

# Computação Paralela e Distribuída

## Ano Lectivo 2024-25

### Laboratório - Aula 1

#### Introdução ao Ambiente Unix

#### Pré-requisitos para este laboratório:

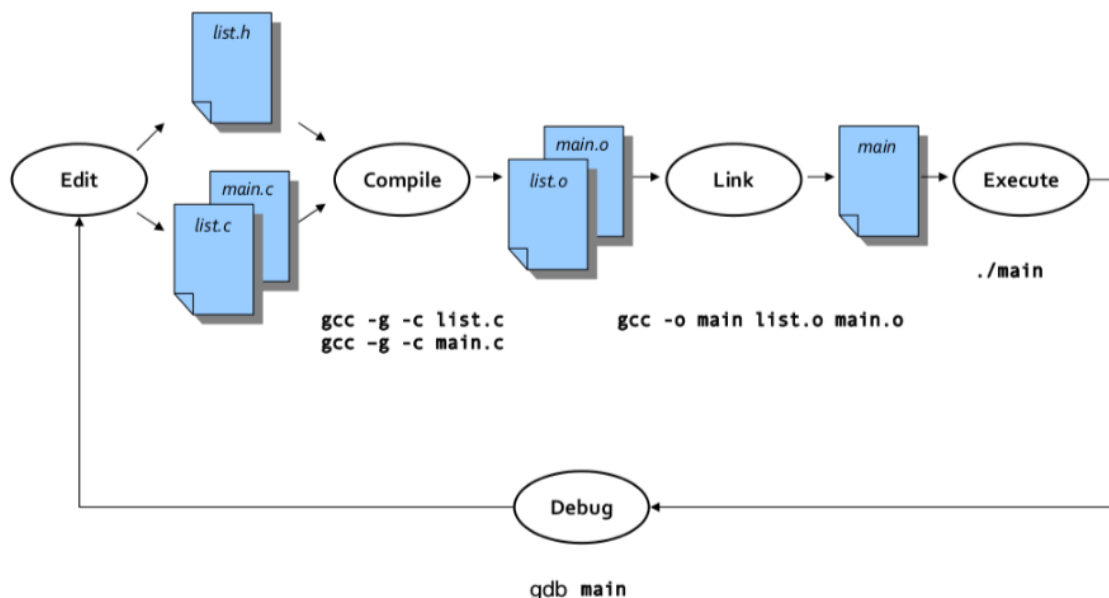
- Já ter instalado o Sistema Operativo Ubuntu e as ferramentas mínimas (gcc, gdb, make) em sua máquina
- Ter revisado as matérias relevantes de Algoritmo e Estrutura de Dados

#### 1. Objectivos

- Introduzir o desenvolvimento de aplicação em linguagem C no ambiente Unix
- Tornar o estudante familiarizado com as ferramentas necessárias (gcc, gdb, make e comandos Unix).

#### 2. Introdução

Perceba o ciclo de desenvolvimento de aplicações em linguagem C no ambiente Unix:



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
```

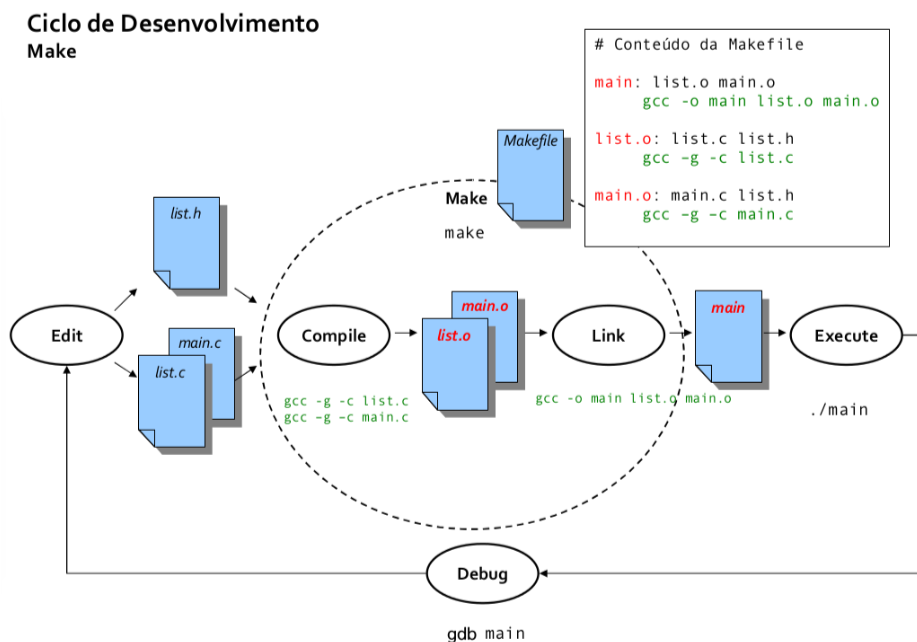
### 3. Geração do executável

- 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

### 4. Utilização do debugger gdb

- a) Execute a aplicação no debugger gdb:  
\$ **gdb** main
- 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?
- 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.
- j) 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?

- k) 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
```
- 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*.



#### 4. Utilização da ferramenta make

- Crie o ficheiro *Makefile* na sua área de trabalho e execute *make*. **O que aconteceu?**
- Apague o ficheiro *list.o*. Re-execute *make*. **Interprete o sucedido.**
- Simule uma alteração ao ficheiro *main.c* com o comando seguinte e re-execute *make*. **Compreenda o resultado.**  

```
$ touch main.c
```
- Simule a alteração do ficheiro *list.h* e execute *make*. **Porque razão todos os ficheiros foram gerados?**

- e) Simule a alteração do ficheiro *list.o*. O que acontece quando faz *make list.o*? E se agora fizer *make*?
- f) Retire a dependência do ficheiro *list.h* da regra *list.o* da *Makefile*. Repita procedimento da alínea d). Explique a diferença no resultado?
- 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?

#### 4. Outros exercícios

Para ser entregue dia 16 de Março de 2025, até às 16 horas.

1. Implemente a função `update_terminated_process`. A função `update_terminated_process` recebe uma lista, um valor de `pid` e um tempo de fim, procura pelo elemento com esse valor de `pid` e atualiza esse elemento com o tempo de fim.

#### 2. Gestor de tarefas pessoais

Construa uma aplicação de organização pessoal que permite gerir as tarefas pendentes de uma pessoa. Cada tarefa:

- é identificada por uma sequência de caracteres única que, por simplicidade, não pode conter espaços;
- tem uma prioridade, definida por um inteiro entre 0 e 5 (5 é mais prioritária, 0 é menos prioritária).

A aplicação tem os seguintes comandos:

- `$ new <prioridade> <id-nova-tarefa>`  
que insere a nova tarefa;
- `$ list <prioridade>`  
que lista todas as tarefas com tarefa da prioridade indicada ou superior; a listagem deve estar ordenada por prioridade (mais prioritárias primeiro) e, entre tarefas igualmente prioritárias, por data de criação (mais recentes primeiro);
- `$ complete <id-nova-tarefa>`  
que retira a tarefa indicada; caso a tarefa não exista, deve ser apresentada a mensagem de erro "TAREFA INEXISTENTE".

**Sugestão:** usar tantas listas quanto níveis de prioridade.