

My Project

Generated by Doxygen 1.8.17

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 call_by_string.c File Reference	3
2.1.1 Function Documentation	3
2.1.1.1 main()	4
2.2 call_by_string.c	4
2.3 ficha2_2.c File Reference	5
2.3.1 Function Documentation	5
2.3.1.1 main()	5
2.4 ficha2_2.c	6
2.5 ficha2_3.c File Reference	6
2.5.1 Function Documentation	6
2.5.1.1 main()	6
2.6 ficha2_3.c	7
2.7 ficha2_4.c File Reference	7
2.7.1 Function Documentation	8
2.7.1.1 main()	8
2.8 ficha2_4.c	8
2.9 ficha2_5.c File Reference	8
2.9.1 Function Documentation	9
2.9.1.1 main()	9
2.10 ficha2_5.c	10
2.11 ficha3_1.c File Reference	10
2.11.1 Function Documentation	10
2.11.1.1 main()	11
2.12 ficha3_1.c	11
2.13 ficha3_2.c File Reference	11
2.13.1 Function Documentation	11
2.13.1.1 main()	12
2.14 ficha3_2.c	12
2.15 ficha3_3.c File Reference	12
2.15.1 Function Documentation	13
2.15.1.1 main()	13
2.16 ficha3_3.c	13
2.17 ficha3_4.c File Reference	14
2.17.1 Function Documentation	14
2.17.1.1 main()	14
2.18 ficha3_4.c	14
2.19 ficha3_5.c File Reference	15
2.19.1 Function Documentation	15

2.19.1.1 main()	15
2.20 ficha3_5.c	16
2.21 ficha3_6.c File Reference	16
2.21.1 Function Documentation	16
2.21.1.1 main()	17
2.22 ficha3_6.c	17
2.23 ficha4_2.c File Reference	17
2.23.1 Function Documentation	18
2.23.1.1 main()	18
2.24 ficha4_2.c	18
2.25 ficha4_2_copy.c File Reference	19
2.25.1 Function Documentation	19
2.25.1.1 main()	19
2.26 ficha4_2_copy.c	20
2.27 ficha5_3.c File Reference	20
2.27.1 Macro Definition Documentation	21
2.27.1.1 MAX_LINE_SIZE	21
2.27.2 Function Documentation	21
2.27.2.1 main()	22
2.27.2.2 readln()	22
2.28 ficha5_3.c	23
2.29 io.c File Reference	23
2.29.1 Function Documentation	24
2.29.1.1 convert_argv_to_v()	24
2.29.1.2 readln()	25
2.30 io.c	25
2.31 linux.h File Reference	25
2.32 linux.h	26
2.33 pid.c File Reference	26
2.33.1 Function Documentation	27
2.33.1.1 print_child_report()	27
2.33.1.2 print_father()	27
2.33.1.3 print_father_report()	27
2.33.1.4 print_pid()	27
2.34 pid.c	28
2.35 pid.h File Reference	28
2.35.1 Function Documentation	28
2.35.1.1 print_child_report()	29
2.35.1.2 print_father()	29
2.35.1.3 print_father_report()	29
2.35.1.4 print_pid()	29
2.36 pid.h	29

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

call_by_string.c	3
ficha2_2.c	5
ficha2_3.c	6
ficha2_4.c	7
ficha2_5.c	8
ficha3_1.c	10
ficha3_2.c	11
ficha3_3.c	12
ficha3_4.c	14
ficha3_5.c	15
ficha3_6.c	16
ficha4_2.c	17
ficha4_2_copy.c	19
ficha5_3.c	20
io.c	23
linux.h	25
pid.c	26
pid.h	28

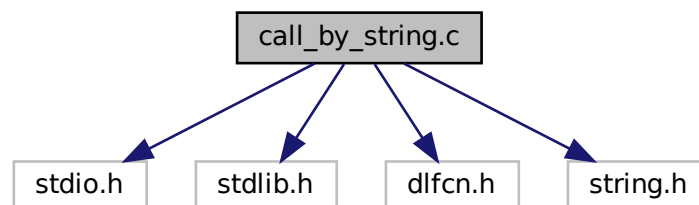
Chapter 2

File Documentation

2.1 call_by_string.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
#include <string.h>
```

Include dependency graph for call_by_string.c:



Functions

- int `main` (int argc, char *argv[])

2.1.1 Function Documentation

2.1.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 6 of file [call_by_string.c](#).

```
00007 {
00008     void *handle;
00009     char *error;
00010     double (*cosine) (double);
00011     int (*func) (int, char*[]);
00012
00013     char command[1024];
00014     handle = dlopen ("/home/alef/Documents/coding/c/so/cmake-build-debug/libfichas.so", RTLD_LAZY);
00015     if (!handle)
00016     {
00017         fprintf (stderr, "%s\n", dlerror ());
00018         exit (EXIT_FAILURE);
00019     }
00020
00021     dlerror ();    /* Clear any existing error */
00022
00023     /* Writing: cosine = (double (*)(double)) dlsym(handle, "cos");
00024     would seem more natural, but the C99 standard leaves
00025     casting from "void *" to a function pointer undefined.
00026     The assignment used below is the POSIX.1-2003 (Technical
00027     Corrigendum 1) workaround; see the Rationale for the
00028     POSIX specification of dlsym(). */
00029
00030     /*(void **)&cosine = dlsym (handle, "cos");
00031     *(void **)&func = dlsym (handle, argv[1]);
00032     if ((error = dlerror ()) != NULL)
00033     {
00034         fprintf (stderr, "%s\n", error);
00035         exit (EXIT_FAILURE);
00036     }
00037
00038     (*func) (argc, argv);
00039
00040     dlclose (handle);
00041     exit (EXIT_SUCCESS);
00042 }
```

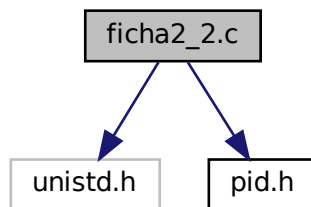
2.2 call_by_string.c

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <dlfcn.h>
00004 #include <string.h>
00005
00006 int main (int argc, char *argv[])
00007 {
00008     void *handle;
00009     char *error;
00010     double (*cosine) (double);
00011     int (*func) (int, char*[]);
00012
00013     char command[1024];
00014     handle = dlopen ("/home/alef/Documents/coding/c/so/cmake-build-debug/libfichas.so", RTLD_LAZY);
00015     if (!handle)
00016     {
00017         fprintf (stderr, "%s\n", dlerror ());
00018         exit (EXIT_FAILURE);
00019     }
00020
00021     dlerror ();    /* Clear any existing error */
00022
00023     /* Writing: cosine = (double (*)(double)) dlsym(handle, "cos");
00024     would seem more natural, but the C99 standard leaves
00025     casting from "void *" to a function pointer undefined.
00026     The assignment used below is the POSIX.1-2003 (Technical
00027     Corrigendum 1) workaround; see the Rationale for the
00028     POSIX specification of dlsym(). */
00029
00030     /*(void **)&cosine = dlsym (handle, "cos");
00031     *(void **)&func = dlsym (handle, argv[1]);
00032     if ((error = dlerror ()) != NULL)
00033     {
```

```
00034     fprintf (stderr, "%s\n", error);
00035     exit (EXIT_FAILURE);
00036 }
00037
00038 (*func) (argc, argv);
00039
00040 dlclose (handle);
00041 exit (EXIT_SUCCESS);
00042 }
00043
```

2.3 ficha2_2.c File Reference

```
#include <unistd.h>
#include "pid.h"
Include dependency graph for ficha2_2.c:
```



Functions

- int [main](#) (void)

2.3.1 Function Documentation

2.3.1.1 main()

```
int main (
    void )
```

Definition at line 4 of file [ficha2_2.c](#).

```
00004     {
00005     fork();
00006     print_child_report ();
00007     return 0;
00008 }
```

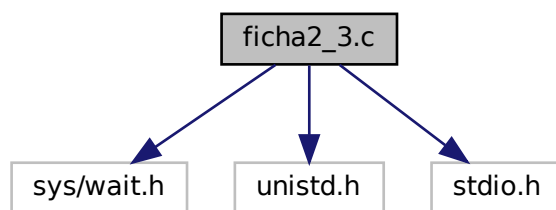
References [print_child_report\(\)](#).

2.4 ficha2_2.c

```
00001 #include <unistd.h>
00002 #include "pid.h"
00003
00004 int main(void) {
00005     fork();
00006     print_child_report ();
00007     return 0;
00008 }
```

2.5 ficha2_3.c File Reference

```
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
Include dependency graph for ficha2_3.c:
```



Functions

- int `main` (void)

Programa que cria dez processos filhos que executam sequencialmente.

2.5.1 Function Documentation

2.5.1.1 main()

```
int main (
    void )
```

Programa que cria dez processos filhos que executam sequencialmente.

- Os filhos imprimem o seu PID e o do seu pai, e finalmente, terminam a sua execução.
- O valor de saída de cada filho é igual ao seu numero de ordem (e.g.: primeiro filho criado termina com o valor 1).

- O pai imprime o código de saída de cada um dos seus filhos.

Definition at line 13 of file `ficha2_3.c`.

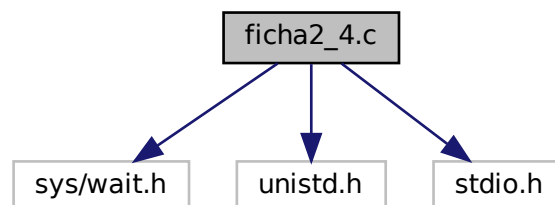
```
00013     {
00014         int status;
00015         int nFilhos = 10;
00016         for (int i = 0; i < nFilhos; i++) {
00017             if (fork() == 0) {// Child does something
00018                 fprintf(stderr, "My father's PID: %d\nMy PID: %d\n\n", getppid(), getpid());
00019                 _exit(i);
00020             } else {// Parent does something
00021                 pid_t wait_ret = wait(&status);
00022                 fprintf(stderr, "Filho %d saiu com código %d\n\n", wait_ret, WEXITSTATUS(status));
00023             }
00024         }
00025         return 0;
00026     }
```

2.6 ficha2_3.c

```
00001 #include <sys/wait.h> // for wait()
00002 #include <unistd.h> // for fork()
00003 #include <stdio.h> // for fprintf()
00004
00013 int main(void) {
00014     int status;
00015     int nFilhos = 10;
00016     for (int i = 0; i < nFilhos; i++) {
00017         if (fork() == 0) {// Child does something
00018             fprintf(stderr, "My father's PID: %d\nMy PID: %d\n\n", getppid(), getpid());
00019             _exit(i);
00020         } else {// Parent does something
00021             pid_t wait_ret = wait(&status);
00022             fprintf(stderr, "Filho %d saiu com código %d\n\n", wait_ret, WEXITSTATUS(status));
00023         }
00024     }
00025     return 0;
00026 }
```

2.7 ficha2_4.c File Reference

```
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
Include dependency graph for ficha2_4.c:
```



Functions

- `int main` (void)
Cria dez processos filhos em concorrência.

2.7.1 Function Documentation

2.7.1.1 main()

```
int main (
    void )
```

Cria dez processos filhos em concorrência.

O pai espera pelo fim da execução de todos os filhos, imprimindo os respectivos códigos de saída.

Definition at line 11 of file [ficha2_4.c](#).

```
00011 {
00012     int status;
00013     int nFilhos = 10;
00014     for (int i = 0; i < nFilhos; i++) { // Child does something
00015         if (fork() == 0) {
00016             fprintf(stderr, "My father's PID: %d\nMy PID: %d\n\n", getppid(), getpid());
00017             _exit(i);
00018         }
00019     }
00020     /* Parent waits each child and prints than immediately */
00021     for (int _ = 0; _ < nFilhos; _++) { //Parent does something
00022         pid_t wait_ret = wait(&status);
00023         fprintf(stderr, "Filho %d acabou com código %d\n\n", wait_ret, WEXITSTATUS(status));
00024     }
00025     return 0;
00026 }
```

2.8 ficha2_4.c

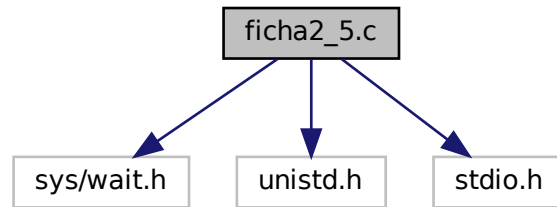
```
00001 #include <sys/wait.h> // for wait()
00002 #include <unistd.h> // for fork()
00003 #include <stdio.h> // for fprintf()
00004
00011 int main(void) {
00012     int status;
00013     int nFilhos = 10;
00014     for (int i = 0; i < nFilhos; i++) { // Child does something
00015         if (fork() == 0) {
00016             fprintf(stderr, "My father's PID: %d\nMy PID: %d\n\n", getppid(), getpid());
00017             _exit(i);
00018         }
00019     }
00020     /* Parent waits each child and prints than immediately */
00021     for (int _ = 0; _ < nFilhos; _++) { //Parent does something
00022         pid_t wait_ret = wait(&status);
00023         fprintf(stderr, "Filho %d acabou com código %d\n\n", wait_ret, WEXITSTATUS(status));
00024     }
00025     return 0;
00026 }
```

2.9 ficha2_5.c File Reference

```
#include <sys/wait.h>
#include <unistd.h>
```

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

Include dependency graph for ficha2_5.c:



Functions

- int `main()`

Cria (dez) filhos em profundidade.

2.9.1 Function Documentation

2.9.1.1 main()

```
int main ( )
```

Cria (dez) filhos em profundidade.

Cada processo imprime seu PID e o PID de seu pai.

O programa vai até o 10º nível de profundidade.

Definition at line 12 of file `ficha2_5.c`.

```

00012     {
00013         int status;
00014         int nFilhos = 10;
00015         /* The starting process. */
00016         fprintf(stderr, "My father's PID: %d\nMy PID: %d\nIteration: %d\n\n", getppid(), getpid(), -1);
00017         /* Creation of nFilhos process. */
00018         for (int i = 0; i < nFilhos; i++) {
00019             if (fork() == 0) { // Child does something
00020                 fprintf(stderr, "My father's PID: %d\nMy PID: %d\nIteration: %d\n\n", getppid(), getpid(),
00021                     i);
00022             } else { // Parent does something
00023                 pid_t terminated = wait(&status);
00024                 fprintf(stderr, "[Pai (my PID is %d)] process %d. Exit with code %d\nIteration: %d\n\n",
00025                     getpid(), terminated, WEXITSTATUS(status), i);
00026                 _exit(0);
00027             }
00028         }
00029         return 0;
00030     }
  
```

2.10 ficha2_5.c

```

00001 #include <sys/wait.h> // for wait()
00002 #include <unistd.h> // for fork()
00003 #include <stdio.h> // for fprintf()
00004
00012 int main() {
00013     int status;
00014     int nFilhos = 10;
00015     /* The starting process. */
00016     fprintf(stderr, "My father's PID: %d\nMy PID: %d\nIteration: %d\n\n", getppid(), getpid(), -1);
00017     /* Creation of nFilhos process. */
00018     for (int i = 0; i < nFilhos; i++) {
00019         if (fork() == 0) { // Child does something
00020             fprintf(stderr, "My father's PID: %d\nMy PID: %d\nIteration: %d\n\n", getppid(), getpid(),
i);
00021         } else { // Parent does something
00022             pid_t terminated = wait(&status);
00023             fprintf(stderr, "[Pai (my PID is %d)] process %d. Exit with code %d\nIteration: %d\n\n",
getpid(), terminated, WEXITSTATUS(status), i);
00024             _exit(0);
00025         }
00026     }
00027 }
00028 return 0;
00029 }

```

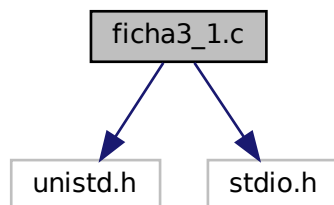
2.11 ficha3_1.c File Reference

```

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

```

Include dependency graph for ficha3_1.c:



Functions

- int [main](#) (void)

2.11.1 Function Documentation

2.11.1.1 main()

```
int main (  
    void )
```

Definition at line 4 of file [ficha3_1.c](#).

```
00004      {  
00005  int ret;  
00006  //Como ls é uma variável de ambiente, não é preciso dar o caminho completo  
00007  ret = execl("/bin/ls", "ls", "-l", "-a", NULL);  
00008  printf("Se tudo ocorreu bem, esse printf nunca acontece\n");  
00009  perror("Erro");  
00010  return ret;  
00011 }
```

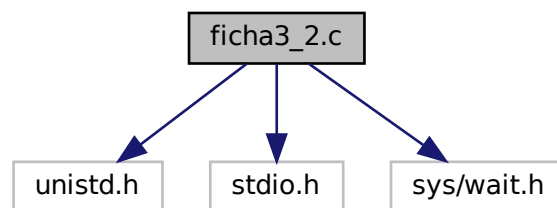
2.12 ficha3_1.c

```
00001 #include <unistd.h>  
00002 #include <stdio.h>  
00003  
00004 int main(void){  
00005  int ret;  
00006  //Como ls é uma variável de ambiente, não é preciso dar o caminho completo  
00007  ret = execl("/bin/ls", "ls", "-l", "-a", NULL);  
00008  printf("Se tudo ocorreu bem, esse printf nunca acontece\n");  
00009  perror("Erro");  
00010  return ret;  
00011 }
```

2.13 ficha3_2.c File Reference

```
#include <unistd.h>  
#include <stdio.h>  
#include <sys/wait.h>
```

Include dependency graph for ficha3_2.c:



Functions

- int [main](#) (void)

2.13.1 Function Documentation

2.13.1.1 main()

```
int main (
    void )
```

Definition at line 4 of file [ficha3_2.c](#).

```
00005 {
00006     int status;
00007     char *exec_args[] = {"", "-l", NULL};
00008     pid_t fork_ret = fork ();
00009     if (fork_ret == 0)
00010     {
00011         fprintf (stderr, "PID do filho %d\n", getpid ());
00012         int exec_ret = execvp ("/bin/ls", exec_args);
00013         fprintf (stderr, "Se tudo ocorreu bem, esse printf nunca acontece\n");
00014         perror ("Erro de exec");
00015         _exit (exec_ret);
00016     }
00017     else
00018     {
00019         fprintf (stderr, "PID do pai %d\n", getpid ());
00020         int wait_ret = wait (&status);
00021         fprintf (stderr, "O filho retornou %d\n", WEXITSTATUS(status));
00022     }
00023     return 0;
00024 }
```

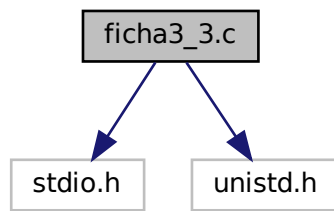
2.14 ficha3_2.c

```
00001 #include <unistd.h>
00002 #include <stdio.h>
00003 #include <sys/wait.h>
00004 int main (void)
00005 {
00006     int status;
00007     char *exec_args[] = {"", "-l", NULL};
00008     pid_t fork_ret = fork ();
00009     if (fork_ret == 0)
00010     {
00011         fprintf (stderr, "PID do filho %d\n", getpid ());
00012         int exec_ret = execvp ("/bin/ls", exec_args);
00013         fprintf (stderr, "Se tudo ocorreu bem, esse printf nunca acontece\n");
00014         perror ("Erro de exec");
00015         _exit (exec_ret);
00016     }
00017     else
00018     {
00019         fprintf (stderr, "PID do pai %d\n", getpid ());
00020         int wait_ret = wait (&status);
00021         fprintf (stderr, "O filho retornou %d\n", WEXITSTATUS(status));
00022     }
00023     return 0;
00024 }
```

2.15 ficha3_3.c File Reference

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

Include dependency graph for ficha3_3.c:



Functions

- int `main` (int argc, char *argv[])

2.15.1 Function Documentation

2.15.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 4 of file [ficha3_3.c](#).

```
00005 {
00006     for (int i = 1; i < argc; i++)
00007     {
00008         for (int j = 0; argv[i][j] != 0; j++)
00009             write (STDOUT_FILENO, &argv[i][j], 1);
00010         if (i < argc - 1)
00011             write (STDOUT_FILENO, " ", 1);
00012     }
00013     write (STDOUT_FILENO, "\n", 1);
00014     return 0;
00015 }
```

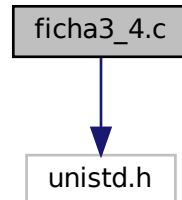
2.16 ficha3_3.c

```
00001 #include <stdio.h>
00002 #include <unistd.h>
00003
00004 int main (int argc, char *argv[])
00005 {
00006     for (int i = 1; i < argc; i++)
00007     {
00008         for (int j = 0; argv[i][j] != 0; j++)
00009             write (STDOUT_FILENO, &argv[i][j], 1);
00010         if (i < argc - 1)
00011             write (STDOUT_FILENO, " ", 1);
00012     }
00013     write (STDOUT_FILENO, "\n", 1);
00014     return 0;
00015 }
```

2.17 ficha3_4.c File Reference

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

Include dependency graph for ficha3_4.c:



Functions

- int `main` (int argc, char *argv[])

2.17.1 Function Documentation

2.17.1.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 3 of file `ficha3_4.c`.

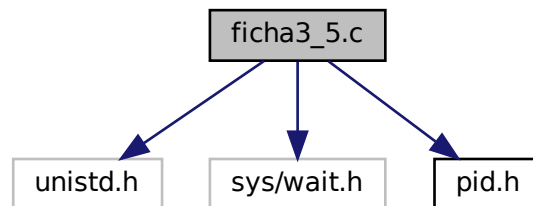
```
00003 {  
00004     char *path = "/home/alef/Documents/coding/c/so/cmake-build-debug/./ficha3_3";  
00005     execvp (path, argv);  
00006     return 0;  
00007 }
```

2.18 ficha3_4.c

```
00001 #include <unistd.h>  
00002  
00003 int main (int argc, char *argv[]){  
00004     char *path = "/home/alef/Documents/coding/c/so/cmake-build-debug/./ficha3_3";  
00005     execvp (path, argv);  
00006     return 0;  
00007 }
```

2.19 ficha3_5.c File Reference

```
#include <unistd.h>
#include <sys/wait.h>
#include "pid.h"
Include dependency graph for ficha3_5.c:
```



Functions

- int `main` (int argc, char *argv[])

2.19.1 Function Documentation

2.19.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 8 of file `ficha3_5.c`.

```
00009 {
00010     int status;
00011     for (int i = 1; i < argc; i++)
00012     {
00013         if (fork () == 0)
00014         {
00015             print_child_report();
00016             execlp (argv[i], argv[i], NULL);
00017             _exit (0);
00018         }
00019     }
00020     for (int i = 1; i < argc; i++)
00021     {
00022         wait (&status);
00023     }
00024     return 0;
00025 }
```

References `print_child_report()`.

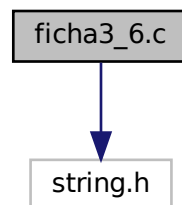
2.20 ficha3_5.c

```
00001 #include <unistd.h>
00002 #include <sys/wait.h>
00003 #include "pid.h"
00004 // Implemente um programa que execute concorrentemente uma lista de executaveis especificados como
00005 // argumentos da linha de comando. Considere os executaveis sem quaisquer argumentos próprios.
00006 // O programa devera esperar pelo fim da execução de todos processos por si criados.
00007
00008 int main (int argc, char *argv[])
00009 {
00010     int status;
00011     for (int i = 1; i < argc; i++)
00012     {
00013         if (fork () == 0)
00014         {
00015             print_child_report();
00016             execlp (argv[i], argv[i], NULL);
00017             _exit (0);
00018         }
00019     }
00020     for (int i = 1; i < argc; i++)
00021     {
00022         wait (&status);
00023     }
00024     return 0;
00025 }
```

2.21 ficha3_6.c File Reference

```
#include <string.h>
```

Include dependency graph for ficha3_6.c:



Functions

- int `main` (char *command[])

2.21.1 Function Documentation

2.21.1.1 main()

```
int main (
    char * command[ ] )
```

Definition at line 4 of file [ficha3_6.c](#).

```
00004 {
00005     int fork_ret, exec_ret, wait_ret, status, res;
00006
00007     char *exec_args[1024];
00008     char*string;
00009     int i = 0;
00010
00011     string = strtok(command, " ");
00012     while (string != NULL){
00013         exec_args[i] = string;
00014     }
00015 }
```

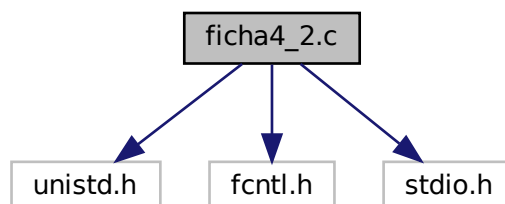
2.22 ficha3_6.c

```
00001 #include <string.h>
00002 // tirei foto, checar screenshot
00003
00004 int main(char* command[]){
00005     int fork_ret, exec_ret, wait_ret, status, res;
00006
00007     char *exec_args[1024];
00008     char*string;
00009     int i = 0;
00010
00011     string = strtok(command, " ");
00012     while (string != NULL){
00013         exec_args[i] = string;
00014     }
00015 }
```

2.23 ficha4_2.c File Reference

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

Include dependency graph for `ficha4_2.c`:



Functions

- int [main](#) ()

2.23.1 Function Documentation

2.23.1.1 main()

```
int main ( )
```

Definition at line 5 of file [ficha4_2.c](#).

```
00006 {
00007     int i_fd = open ("/etc/passwd", O_RDONLY);
00008     int o_fd = open ("saida.txt", O_CREAT | O_RDWR, 0666);
00009     int e_fd = open ("erros.txt", O_CREAT | O_RDWR, 0666);
00010
00011     dup2 (i_fd, STDIN_FILENO);
00012     dup2 (o_fd, STDOUT_FILENO);
00013     dup2 (e_fd, STDERR_FILENO);
00014
00015     close (i_fd);
00016     close (e_fd);
00017     close (o_fd);
00018
00019     int n_lines = 10;
00020     char buffer;
00021     char line[1024];
00022     int i = 0;
00023     while (n_lines > 0)
00024     {
00025         while (read (0, &buffer, 1) != 0)
00026         {
00027             line[i] = buffer;
00028             i++;
00029             if (buffer == '\n')
00030             {
00031                 break;
00032             }
00033         }
00034         write (1, line, i);
00035         write (2, line, i);
00036         i = 0;
00037         n_lines--;
00038     }
00039     return 0;
00040 }
```

2.24 ficha4_2.c

```
00001 #include <unistd.h>
00002 #include <fcntl.h>
00003 #include <stdio.h>
00004
00005 int main ()
00006 {
00007     int i_fd = open ("/etc/passwd", O_RDONLY);
00008     int o_fd = open ("saida.txt", O_CREAT | O_RDWR, 0666);
00009     int e_fd = open ("erros.txt", O_CREAT | O_RDWR, 0666);
00010
00011     dup2 (i_fd, STDIN_FILENO);
00012     dup2 (o_fd, STDOUT_FILENO);
00013     dup2 (e_fd, STDERR_FILENO);
00014
00015     close (i_fd);
00016     close (e_fd);
00017     close (o_fd);
00018
00019     int n_lines = 10;
00020     char buffer;
00021     char line[1024];
00022     int i = 0;
00023     while (n_lines > 0)
00024     {
00025         while (read (0, &buffer, 1) != 0)
00026         {
00027             line[i] = buffer;
00028             i++;
00029             if (buffer == '\n')
```



```

00030         {
00031             break;
00032         }
00033     }
00034     write (1, line, i);
00035     write (2, line, i);
00036     i = 0;
00037     n_lines--;
00038 }
00039 return 0;
00040 }

```

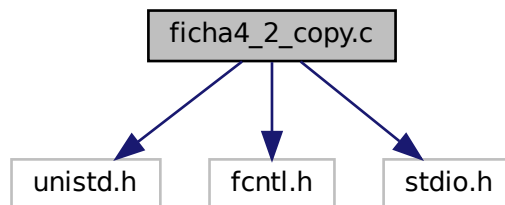
2.25 ficha4_2_copy.c File Reference

```

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

```

Include dependency graph for ficha4_2_copy.c:



Functions

- int [main](#) ()

2.25.1 Function Documentation

2.25.1.1 main()

```
int main ( )
```

Definition at line 5 of file [ficha4_2_copy.c](#).

```

00006 {
00007     printf("%u\n", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00008     printf("%u\n", S_IRUSR);
00009     printf("%u\n", S_IWUSR );
00010     printf("%u\n", S_IRGRP );
00011     printf("%u\n", S_IWGRP );
00012     printf ("%u\n", S_IROTH);
00013     int i_fd = open ("/etc/passwd", O_RDONLY);
00014     int o_fd = open ("saida.txt", O_CREAT | O_RDWR, 436);
00015     int e_fd = open ("erros.txt", O_CREAT | O_RDWR, 958);
00016     // 400 ^ 200 ^ ((400 ^ 200) » 3) ^ (400 »3 »3)

```

```

00017 // 666 = 436 = 958
00018 // -rw-rw-r--
00019 //int e_fd = open ("erros.txt", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00020 //printf("%d\n", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00021 dup2 (i_fd, STDIN_FILENO);
00022 dup2 (o_fd, STDOUT_FILENO);
00023 dup2 (e_fd, STDERR_FILENO);
00024
00025 close (i_fd);
00026 close (e_fd);
00027 close (o_fd);
00028
00029 int n_lines = 10;
00030 char c;
00031 for (int i = 0; i < n_lines;)
00032 {
00033     c = (char)getchar ();
00034     if (c == '\n')
00035         i++;
00036     putchar(c);
00037 }
00038 return 0;
00039 }

```

2.26 ficha4_2_copy.c

```

00001 #include <unistd.h>
00002 #include <fcntl.h>
00003 #include <stdio.h>
00004
00005 int main ()
00006 {
00007     printf("%u\n", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00008     printf("%u\n", S_IRUSR);
00009     printf("%u\n", S_IWUSR );
00010     printf("%u\n", S_IRGRP );
00011     printf("%u\n", S_IWGRP );
00012     printf ("%u\n", S_IROTH);
00013     int i_fd = open ("/etc/passwd", O_RDONLY);
00014     int o_fd = open ("saida.txt", O_CREAT | O_RDWR, 436);
00015     int e_fd = open ("erros.txt", O_CREAT | O_RDWR, 958);
00016     // 400 ^ 200 ^ ((400 ^ 200) » 3) ^ (400 »3 »3)
00017     // 666 = 436 = 958
00018     // -rw-rw-r--
00019     //int e_fd = open ("erros.txt", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00020     //printf("%d\n", S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH);
00021     dup2 (i_fd, STDIN_FILENO);
00022     dup2 (o_fd, STDOUT_FILENO);
00023     dup2 (e_fd, STDERR_FILENO);
00024
00025     close (i_fd);
00026     close (e_fd);
00027     close (o_fd);
00028
00029     int n_lines = 10;
00030     char c;
00031     for (int i = 0; i < n_lines;)
00032     {
00033         c = (char)getchar ();
00034         if (c == '\n')
00035             i++;
00036         putchar(c);
00037     }
00038     return 0;
00039 }

```

2.27 ficha5_3.c File Reference

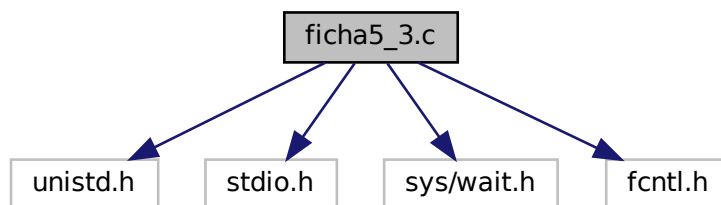
```

#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>

```

```
#include <fcntl.h>
```

Include dependency graph for ficha5_3.c:



Macros

- `#define` [MAX_LINE_SIZE](#) 1024

Functions

- `ssize_t` [readln](#) (int fildes, void *buf, `size_t` nbyte)
- `int` [main](#) (int argc, char *argv[])

2.27.1 Macro Definition Documentation

2.27.1.1 MAX_LINE_SIZE

```
#define MAX_LINE_SIZE 1024
```

Definition at line 6 of file [ficha5_3.c](#).

2.27.2 Function Documentation

2.27.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Definition at line 24 of file [ficha5_3.c](#).

```
00024                                     {
00025
00026     ssize_t res = 0;
00027     int p[2];
00028     char buffer[MAX_LINE_SIZE];
00029     int pid;
00030     int status;
00031
00032     if (pipe(p) != 0) {
00033         perror("pipe");
00034         return -1;
00035     }
00036
00037     switch ((pid = fork())) {
00038         case -1: perror("fork");
00039             return -1;
00040         case 0: close(p[1]);
00041             dup2(p[0], 0);
00042             close(p[0]);
00043             res = execlp("wc", "wc", NULL);
00044             _exit(0);
00045
00046         default: close(p[0]);
00047             close(1);
00048
00049             while ((res = readln(0, buffer, MAX_LINE_SIZE)) > 0) {
00050                 write(p[1], buffer, res);
00051             }
00052             close(p[1]);
00053
00054             if (wait(&status) < 0) {
00055                 perror("wait");
00056                 return -1;
00057             }
00058             if (status < 0) {
00059                 perror("Filho");
00060             }
00061     }
00062     return 0;
00063 }
```

References [MAX_LINE_SIZE](#), and [readln\(\)](#).

2.27.2.2 readln()

```
ssize_t readln (
    int fildes,
    void * buf,
    size_t nbyte )
```

Definition at line 8 of file [ficha5_3.c](#).

```
00008                                     {
00009
00010     ssize_t res = 0;
00011     int i = 0;
00012
00013     while (i < nbyte && (res = read(fildes, &buf[i], 1)) > 0) {
00014
00015         if (((char *)buf)[i] == '\n') {
00016             return i + 1;
00017         }
00018         i += res;
00019     }
00020
00021     return i;
00022 }
```

Referenced by [main\(\)](#).

2.28 ficha5_3.c

```

00001 #include <unistd.h>
00002 #include <stdio.h>
00003 #include <sys/wait.h>
00004 #include <fcntl.h>
00005
00006 #define MAX_LINE_SIZE 1024
00007
00008 ssize_t readln(int fildes, void *buf, size_t nbyte) {
00009     ssize_t res = 0;
00010     int i = 0;
00011
00012     while (i < nbyte && (res = read(fildes, &buf[i], 1)) > 0) {
00013
00014         if (((char *)buf)[i] == '\n') {
00015             return i + 1;
00016         }
00017         i += res;
00018     }
00019     return i;
00020 }
00021
00022 int main(int argc, char *argv[]) {
00023     ssize_t res = 0;
00024     int p[2];
00025     char buffer[MAX_LINE_SIZE];
00026     int pid;
00027     int status;
00028
00029     if (pipe(p) != 0) {
00030         perror("pipe");
00031         return -1;
00032     }
00033
00034     switch ((pid = fork())) {
00035         case -1: perror("fork");
00036                 return -1;
00037         case 0: close(p[1]);
00038                 dup2(p[0], 0);
00039                 close(p[0]);
00040                 res = execlp("wc", "wc", NULL);
00041                 _exit(0);
00042         default: close(p[0]);
00043                 close(1);
00044
00045                 while ((res = readln(0, buffer, MAX_LINE_SIZE)) > 0) {
00046                     write(p[1], buffer, res);
00047                 }
00048                 close(p[1]);
00049
00050                 if (wait(&status) < 0) {
00051                     perror("wait");
00052                     return -1;
00053                 }
00054                 if (status < 0) {
00055                     perror("Filho");
00056                 }
00057     }
00058     return 0;
00059 }

```

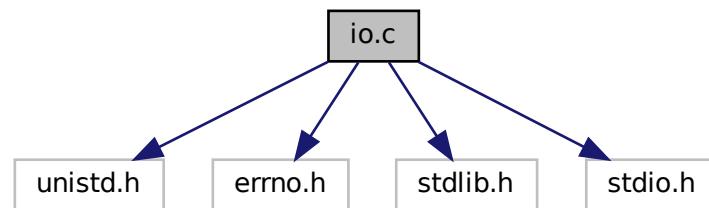
2.29 io.c File Reference

```

#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>

```

Include dependency graph for `io.c`:



Functions

- `char * readln (int fd, ssize_t max)`
- `char * convert_argv_to_v (int argc, char *argv[], int max)`

2.29.1 Function Documentation

2.29.1.1 `convert_argv_to_v()`

```
char* convert_argv_to_v (  
    int argc,  
    char * argv[],  
    int max )
```

Definition at line 21 of file `io.c`.

```
00022 {  
00023     char static new[2048];  
00024     int c = 0;  
00025     for (int i = 0; i < max; i++)  
00026     {  
00027         for (int j = 0; argv[i][j] != 0; j++)  
00028             new[c++] = argv[i][j];  
00029         if (i < argc - 1)  
00030             new[c++] = ' ';  
00031     }  
00032     new[c] = 0;  
00033     return new;  
00034 }
```

2.29.1.2 readln()

```
char* readln (
    int fd,
    ssize_t max )
```

Definition at line 6 of file [io.c](#).

```
00006                                     {
00007     char b;
00008     char static new[2048];
00009     unsigned long c = 0;
00010     long l = 0;
00011     long r;
00012     while ((r = read(fd,&b,1)) == -1 && errno==EINTR){
00013         new[c++] = b;
00014         if (b == '\n'){
00015             break;
00016         }
00017     }
00018     return realloc(new,c);
00019 }
```

2.30 io.c

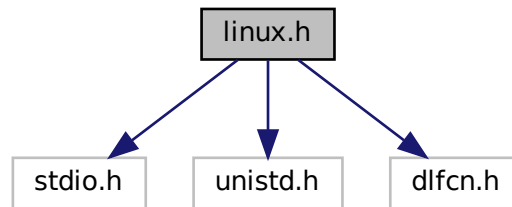
```
00001 #include <unistd.h>
00002 #include <errno.h>
00003 #include <stdlib.h>
00004 #include <stdio.h>
00005
00006 char* readln(int fd, ssize_t max){
00007     char b;
00008     char static new[2048];
00009     unsigned long c = 0;
00010     long l = 0;
00011     long r;
00012     while ((r = read(fd,&b,1)) == -1 && errno==EINTR){
00013         new[c++] = b;
00014         if (b == '\n'){
00015             break;
00016         }
00017     }
00018     return realloc(new,c);
00019 }
00020
00021 char *convert_argv_to_v (int argc, char *argv[], int max)
00022 {
00023     char static new[2048];
00024     int c = 0;
00025     for (int i = 0; i < max; i++)
00026     {
00027         for (int j = 0; argv[i][j] != 0; j++)
00028             new[c++] = argv[i][j];
00029         if (i < argc - 1)
00030             new[c++] = ' ';
00031     }
00032     new[c] = 0;
00033     return new;
00034 }
```

2.31 linux.h File Reference

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

```
#include <dlfcn.h>
```

Include dependency graph for linux.h:



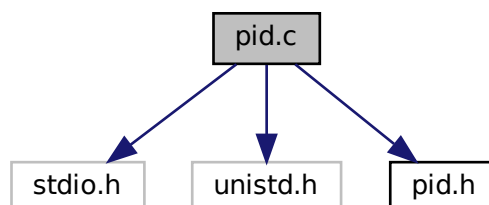
2.32 linux.h

```
00001 #ifndef _LINUX_H_
00002 #define _LINUX_H_
00003 #include <stdio.h>
00004 #include <unistd.h>
00005 #include <dlfcn.h>
00006
00007 #endif //_LINUX_H_
```

2.33 pid.c File Reference

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

Include dependency graph for pid.c:



Functions

- int [print_pid](#) (void)
- int [print_father](#) (void)
- int [print_child_report](#) (void)
- int [print_father_report](#) (pid_t pid, int exit_code)

2.33.1 Function Documentation

2.33.1.1 print_child_report()

```
int print_child_report (
    void )
```

Definition at line 15 of file [pid.c](#).

```
00016 {
00017     fprintf (stderr, "My father's PID: %d\nMy PID: %d\n", getppid (), getpid ());
00018     return 0;
00019 }
```

Referenced by [main\(\)](#).

2.33.1.2 print_father()

```
int print_father (
    void )
```

Definition at line 10 of file [pid.c](#).

```
00011 {
00012     printf ("My father's PID: %d\n", getppid ());
00013     return 0;
00014 }
```

2.33.1.3 print_father_report()

```
int print_father_report (
    pid_t pid,
    int exit_code )
```

Definition at line 21 of file [pid.c](#).

```
00022 {
00023     fprintf (stderr, "Filho %d saiu com código %d\n", pid, exit_code);
00024     return 0;
00025 }
```

2.33.1.4 print_pid()

```
int print_pid (
    void )
```

Definition at line 5 of file [pid.c](#).

```
00006 {
00007     printf ("My PID: %d\n", getpid ());
00008     return 0;
00009 }
```

2.34 pid.c

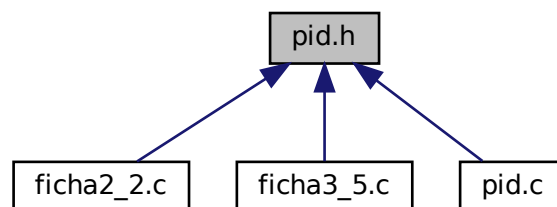
```

00001 #include <stdio.h>
00002 #include <unistd.h>
00003 #include "pid.h"
00004
00005 int print_pid (void)
00006 {
00007     printf ("My PID: %d\n", getpid ());
00008     return 0;
00009 }
00010 int print_father (void)
00011 {
00012     printf ("My father's PID: %d\n", getppid ());
00013     return 0;
00014 }
00015 int print_child_report (void)
00016 {
00017     fprintf (stderr, "My father's PID: %d\nMy PID: %d\n", getppid (), getpid ());
00018     return 0;
00019 }
00020
00021 int print_father_report (pid_t pid, int exit_code)
00022 {
00023     fprintf (stderr, "Filho %d saiu com código %d\n", pid, exit_code);
00024     return 0;
00025 }

```

2.35 pid.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int `print_pid` (void)
- int `print_father` (void)
- int `print_child_report` (void)
- int `print_father_report` (pid_t, int)

2.35.1 Function Documentation

2.35.1.1 print_child_report()

```
int print_child_report (
    void )
```

Definition at line 15 of file [pid.c](#).

```
00016 {
00017     fprintf (stderr, "My father's PID: %d\nMy PID: %d\n", getppid (), getpid ());
00018     return 0;
00019 }
```

Referenced by [main\(\)](#).

2.35.1.2 print_father()

```
int print_father (
    void )
```

Definition at line 10 of file [pid.c](#).

```
00011 {
00012     printf ("My father's PID: %d\n", getppid ());
00013     return 0;
00014 }
```

2.35.1.3 print_father_report()

```
int print_father_report (
    pid_t ,
    int )
```

Definition at line 21 of file [pid.c](#).

```
00022 {
00023     fprintf (stderr, "Filho %d saiu com código %d\n", pid, exit_code);
00024     return 0;
00025 }
```

2.35.1.4 print_pid()

```
int print_pid (
    void )
```

Definition at line 5 of file [pid.c](#).

```
00006 {
00007     printf ("My PID: %d\n", getpid ());
00008     return 0;
00009 }
```

2.36 pid.h

```
00001 #ifndef _PID_H_
00002 #define _PID_H_
00003 int print_pid (void);
00004 int print_father (void);
00005 int print_child_report (void);
00006 int print_father_report (pid_t, int);
00007 #endif // _PID_H_
```


Index

call_by_string.c, [3](#), [4](#)
 main, [3](#)
convert_argv_to_v
 io.c, [24](#)

ficha2_2.c, [5](#), [6](#)
 main, [5](#)
ficha2_3.c, [6](#), [7](#)
 main, [6](#)
ficha2_4.c, [7](#), [8](#)
 main, [8](#)
ficha2_5.c, [8](#), [10](#)
 main, [9](#)
ficha3_1.c, [10](#), [11](#)
 main, [10](#)
ficha3_2.c, [11](#), [12](#)
 main, [11](#)
ficha3_3.c, [12](#), [13](#)
 main, [13](#)
ficha3_4.c, [14](#)
 main, [14](#)
ficha3_5.c, [15](#), [16](#)
 main, [15](#)
ficha3_6.c, [16](#), [17](#)
 main, [16](#)
ficha4_2.c, [17](#), [18](#)
 main, [18](#)
ficha4_2_copy.c, [19](#), [20](#)
 main, [19](#)
ficha5_3.c, [20](#), [23](#)
 main, [21](#)
 MAX_LINE_SIZE, [21](#)
 readln, [22](#)

io.c, [23](#), [25](#)
 convert_argv_to_v, [24](#)
 readln, [24](#)

linux.h, [25](#), [26](#)

main
 call_by_string.c, [3](#)
 ficha2_2.c, [5](#)
 ficha2_3.c, [6](#)
 ficha2_4.c, [8](#)
 ficha2_5.c, [9](#)
 ficha3_1.c, [10](#)
 ficha3_2.c, [11](#)
 ficha3_3.c, [13](#)
 ficha3_4.c, [14](#)

 ficha3_5.c, [15](#)
 ficha3_6.c, [16](#)
 ficha4_2.c, [18](#)
 ficha4_2_copy.c, [19](#)
 ficha5_3.c, [21](#)
MAX_LINE_SIZE
 ficha5_3.c, [21](#)

pid.c, [26](#), [28](#)
 print_child_report, [27](#)
 print_father, [27](#)
 print_father_report, [27](#)
 print_pid, [27](#)

pid.h, [28](#), [29](#)
 print_child_report, [28](#)
 print_father, [29](#)
 print_father_report, [29](#)
 print_pid, [29](#)

print_child_report
 pid.c, [27](#)
 pid.h, [28](#)

print_father
 pid.c, [27](#)
 pid.h, [29](#)

print_father_report
 pid.c, [27](#)
 pid.h, [29](#)

print_pid
 pid.c, [27](#)
 pid.h, [29](#)

readln
 ficha5_3.c, [22](#)
 io.c, [24](#)