Memorando

**De:** Pedro Afonso

**Nº de Matrícula:** 20220252

**Disciplina:** CPD

**Assunto:** Experiência da aula prática

**Data:** 16/03/24

# 1. Introdução

Este memorando descreve a execução de uma experiência laboratorial no desenvolvimento de aplicações em linguagem C no ambiente Linux. A experiência incluiu a utilização de ferramentas como gcc, gdb, make, e abordou técnicas de depuração e geração de executáveis. Foram realizados testes e manipulações de código, além de análise de erros comuns, como segmentation fault. As capturas de tela inseridas ao longo do memorando ilustram a execução dos passos descritos.

# 2. Experiências Realizadas

Durante a experiência, foram realizados os seguintes passos:

**Ciclo de Desenvolvimento e Geração do Executável:**

* Descompactação do arquivo com o comando:
* $ tar -zxvf aula1-eg1.tgz
* Compilação dos arquivos fonte:
* $ gcc -g -c list.c main.c
* Ligação dos objetos para criar o executável:
* $ gcc -o main list.o main.o
* Execução do programa:
* $ ./main

**Utilização do Debugger GDB:**

* Execução do programa com o gdb:
* $ gdb main
* Colocação de breakpoint na função insert\_new\_process na linha 36:
* $ b list.c:36
* Execução do programa até o breakpoint:
* $ r
* Visualização de variáveis no escopo atual:
* $ p item
* $ p \*item

**Nota:** O comando display \*item só funcionará se, na sequência de execução, estivermos no escopo em que a variável item pertence.

1. Controle de execução com os comandos step e next para avançar pelo código.

**Análise de Erros:**

* Definição do tamanho do arquivo core dump:
* $ ulimit -c 10000000

**Nota:** O comando define o limite de 10 MB, mas o arquivo só será gerado se ocorrer um erro no programa.

* Análise de segmentation fault com o gdb:
* $ gdb main core
* $ bt

O comando backtrace (bt) mostra a pilha de chamadas até o ponto do erro.

**Utilização da Ferramenta Make:**

* Criação do arquivo Makefile e execução do comando make para compilar o programa.
* Teste da recompilação ao modificar arquivos fonte ou cabeçalhos.
* Utilização da regra clean para limpar os arquivos objeto e o executável:
* $ make clean

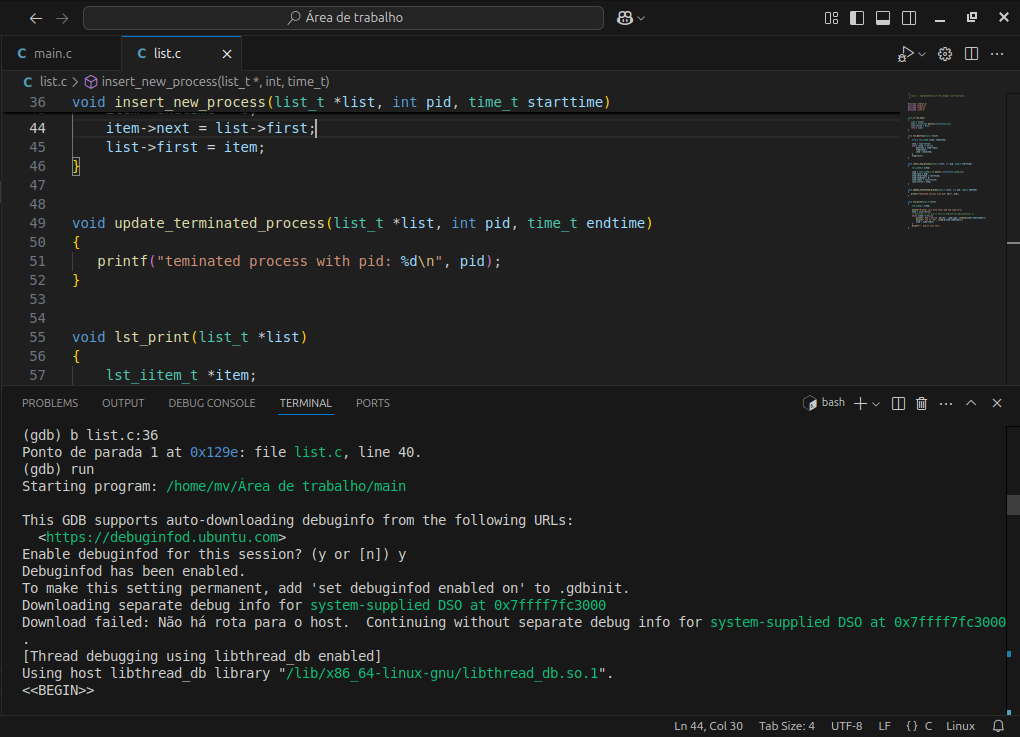


Figura - Uso do gbd run, break

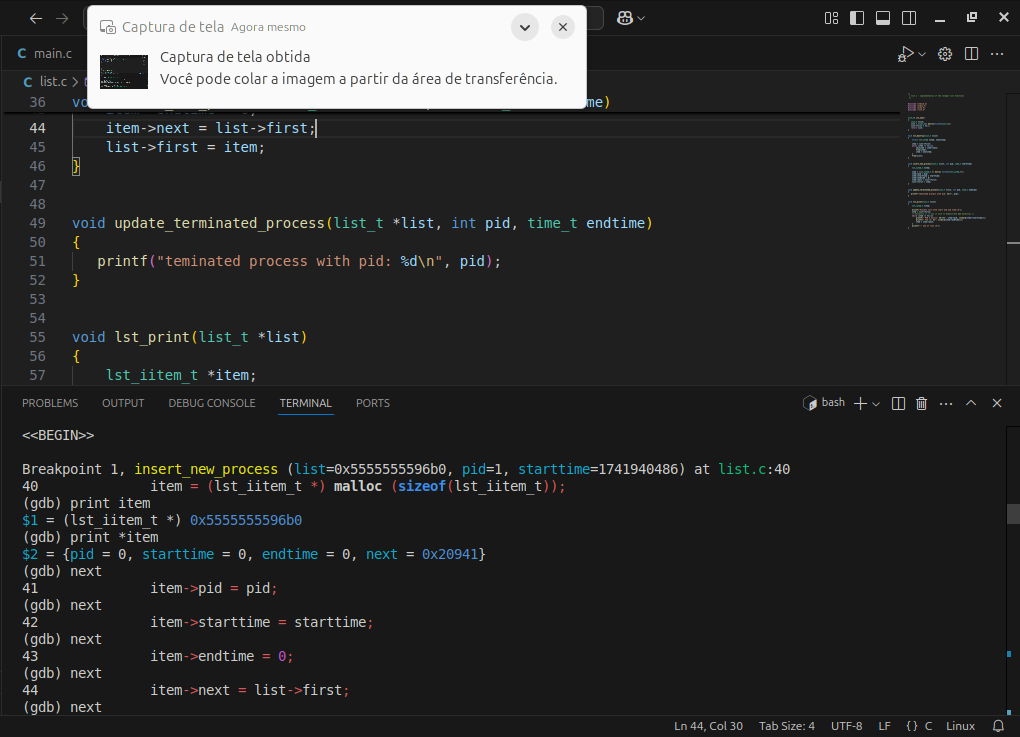


Figura - Uso de gbd next e break

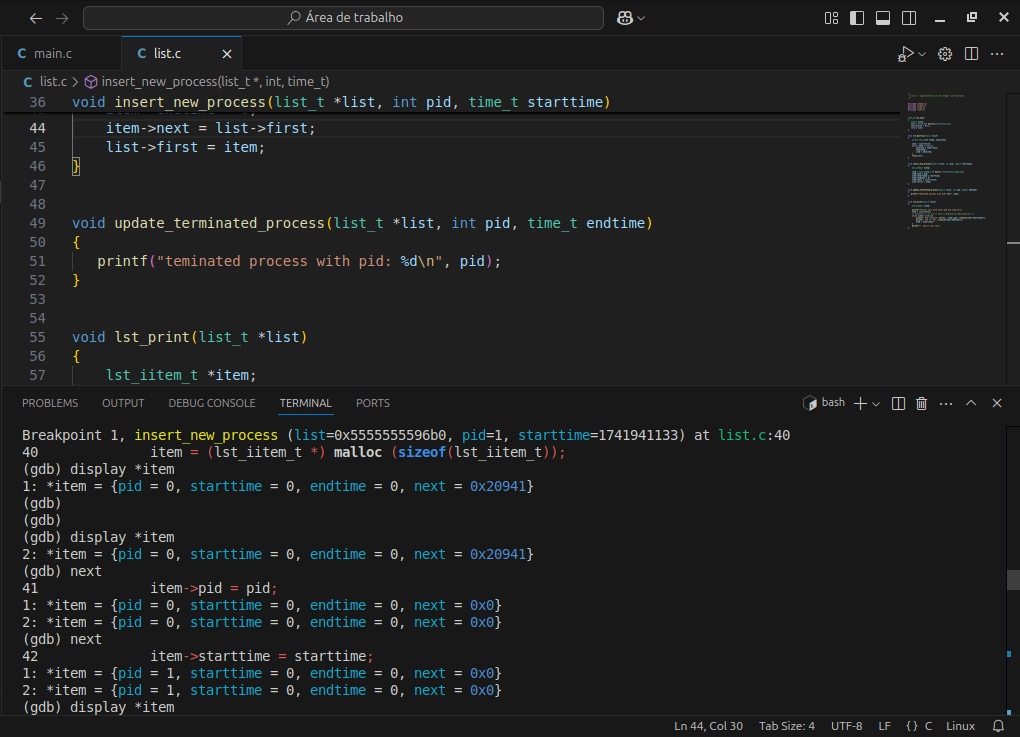


Figura - Uso do comando gbd display

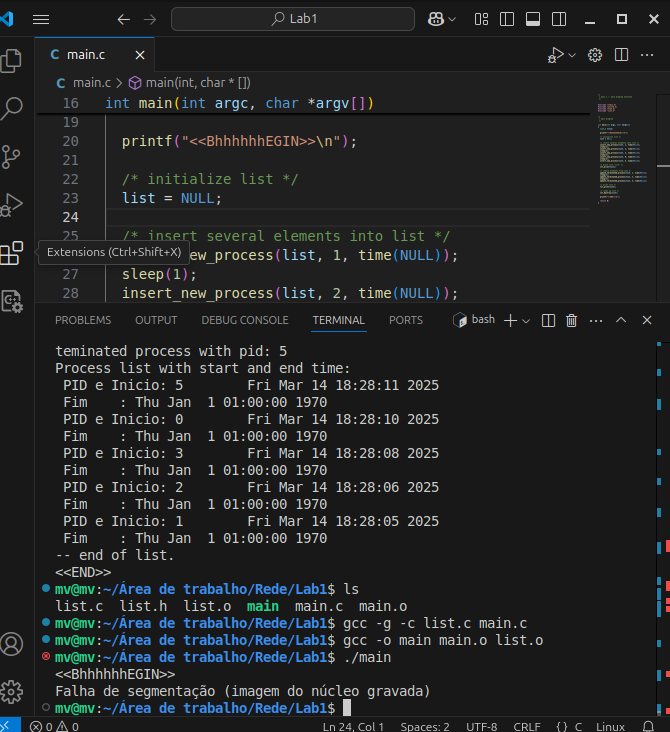


Figura - Erro de falha de seguimentação

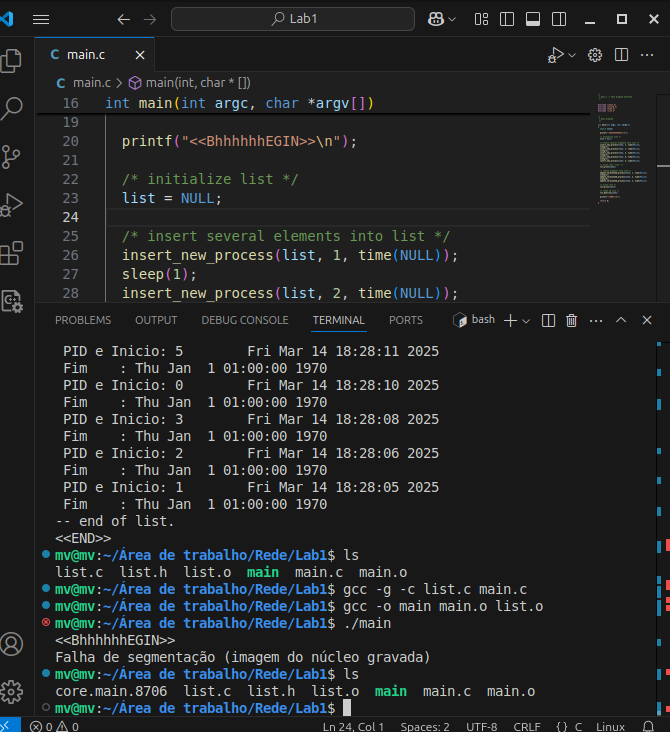


Figura - Arquivo Core Dump gerado

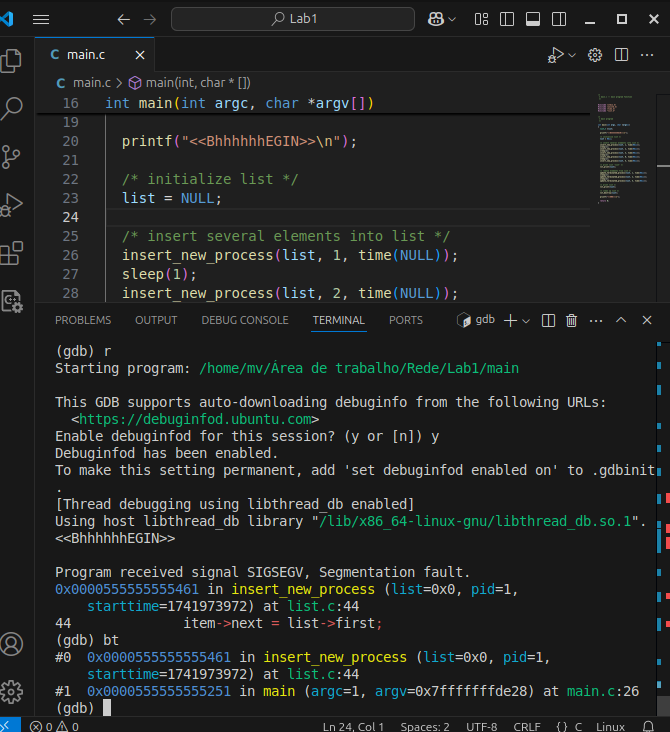


Figura - Visualizando o pilha de erro com bt

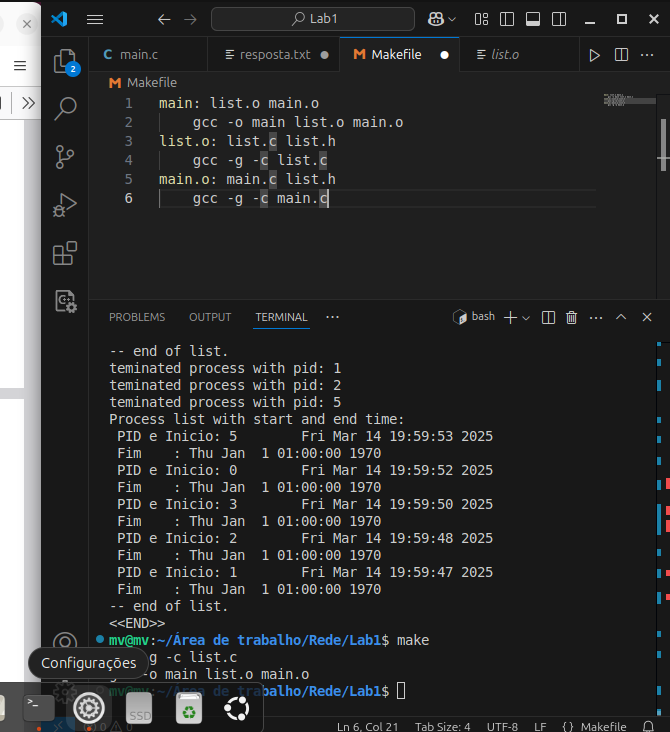


Figura - Usu do make alterando o arquivo list.c sem o list.h

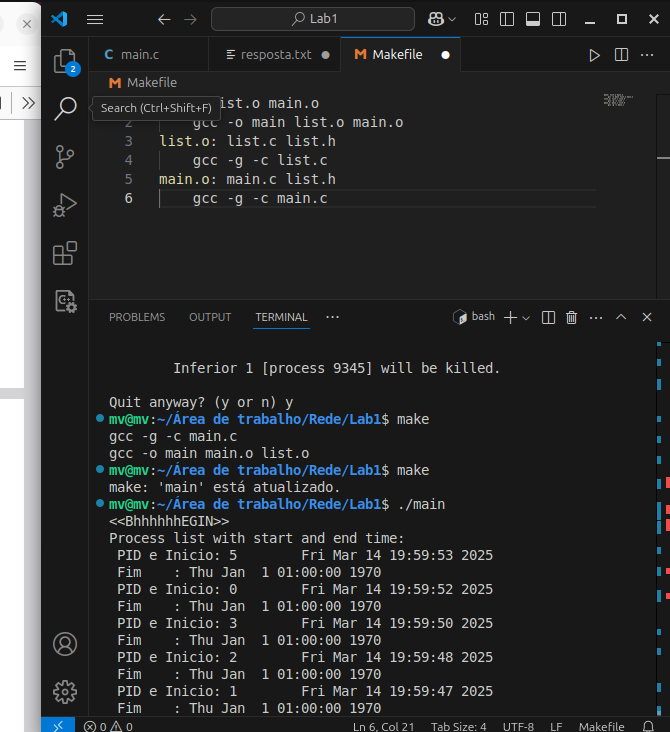


Figura - Uso do make alterando o arquivo main.c

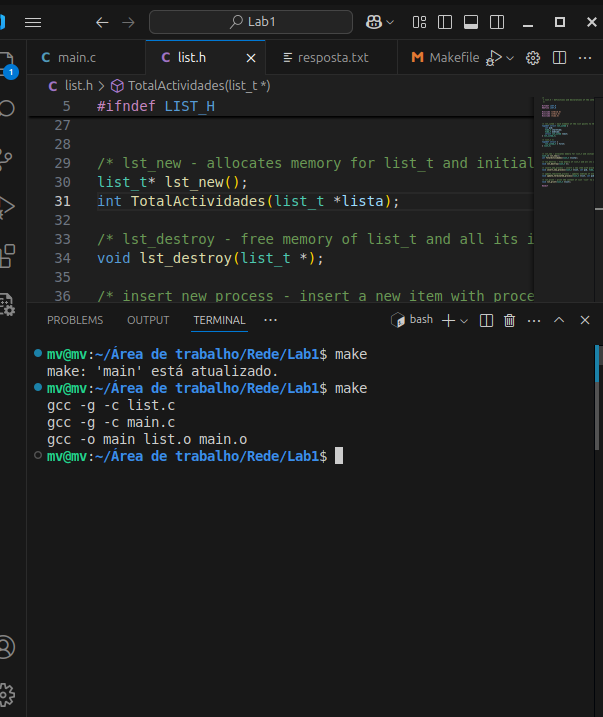


Figura - Usando o make para executar o programa

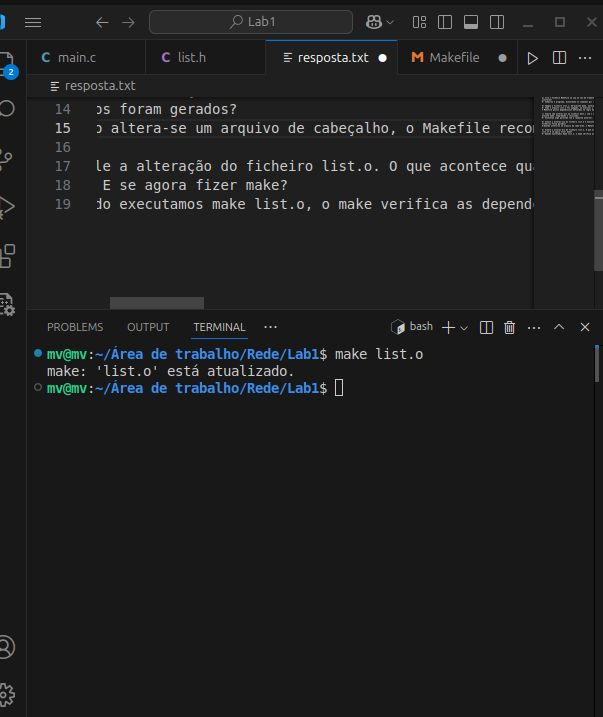


Figura - Usando o make list.o para executar a regra list.o

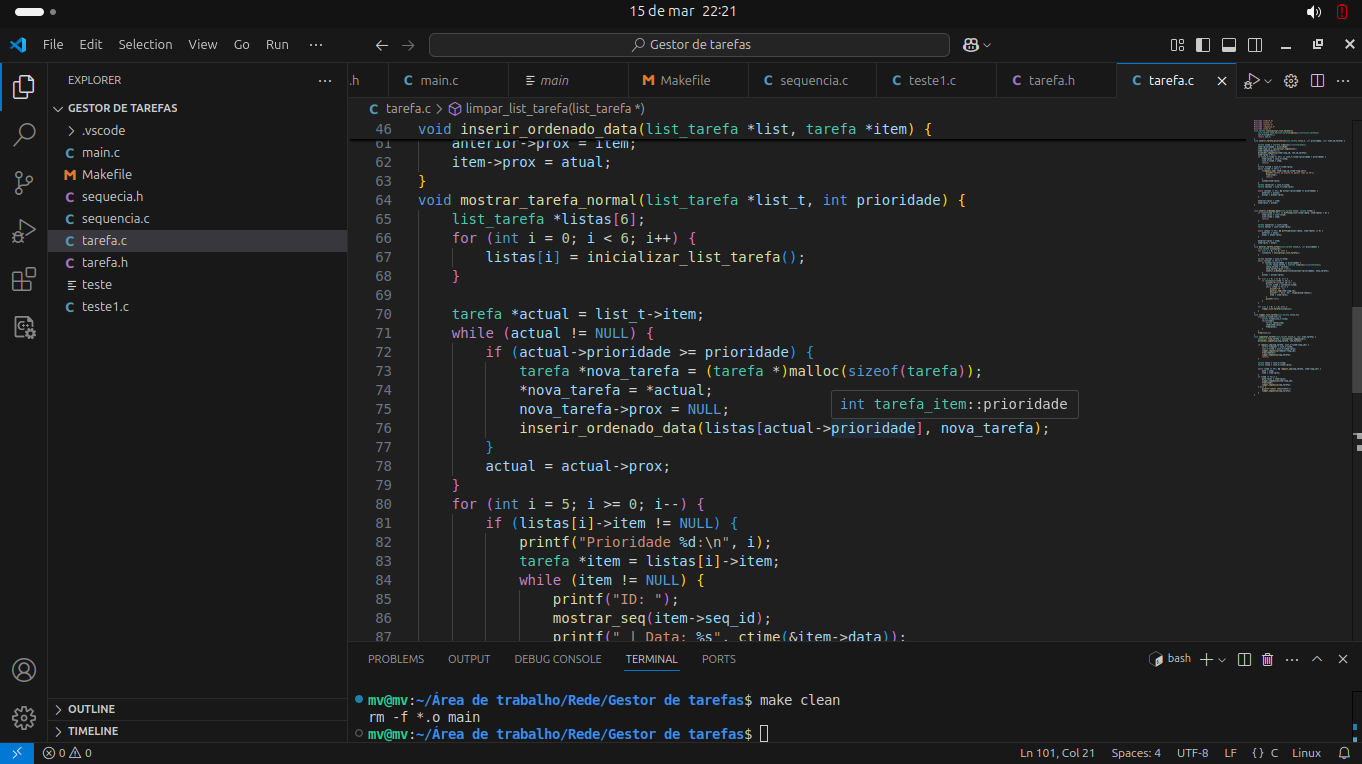


Figura - Usando o make clean

**Solução da funão update\_terminated\_process**

1. void update\_terminated\_process(list\_t \*list, int pid, time\_t endtime) {
2. lst\_iitem\_t \*item = list->first;
3. while (item != NULL) {
4. if (item->pid == pid) {
5. item->endtime = endtime;
6. return;
7. }
8. item = item->next;
9. }
10. }

# 3. Desafios

Durante o desenvolvimento do projeto, alguns desafios foram enfrentados, como a compreensão das etapas de compilação e depuração no Unix e a análise de erros em tempo de execução.

a) Crie o ficheiro Makefile na sua á rea de trabalho e execute make. O que

aconteceu?

R: Compilou o programa, executando os comandos gcc -g -c main.c, gcc -g -c list.c, depois linkou os arquivos .o que foram compilados.

b) Apague o ficheiro list.o. Re-execute make. Interprete o sucedido.

R: Como o arquivo list.o foi apagado, ao executar o comando make novamente, ele verifica que o arquivo objeto não está presente e, portanto, recompila o arquivo list.c para gerar o arquivo list.o novamente.

O Makefile possui dependências definidas na regra main, que dependem dos arquivos list.o e main.o. Sempre que um arquivo objeto (.o) necessário estiver ausente ou desatualizado, o make executa a regra correspondente para gerá-lo antes de criar o executável final.

c) Simule uma alteração ao ficheiro main.c com o comando seguinte e re-

execute make. Compreenda o resultado.

R: Aconteceu algo parecido com a resposta anterior. O make percebeu que o arquivo main.c foi alterado e, por isso, recompilou apenas a dependência main.o para gerar um novo arquivo objeto atualizado. Em seguida linkou os arquivos object(.o).

d) Simule a alteração do ficheiro list.h e execute make. Porque razão todos os

ficheiros foram gerados?

R:Quando altera-se um arquivo de cabeçalho, o Makefile recompila todos os arquivos que dependem desse cabeçalho, assim sendo foram recompilados o main.c o eo list.c e depois linkados novamente.

e) Simule a alteração do ficheiro list.o. O que acontece quando faz make

list.o? E se agora fizer make?

R:Quando executamos make list.o, o make verifica as dependências e, como houve alteração no list.c, recompila o arquivo com os dados atualizados do list.h e não relinka. Se rodarmos apenas make, ele vai relinkar não vai executar as dependências.

g) Adicione a regra seguinte no fim do ficheiro. O que descreve esta regra?

Identifique: o alvo, as dependências e o comando.

R: Regra ou Alvode descreve o seguinte.

Um comando que apaga todos os arquivos do forma .o e o arquivo executavel main.

Alvo: clean.

Dependenciais: nenhuma

h) Execute make clean. O que aconteceu? Porque razão o comando é

executado sempre que esta regra é invocada explicitamente?

R: Apagou os arquivos de formato .o e o arquivo main.

Ele e acontece sempre que chamamos a regra clean explicitamente.Implementação da Função update\_terminated\_process

**Solução da funão update\_terminated\_process**

A função update\_terminated\_process foi desenvolvida para atualizar o tempo de término de um processo identificado pelo seu PID em uma lista encadeada. A função percorre a lista, localiza o processo correspondente e atualiza o campo endtime.

1. Código da função:
2. void update\_terminated\_process(list\_t \*list, int pid, time\_t endtime) {
3. lst\_iitem\_t \*item = list->first;
4. while (item != NULL) {
5. if (item->pid == pid) {
6. item->endtime = endtime;
7. return;
8. }
9. item = item->next;
10. }
11. }

# 4. Referências Bibliográficas

* Documentação oficial do GCC e GDB.
* Tutoriais sobre depuração e uso de Makefile no ambiente Unix.
* Apostilas e notas de aula fornecidas pelo professor.

# 5. Repositório GitHub

https://github.com/pedroaly/Memorando\_1.