

## MEMORANDO

**DE:** Silvio Jose

**Nº MATRÍCULA:** 20212175

**DISCIPLINA:** Computação Paralela e Distribuída

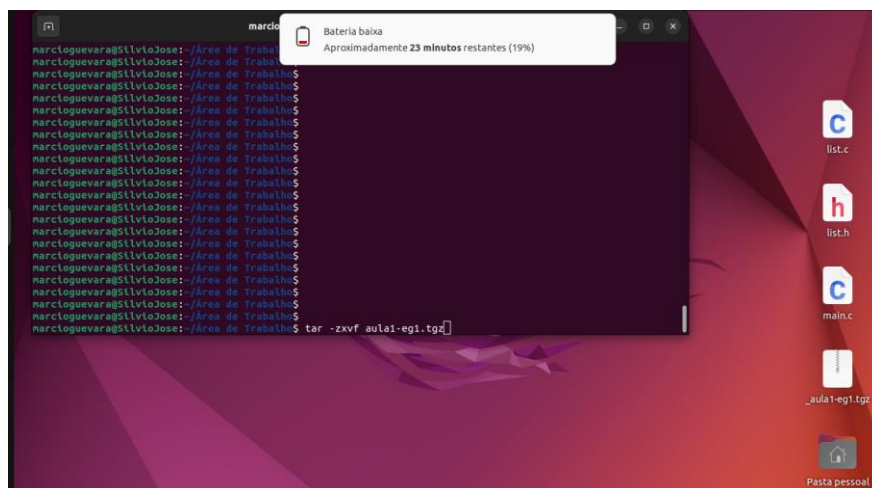
**TURMA:** EINF6-M3

**ASSUNTO:** Introdução ao Ambiente Unix

**DATA:** 20 de Março de 2024

Copie o ficheiro aula1-eg1.tgz para a sua área de trabalho e descomprima o seu conteúdo com o comando:

\$ tar -zxvf aula1-eg1.tgz

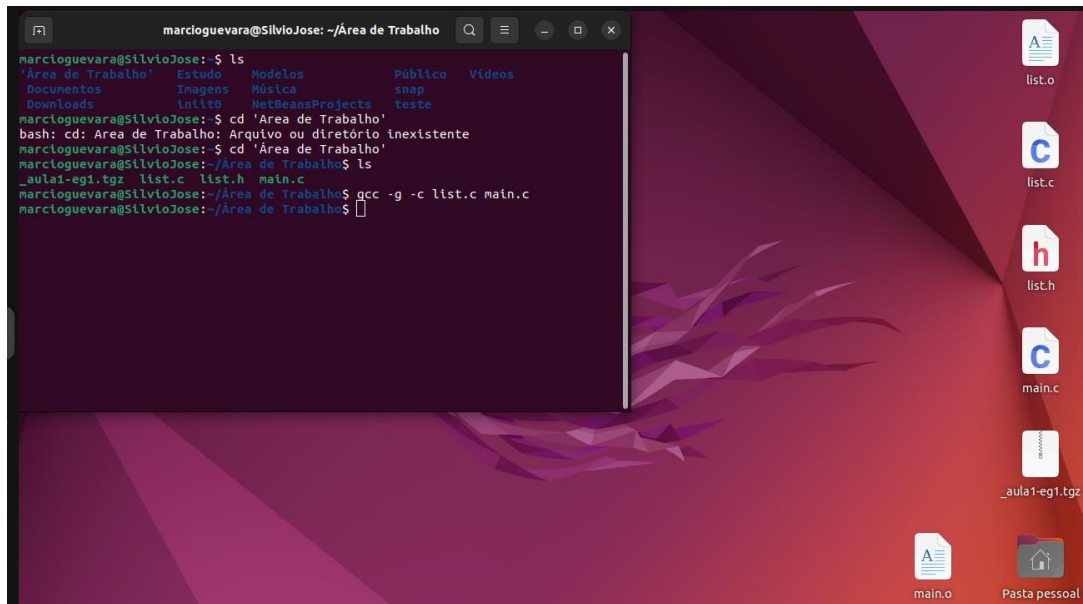


### 3. Geração do executável

a) Visualize e compreenda o conteúdo dos ficheiros.

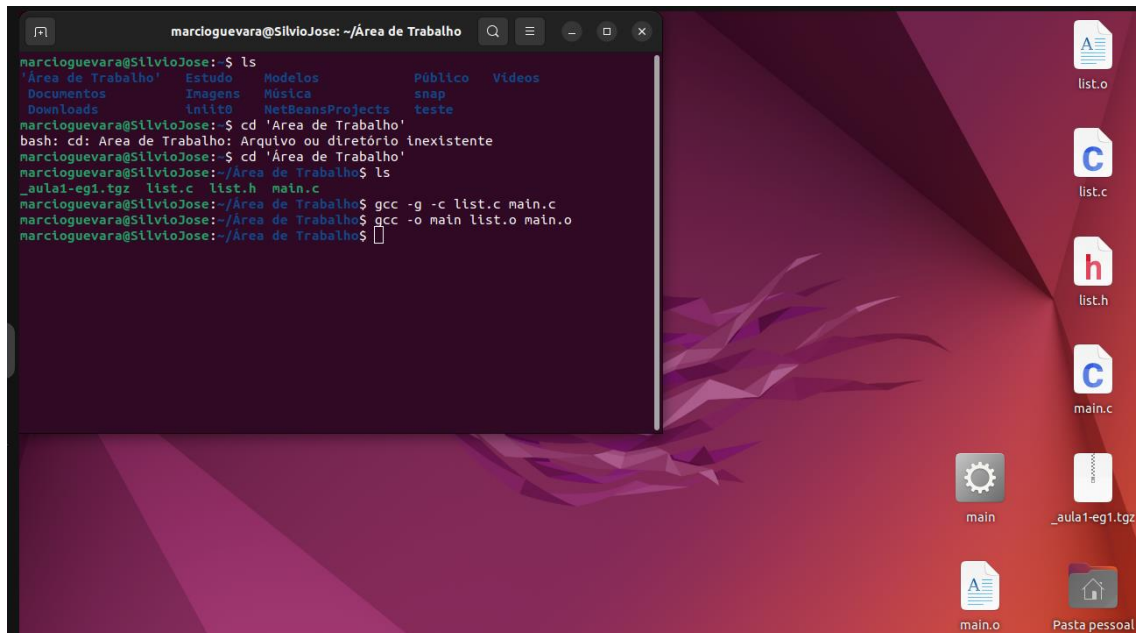
b) Compile os ficheiros. Verifique depois que os ficheiros objecto foram criados usando o comando "ls" através da interface operacional (shell).

\$ gcc -g -c list.c main.c



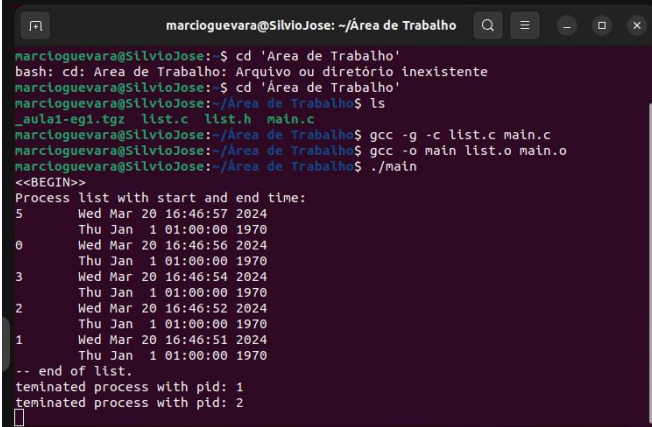
c) Faça a ligação dos ficheiros objecto de modo a produzir o executável denominado main.

\$ gcc -o main list.o main.o



d) Execute a aplicação main e compreenda o resultado.

\$ ./main

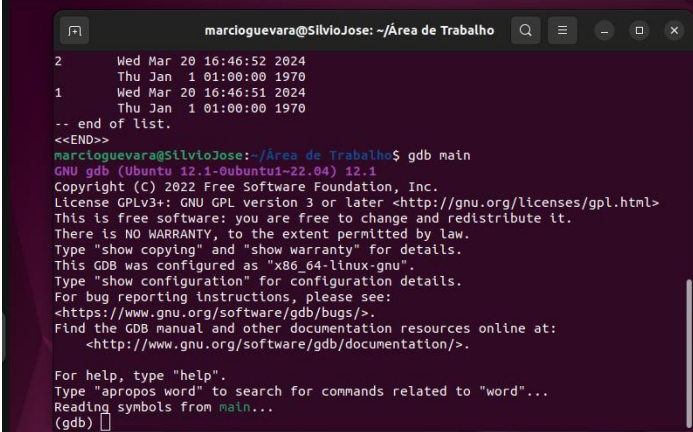


```
marcioguevara@SilvioJose: ~/Área de Trabalho
marcioguevara@SilvioJose: $ cd 'Área de Trabalho'
bash: cd: Área de Trabalho: Arquivo ou diretório inexistente
marcioguevara@SilvioJose: $ cd 'Área de Trabalho'
marcioguevara@SilvioJose: ~/Área de Trabalho$ ls
_aula1-eg1.tgz list.c list.h main.c
marcioguevara@SilvioJose: ~/Área de Trabalho$ gcc -g -c list.c main.c
marcioguevara@SilvioJose: ~/Área de Trabalho$ gcc -o main list.o main.o
marcioguevara@SilvioJose: ~/Área de Trabalho$ ./main
<<BEGIN>>
Process list with start and end time:
5   Wed Mar 20 16:46:57 2024
   Thu Jan 1 01:00:00 1970
0   Wed Mar 20 16:46:56 2024
   Thu Jan 1 01:00:00 1970
3   Wed Mar 20 16:46:54 2024
   Thu Jan 1 01:00:00 1970
2   Wed Mar 20 16:46:52 2024
   Thu Jan 1 01:00:00 1970
1   Wed Mar 20 16:46:51 2024
   Thu Jan 1 01:00:00 1970
-- end of list.
terminated process with pid: 1
terminated process with pid: 2
█
```

#### 4. Utilização do debugger gdb

a) Execute a aplicação no debugger gdb:

\$ gdb main



```
marcioguevara@SilvioJose: ~/Área de Trabalho
2   Wed Mar 20 16:46:52 2024
   Thu Jan 1 01:00:00 1970
1   Wed Mar 20 16:46:51 2024
   Thu Jan 1 01:00:00 1970
-- end of list.
<<END>>
marcioguevara@SilvioJose: ~/Área de Trabalho$ gdb main
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
(gdb) █
```

b) Coloque um breakpoint na primeira instrução da função

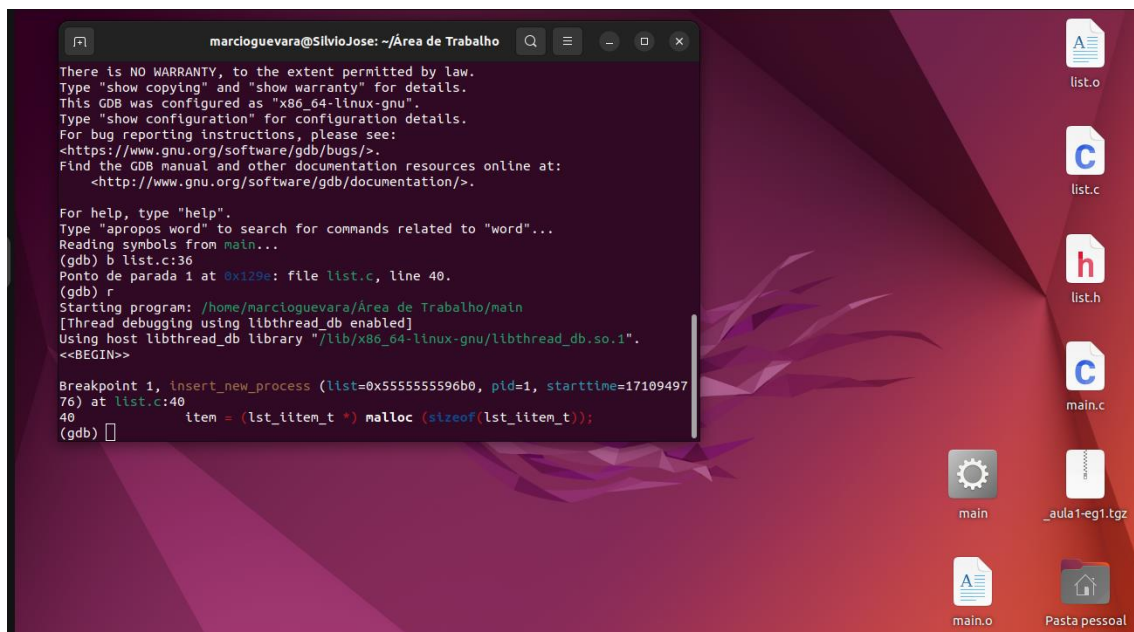
insert\_new\_process (ficheiro list.c e linha 36):

```
$ b list.c:36
```

c) Execute a aplicação usando o comando run:

```
$ r
```

A aplicação executa-se até ao breakpoint.



d) Pode agora ver o valor das variáveis que estão no scope da função usando

o comando print:

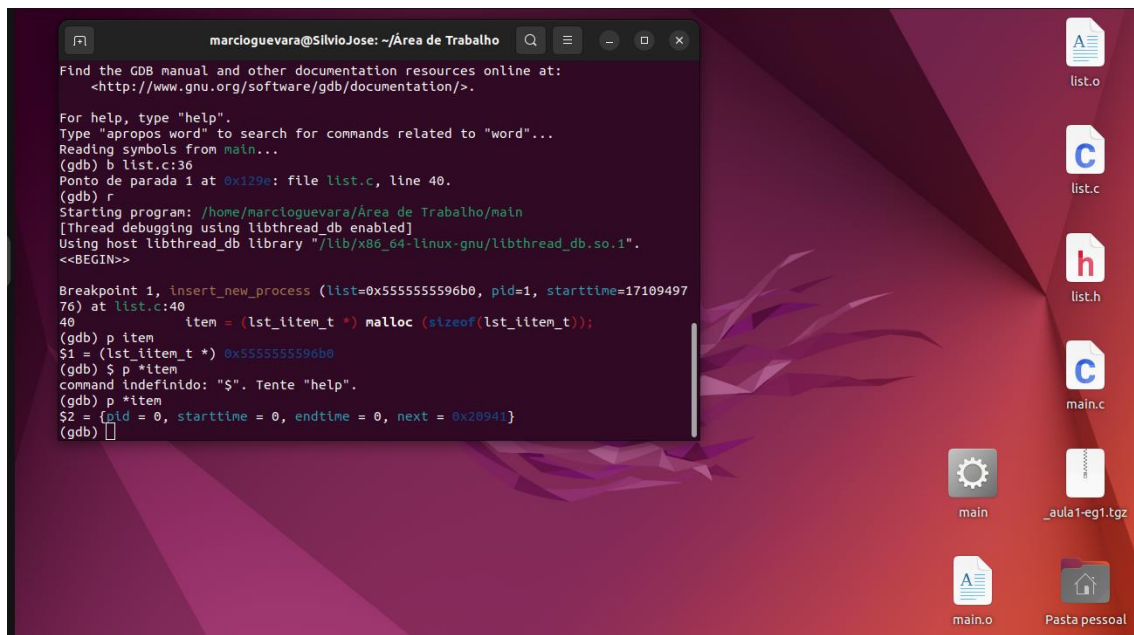
```
$ p item
```

ou

```
$ p *item
```

Qual a diferença entre estes dois comandos?

**R: A diferença entre os dois comandos é que o comando \$ p item apresenta o conteúdo da variável e o comando \$ p \*item desempacota o conteúdo da variável item em elementos.**



e) Pode agora executar a aplicação utilizando o comando de step para executar a próxima instrução entrando dentro das funções:

```
$ s
```

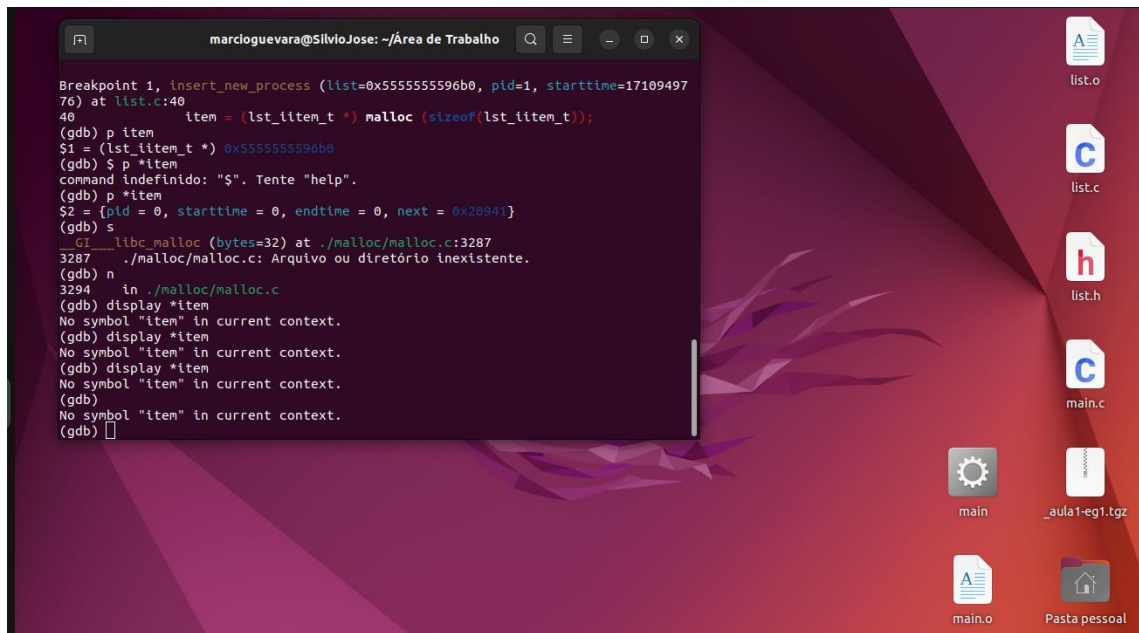
ou o comando de next para executar a próxima instrução sem entrar dentro das funções:

```
$ n
```

f) Enquanto executa os comandos next ou step pode ir executando o comando print para mostrar o valor das variáveis ou executar apenas uma vez o comando display:

```
$ display *item
```



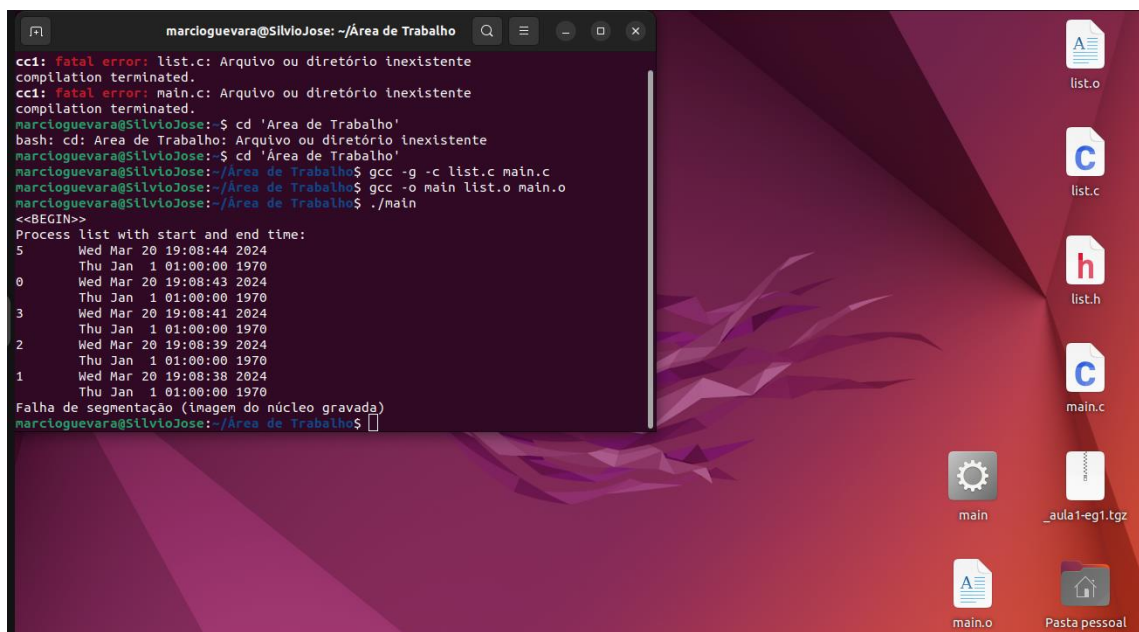


g) Altere agora o ficheiro list.c: Descomente o comentário da linha 61 e comente a linha 62.

h) Execute o seguinte comando fora do gdb para colocar o limite do tamanho do ficheiro core com o valor de 10Mb:

\$ ulimit -c 10000000

i) Gere o executável e execute o programa fora do gdb. O erro de segmentation fault irá ocorrer.



O gdb é uma muita boa ferramenta para saber o que aconteceu. Para isso, comece por listar os ficheiros que estão na diretoria:

\$ ls

Que observa?

**R: observou;se um conjunto de arquivos e ficheiros**

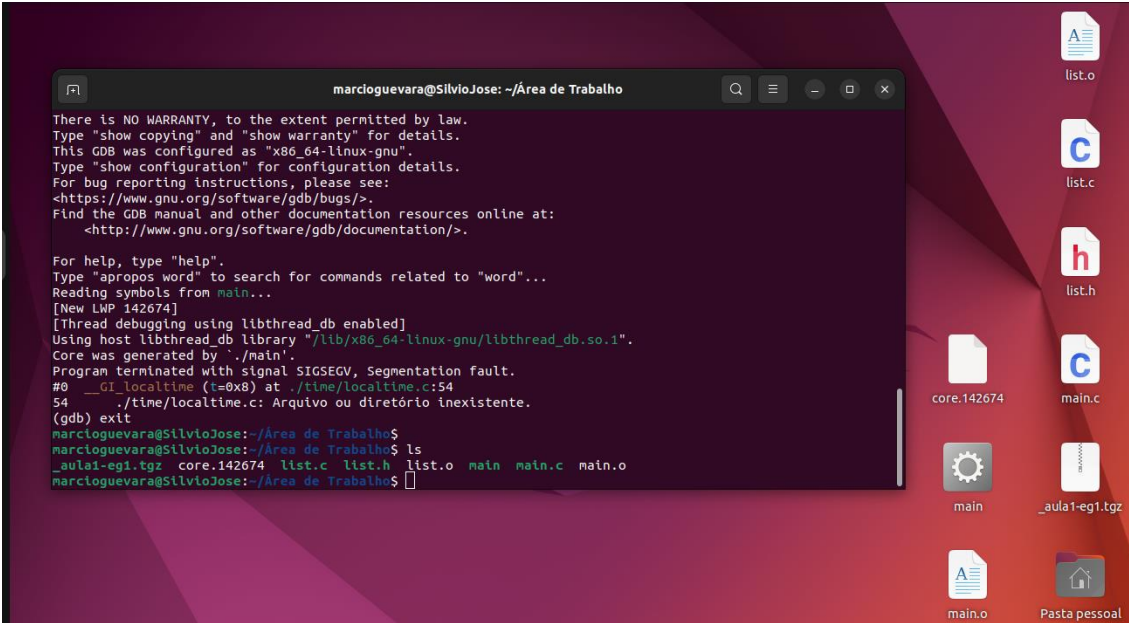
**\_aula1-eg1.tgz list.c list.h list.o main main.c main.o**

Use em seguida o gdb para saber onde ocorreu o erro executando:

\$ gdb main <ficheiro core>

1) O gdb irá ficar parado na instrução onde ocorreu o erro. Para saber qual a instrução onde o erro ocorreu execute o comando backtrace:

\$ bt



```
marcloguevara@SilvioJose: ~/Área de Trabalho
There is NO WARRANTY, to the extent permitted by law.
Type "show copyng" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

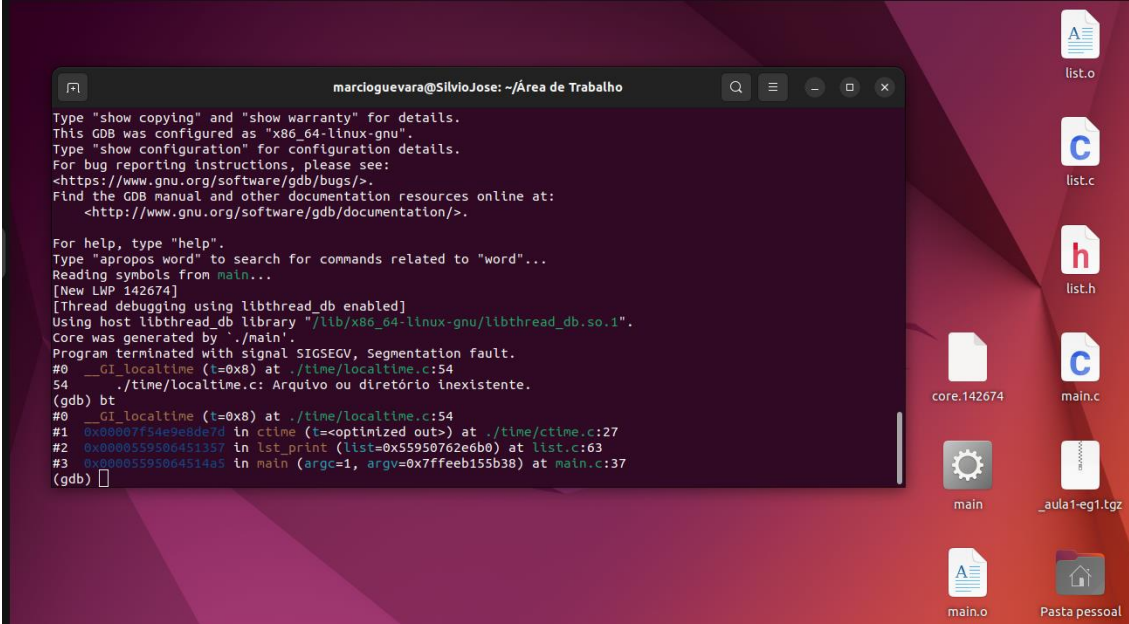
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
[New LWP 142674]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by './main'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  __GI_localtime (t=0x8) at ./time/localtime.c:54
54  ./time/localtime.c: Arquivo ou diretório inexistente.
(gdb) exit
marcloguevara@SilvioJose:~/Área de Trabalho$
marcloguevara@SilvioJose:~/Área de Trabalho$ ls
_aula1-eg1.tgz  core.142674  list.c  list.h  list.o  main  main.c  main.o
marcloguevara@SilvioJose:~/Área de Trabalho$
```

Use em seguida o gdb para saber onde ocorreu o erro executando:

\$ gdb main <ficheiro core>

l) O gdb irá ficar parado na instrução onde ocorreu o erro. Para saber qual a instrução onde o erro ocorreu execute o comando backtrace:

\$ bt



```
marcioguevara@SilvioJose: ~/Área de Trabalho
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
[New LWP 142674]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by './main'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  __GI_localtime (t=0x8) at ./time/localtime.c:54
54  ./time/localtime.c: Arquivo ou diretório inexistente.
(gdb) bt
#0  __GI_localtime (t=0x8) at ./time/localtime.c:54
#1  0x00007f54e9e8de70 in ctime (t=<optimized out>) at ./time/ctime.c:27
#2  0x0000555950645137 in lst_print (list=0x55950762e6b0) at list.c:63
#3  0x0000555950645145 in main (argc=1, argv=0x7fffeb155b38) at main.c:37
(gdb)
```

m) O gdb mostra-lhe assim a função em que ocorreu o erro e todas as funções que chamaram essa até ao nível da função main. Algumas dessas funções são de sistema, mas há duas que podemos reconhecer: lst\_print e main. O primeiro número em cada linha indica o nível em que essa função está.

Para observar as variáveis que estão no nível da função lst\_print deve mudar para esse nível executando o comando frame seguido do nível. Por exemplo:

\$ frame 2

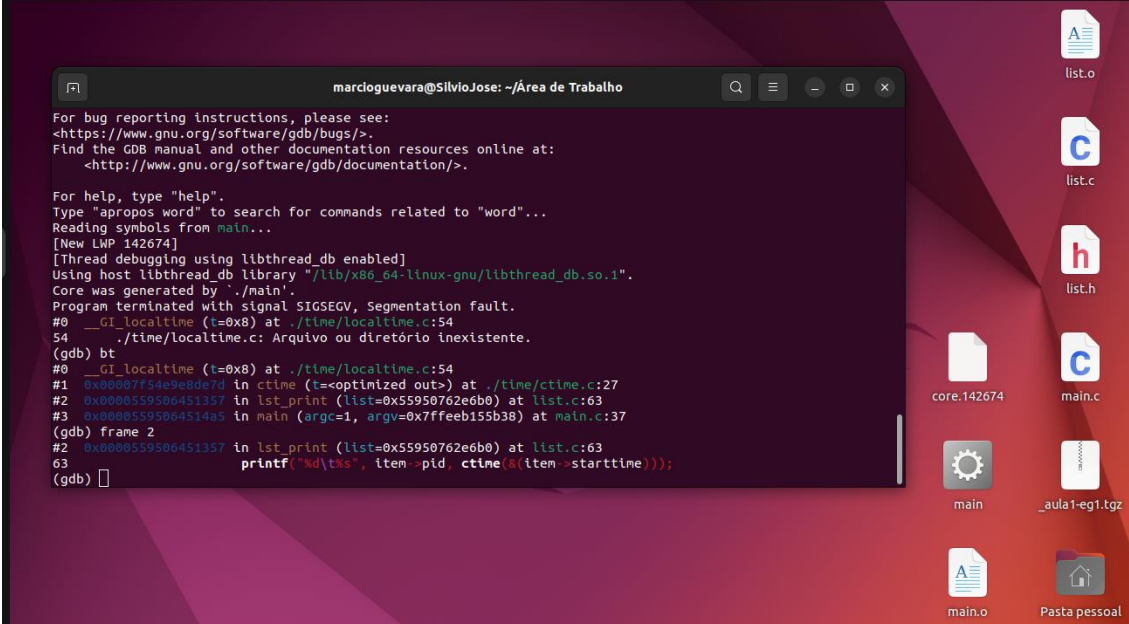
n) Pode agora ver o conteúdo das variáveis que estão no scope da função lst\_print:

\$ frame 2

Pode assim ver que o erro ocorreu porque a variável item é NULL. A partir daqui poderia colocar um breakpoint no início do ciclo while,



correr o programa de novo e seguir depois passo a passo, enquanto vai observando os valores das variáveis, até descobrir o que gerou o segmentation fault.



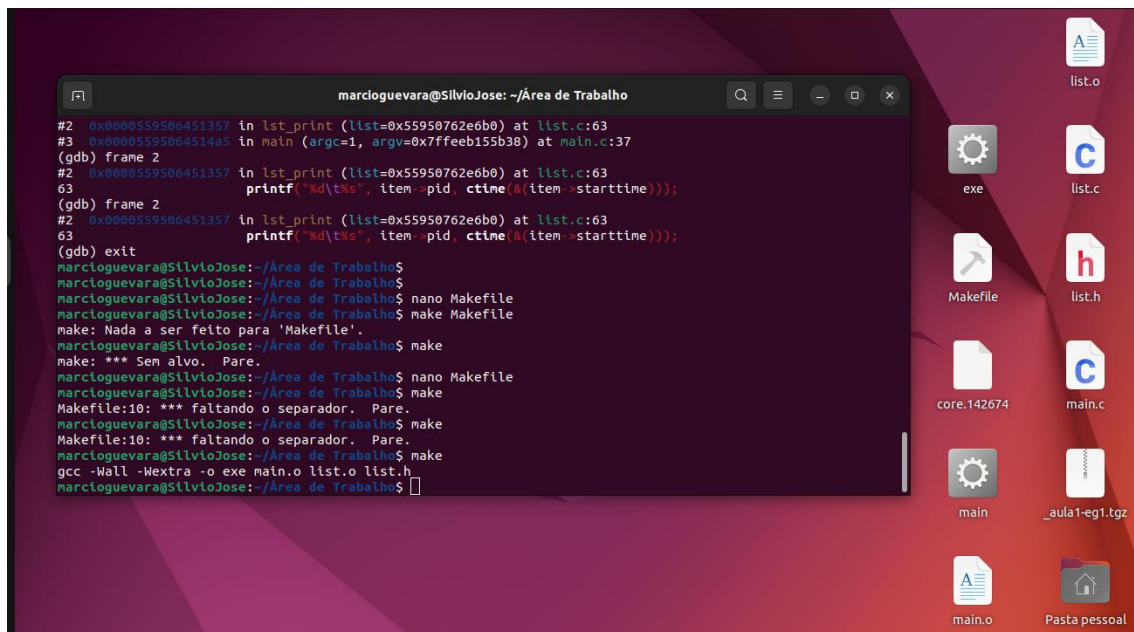
```
marcioguevara@SilvioJose: ~/Área de Trabalho
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
[New LWP 142674]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Core was generated by './main'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  __GI_localtime (t=0x8) at ./time/localtime.c:54
54  ./time/localtime.c: Arquivo ou diretório inexistente.
(gdb) bt
#0  __GI_localtime (t=0x8) at ./time/localtime.c:54
#1  0x00007f54e9e8de7d in ctime (t=<optimized out>) at ./time/ctime.c:27
#2  0x0000559506451357 in lst_print (list=0x55950762e6b0) at list.c:63
#3  0x00005595064514a5 in main (argc=1, argv=0x7fffeb155b38) at main.c:37
(gdb) frame 2
#2  0x0000559506451357 in lst_print (list=0x55950762e6b0) at list.c:63
63  printf("%d\t%s", item->pid, ctime(&(item->starttime)));
(gdb)
```

#### 4. Utilização da ferramenta make

Crie o ficheiro Makefile na sua área de trabalho e execute make. O que aconteceu?

**R:** apos executar o make foi criado um ficheiro executavel exeque juntou o main.c e list.c e list.h



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

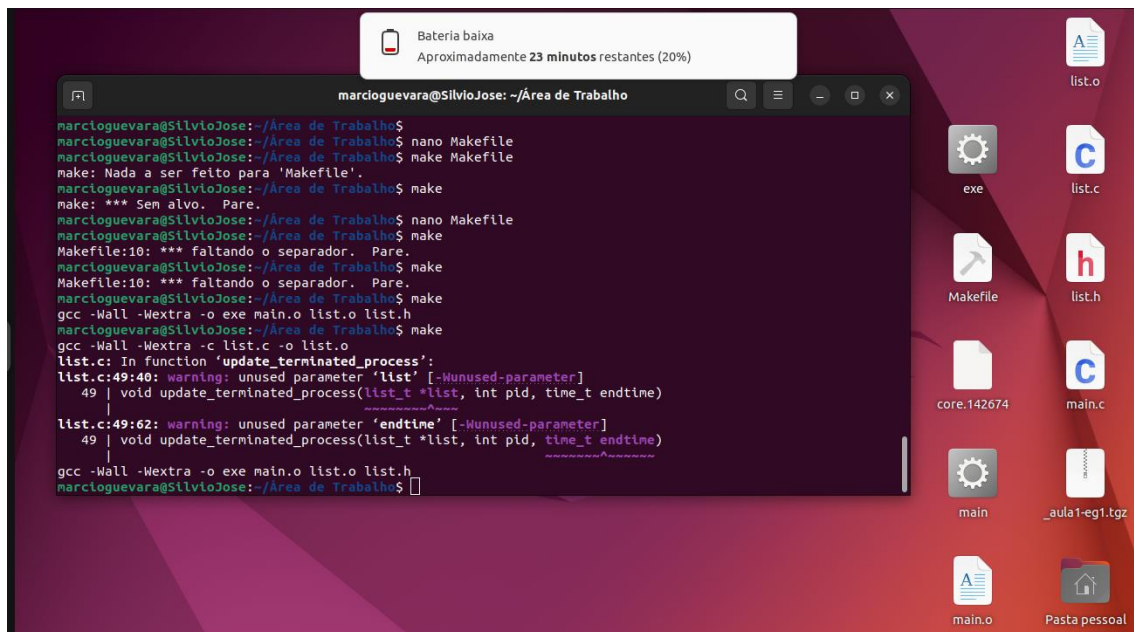
**R:** As mensagens que você está vendo são avisos (warnings) produzidos pelo compilador GCC durante o processo de compilação. Vou explicar o que cada um deles significa:

**warning: unused parameter 'list' [-Wunused-parameter]**

Este aviso indica que você definiu um parâmetro chamado list na função update\_terminated\_process, mas não o utilizou dentro do corpo da função. Isso não é um erro, mas apenas um aviso para alertá-lo de que você tem um parâmetro que não está sendo usado. Isso pode indicar que o parâmetro é desnecessário para essa função ou que você esqueceu de usá-lo em algum lugar dentro da função.

**warning: unused parameter 'endtime' [-Wunused-parameter]**

Similarmente ao primeiro aviso, este aviso indica que você definiu um parâmetro chamado endtime na função update\_terminated\_process, mas não o utilizou dentro do corpo da função.



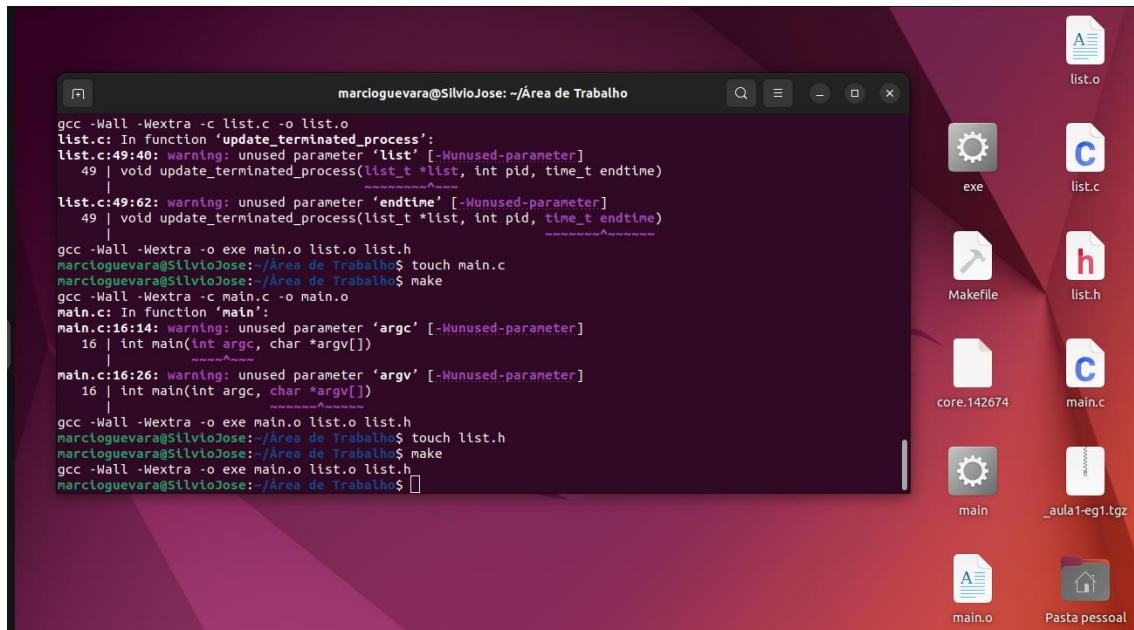
c) Simule uma alteração ao ficheiro main.c com o comando seguinte e re-execute make. Compreenda o resultado.

```
$ touch main.c
```

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

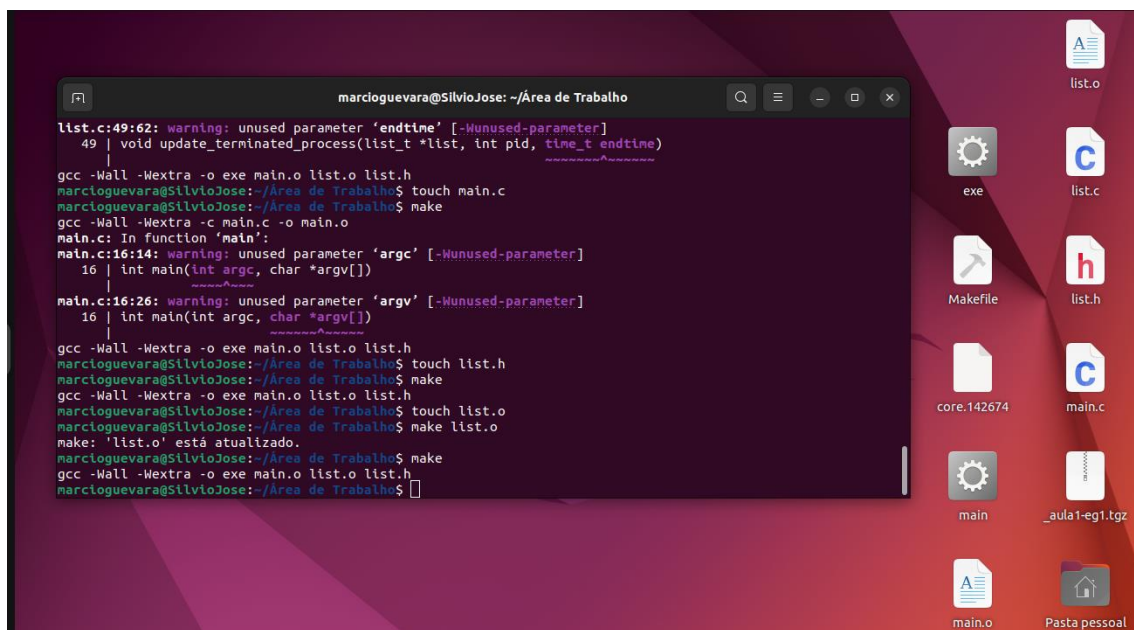
**R:** Ao alterar o arquivo list.h e depois executar o comando make, todos os arquivos foram recompilados porque o Makefile depende dos arquivos de cabeçalho (.h) para determinar quando os arquivos fonte (.c) precisam ser recompilados.

Normalmente, quando um arquivo de cabeçalho é modificado, isso afeta os arquivos de origem que incluem esse cabeçalho. O Makefile rastreia essas dependências e, quando um arquivo de cabeçalho é modificado, o Make determina que os arquivos de origem que incluem esse cabeçalho precisam ser recompilados. Isso garante que as alterações feitas nos arquivos de cabeçalho sejam refletidas adequadamente nos arquivos objeto e, finalmente, no executável.



e) Simule a alteração do ficheiro list.o. O que acontece quando faz make list.o? E se agora fizer make?

**R: Quando fazemos make list.o diz que o ficheiro esta actualizado  
Quando fazemos make lista ou gera todos os ficheiros.**



Retire a dependência do ficheiro list.h da regra list.o da Makefile. Repita

procedimento da alínea d). Explique a diferença no resultado?

**R: e a dependência do arquivo list.h for removida da regra list.o no Makefile, significa que o Makefile não reconhecerá alterações no arquivo list.h como um motivo para recompilar o arquivo list.o.**

**Isso quer dizer que, mesmo que você altere o arquivo list.h, o Makefile não reconhecerá essa alteração como uma razão para recompilar o arquivo objeto list.o. Isso pode levar a resultados imprevisíveis e a possibilidade de seu programa ficar desatualizado se as mudanças no arquivo list.h forem significativas e necessárias para a correta operação do seu código.**

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

Identifique: o alvo, as dependências e o comando. Tenha em atenção que os espaços iniciais em cada linha são tabs.

clean:

```
rm -f *.o main
```

h) Execute make clean. O que aconteceu? Porque razão o comando é executado sempre que esta regra é invocada explicitamente?

**R: Quando você executa o comando make clean, você está invocando explicitamente a regra clean definida no Makefile. A regra clean é uma regra especial que normalmente é usada para limpar o diretório de arquivos temporários e arquivos gerados durante o processo de compilação.**

**A regra clean geralmente contém comandos para excluir arquivos que não são necessários após a compilação, como arquivos objeto, executáveis e outros arquivos temporários.**