    fileflags, %ecx
    movl    filemode, %edx
    int     $0x80
    movl    %eax, file_ptr
```

```
filename:        .asciz "estoque.txt"
fileflags:       .int 577 # O_WRONLY (4) | O_CREAT (512) | O_TRUNC (1) = 577 (em
binário)
filemode:       .int 0644
```

- %eax = 5 (número da *syscall* para abrir arquivo);
- %ebx = Nome do arquivo (**filename** contém **estoque.txt**);
- %ecx = Flags para abertura do arquivo. Nesse caso será aberto apenas para escrita (O_WRONLY), deve-se criar o arquivo caso ele não exista (O_CREAT)

e, também, caso ele exista, deve-se apagar o conteúdo existente no arquivo (O_TRUNC);

- %edx = Define as permissões de acesso do novo arquivo.

O retorno dessa *syscall* é o ponteiro para o arquivo de saída (conhecido como **file descriptor**) presente em %eax. Depois, armazenou-se o valor de %eax em uma na variável **file_ptr**.

Preencher o **node_buffer** com os atributos de cada nó:

```
# using sprintf to fill our buffer with the format_string
pushl   SALE_REF(%edi)
pushl   PURCHASE_REF(%edi)
pushl   QUANTITY_REF(%edi)
pushl   SUPPLIER_REF(%edi)
pushl   EXPIRATION_REF(%edi)
pushl   TYPE_REF(%edi)
pushl   LOT_REF(%edi)
pushl   NAME_REF(%edi)
pushl   $format_write_string
pushl   $node_buffer
call    sprintf
addl    $40, %esp
```

Chamada de sistema para escrever node_buffer no arquivo:

```
# calculate the string lenght, assign to %edx (requirement for syscall write)
pushl   $node_buffer
call    strlen
addl    $4, %esp
movl    %eax, %edx

# write to file (syscall write)
movl    $4, %eax
movl    file_ptr, %ebx
movl    $node_buffer, %ecx
int     $0x80
```

- %eax = 4 (número da *syscall* para escrever no arquivo);
- %ebx = **file_ptr**;
- %ecx = **node_buffer**;
- %edx = tamanho da string do buffer.

Chamada de sistema para fechar arquivo:

```
# close file (syscall close)
    movl    $6, %eax
    movl    file_ptr, %ebx
    int     $0x80
```

- %eax = 6 (número da *syscall* para fechar arquivo);
- %ebx = ***file_ptr***.

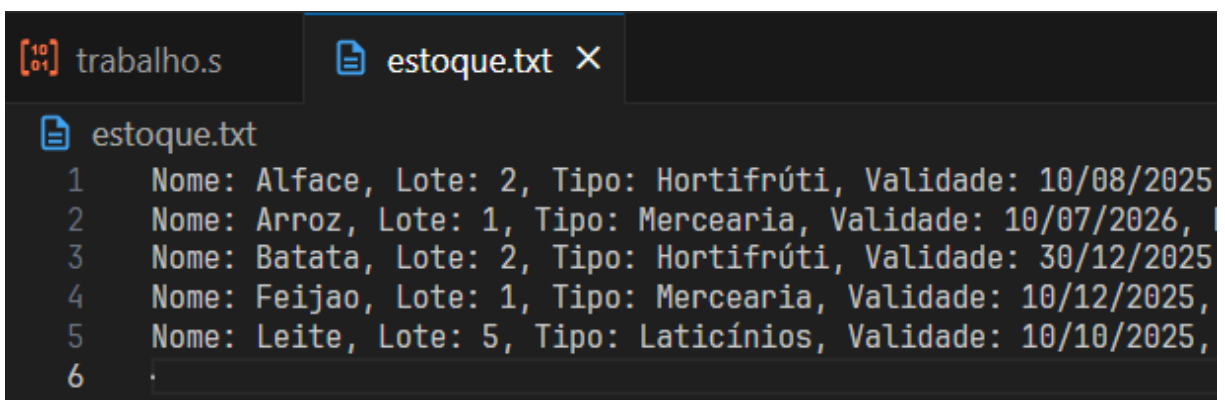
Depois de gravado os produtos no arquivo, a lista ligada é completamente excluída, como é possível ver no print:

```
=====
=== CONTROLE DE ESTOQUE ===
=====
1. Inserir Novo Produto
2. Remover Produto
3. Atualizar Produto
4. Consultar Produto
5. Consulta Financeira
6. Gerar Relatório
7. Gravar Registros em Arquivo
8. Carregar Registros do Arquivo
0. Sair do Programa
-----
Escolha uma opção: 7

[SUCESSO] Dados salvos em 'estoque.txt'.

----LISTA ORDENADA----
```

Aqui está o arquivo ***estoque.txt*** gerado a partir da execução de um exemplo:



The screenshot shows a terminal window with two tabs: 'trabalho.s' and 'estoque.txt'. The 'estoque.txt' tab is active, displaying the following content:

```
estoque.txt
1  Nome: Alface, Lote: 2, Tipo: Hortifrúti, Validade: 10/08/2025
2  Nome: Arroz, Lote: 1, Tipo: Mercearia, Validade: 10/07/2026,
3  Nome: Batata, Lote: 2, Tipo: Hortifrúti, Validade: 30/12/2025
4  Nome: Feijao, Lote: 1, Tipo: Mercearia, Validade: 10/12/2025,
5  Nome: Leite, Lote: 5, Tipo: Laticínios, Validade: 10/10/2025,
6  .
```

Caso 8 - Carregar Registros do Arquivo

Para carregar os registros presentes em **estoque.txt** de volta para a lista ligada usando chamadas de sistema, devemos seguir a seguinte sequência de passos:

- Abrir o arquivo para leitura;
- Percorrer o arquivo inteiro através de um loop. A leitura será feita linha por linha, visto que cada produto ocupa exatamente uma linha do arquivo. Para cada linha lida, vamos colocar os valores dos atributos do produto em suas devidas variáveis globais por meio da função **sscanf** e reutilizar as funções **create_node**, **fill_node_with_data** e **insert_node_by_name** para recriar a lista ligada;
- Ao final, já com todos os nós devidamente lidos e novamente alocados na lista, basta fazer o fechamento do arquivo.

Abaixo estão trechos de código, mostrando a implementação das etapas listadas acima:

Abrir o arquivo para leitura:

```
# Open file (read only)
movl    $5, %eax
movl    $filename, %ebx
movl    $0, %ecx      # file flag O_RDONLY
int     $0x80
movl    %eax, file_ptr
```

- %eax = 5 (número da *syscall* para abertura de arquivo)
- %ebx = Nome do arquivo
- %ecx = Flags para abertura do arquivo. Nesse caso será aberto apenas para leitura (O_RDONLY)

Lendo o arquivo estoque.txt:

```
start_read_loop:
    call    read_line

    # Check if end of file has been reached
    cmpl    $0, %eax
    je      end_read_loop

    # using sscanf to populate the global variables
```

```

pushl    $sale_price
pushl    $purchase_price
pushl    $stock_quantity
pushl    $supplier
pushl    $expiry_date
pushl    $type_string
pushl    $lot
pushl    $product_name
pushl    $format_read_string
pushl    $node_buffer
call     sscanf
addl     $40, %esp

# recreating the linked list
call     create_node
call     fill_node_with_data
call     insert_node_by_name

```

É a função ***read_line*** que faz a leitura de uma linha do arquivo byte por byte até encontrar um caractere “\n”. Assim como está exemplificado no código:

```

read_line:
    movl    $0, buffer_index

start_read_line_loop:
    # syscall read
    movl    $3, %eax
    movl    file_ptr, %ebx
    movl    $temp_char, %ecx
    movl    $1, %edx
    int     $0x80

    cmpl    $0, %eax
    je      end_of_file

    # Copy the byte to node_buffer[buffer_index]
    movl    buffer_index, %esi
    movb    temp_char, %al
    movb    %al, node_buffer(%esi)

    # Check if the byte is '\n' (end of line)
    cmpb    $'\n', temp_char
    je      end_read_line_loop

    incl    buffer_index
    jmp     start_read_line_loop
end_read_line_loop:
    # Add string terminator '\0'
    movl    buffer_index, %esi
    movb    $0, node_buffer(%esi)
    ret

end_of_file:
    ret

```

A chamada de sistema para leitura na função ***read_line*** espera os seguintes parâmetros:

- `%eax = 3` (número da *syscall* para leitura de arquivo já aberto);
- `%ebx = file_ptr`;
- `%ecx =` buffer para armazenar o dado lido. Neste caso é a variável ***temp_char*** que tem espaço para armazenar 1 byte lido, ou seja, 1 caractere;
- `%edx =` Número de bytes a ler (1 byte).

Dessa forma, em uma linha os bytes vão sendo lidos 1 por 1 e armazenados temporariamente na variável ***temp_char*** para serem transferidos a um buffer maior capaz de armazenar a linha inteira (***node_buffer***).

Fechamento do arquivo:

```
# syscall close file
    movl    $6, %eax
    movl    file_ptr, %ebx
    int     $0x80
```

Abaixo é possível visualizar um exemplo da execução deste caso. Lembrando que o arquivo ***estoque.txt*** estava preenchido com os produtos do caso anterior.

```
Escolha uma opção: 8

[SUCESSO] Produto inserido na lista.
[SUCESSO] Produto inserido na lista.
[SUCESSO] Produto inserido na lista.
[SUCESSO] Produto inserido na lista.
[SUCESSO] Produto inserido na lista.
[SUCESSO] Dados carregados de 'estoque.txt'.

----LISTA ORDENADA----
Alface -> Arroz -> Batata -> Feijao -> Leite ->
```