

# Criação de Processos

Ambientação Linux, GCC, Makefile e Processos  
Pai-Filho

# Sistema Operacional Linux

- Histórico → anos 1990
- Free Software Foundation
- Base do BSD (que era baseado no Unix)
- Características
  - Modularidade
  - Segurança
  - Ambiente gráfico (GUI)
  - Interface por linha de comando (bash)

# Ambiente Gráfico

- Interfaces (distribuições)
  - KDE, Gnome, XFCE, LXDE
  - <https://livrelinux.wordpress.com/2010/03/21/afinal-o-que-e-kde-gnome-xfce-lxde/>
  - Intuitivas

# Interface de linha de comando

- Interpretação de comandos
- Usado para execução de programas
- Vários programas (shells)
  - Bash (<https://www.gnu.org/software/bash/>)
  - Terminal (emulador de interpretador do GNOME) → permite eventos do mouse
- Para execução de comandos no terminal...
  - digitar comando + opções
  - pressionar 'ENTER'

# Interface de linha de comando

- Listar diretório (**ls**)
- Listar diretório com visualização de arquivos ocultos (**ls -a** ou **la**)
- Ir para diretório raiz (**cd /**)
- Listar diretório raiz → visualizar /home
- Ir para 'home' (**cd**)
- Ver usuário (**whoami**)
- Criar pasta (**mkdir**)
- Mudar para pasta (**cd "pasta"**)
- Ver manual para copiar, mover e apagar pastas e arquivos (**man "comando"**)
- Criar arquivo texto com "cat" (**cat > "nome\_arquivo"**)
- Mover para 'home' (**mv "arquivo" ..**)

# Interface de linha de comando

- Apagar arquivo em “home” (**rm** “arquivo”)
- Voltar para pasta (**cd** “pasta”)
- Abrir editor de texto (gedit)
- Escrever um “hello, world!” em C/C++

```
#include <stdio.h>
```

```
int main(void){  
    printf("\nHello, World!\n");  
    return 0;  
}
```

Para compilar...

```
gcc -o hello hello.c
```

Para executar...

```
./hello
```

# Executável com muitos arquivos fontes?

- Uso do “make”
- Arquivo texto → Makefile
- Referências

<http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>

<https://www.gnu.org/software/make/manual/make.html>

# Exemplo

## hellomake.c

```
#include <hellomake.h>

int main() {
    // call a function in another file
    myPrintHelloMake();

    return(0);
}
```

## hellofunc.c

```
#include <stdio.h>
#include <hellomake.h>

void myPrintHelloMake(void) {
    printf("Hello makefiles!\n");

    return;
}
```

## hellomake.h

```
/*
example include file
*/

void myPrintHelloMake(void);
```

gcc -o hellomake hellomake.c hellofunc.c -I.




# Exemplo

- Criar um arquivo texto chamado **Makefile**
- Conteúdo...

```
hellomake: hellomake.c hellofunc.c  
    gcc -o hellomake hellomake.c hellofunc.c -l.
```

TAB aqui!



# Exemplo

CC=gcc

CFLAGS=-I.

hellomake: hellomake.o hellofunc.o

gcc -o hellomake hellomake.o hellofunc.o

# Exemplo

CC=gcc

CFLAGS=-I.

DEPS = hellomake.h

%.o: %.c \$(DEPS)

\$(CC) -c -o \$@ \$< \$(CFLAGS)

hellomake: hellomake.o hellofunc.o

gcc -o hellomake hellomake.o hellofunc.o -I.

# Exemplo

CC=gcc

CFLAGS=-I.

DEPS = hellomake.h

OBJ = hellomake.o hellofunc.o

%.o: %.c \$(DEPS)

\$(CC) -c -o \$@ \$< \$(CFLAGS)

hellomake: \$(OBJ)

gcc -o \$@ \$^ \$(CFLAGS)

# E processo pai/filho?

- Referência: <http://www.dca.ufrn.br/~adelardo/cursos/DCA409/all.html>
- `fork()`, `wait()`, `exit()`
- Fazer código para pai criar filho e ambos exibirem seus PIDs (usar `getpid()`)
- Lembrar da estrutura básica...

# Exemplo

```
#include <stdio.h>
#include <unistd.h>
```

```
int main()
{
    int pid ;
```

```
    pid=fork() ;
```

```
    if(pid== -1) /* erro */
```

```
    {
```

```
        printf("\nimpossivel de criar um filho\n") ;
```

```
        exit(-1) ;
```

```
    }
```

```
    else if(pid==0) /* filho */
```

```
    {
```

```
        // CÓDIGO DO FILHO
```

```
        exit(1) ;
```

```
    }
```

```
    else /* pai */
```

```
    {
```

```
        // CÓDIGO DO PAI
```

```
    }
```

```
    exit(0);
```

```
}
```

teste de erro na criação do processo! Tanto o pai quanto o filho executam!

código do processo filho

código do processo pai

# Exemplo

Fazer programa para pai receber um número inteiro, realizar os devidos testes (menor que zero?), e criar um filho para gerar e exibir a uma quantidade de termos da Série de Fibonacci igual ao número lido!

Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, ...

Algoritmo:  $fib_0 = 0$   
 $fib_1 = 1$   
 $fib_n = fib_{n-1} + fib_{n-2}$