PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

INF1316
Sistemas Operacionais

Laboratório 1

2210875 Luísa Ferreira da Silveira

2210461 Rafael Chaves Bayão Ribeiro

Professores:

Luiz Fernando Seibel

Rio de Janeiro
Setembro de 2024

1. EXERCÍCIO 1

1.1. Objetivo

Elaborar programa para criar 2 processos hierárquicos (pai e filho) onde é declarado um vetor de 10 posições inicializado com 0. Após o fork() o processo pai faz um loop de somando 1 às posições do vetor, exibe o vetor e espera o filho terminar. O processo filho subtrai 1 de todas as posições do vetor, exibe o vetor e termina. Explique os resultados obtidos (por quê os valores de pai e filho são diferentes? Os valores estão consistentes com o esperado?)

1.2. Estrutura do Programa

O módulo principal do programa realiza a criação de um processo filho e a manipulação de um vetor compartilhado entre o processo pai e o processo filho, onde o processo pai acrescenta 1 às posições do vetor e o processo filho subtrai 1.

Funções Implementadas:

- fork(): Cria um novo processo.
- waitpid(): O processo pai espera o processo filho acabar.
- printf(): Imprime o estado do vetor.

1.3. Solução

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

int main() {
  int vetor[10] = {0};
  int i, status;
  int pid = fork();
  if (pid != 0) { /* do pai */
    for (i=0; i < 10; i++){
      vetor[i] += 1;

    }
    printf("Saída do Pai:\n");
    for (i = 0; i < 10; i++) {
        printf("%d %d\n", i, vetor[i]);
    }
    waitpid(-1, &status, 0);
    exit(0);
  }
  else { /* do filho */
    for (i=0; i < 10; i++){
      vetor[i] -= 1;
    }
    printf("Saída do filho:\n");
    for (i=0; i < 10; i++){
      printf("%d %d\n",i,vetor[i]);
    }
    exit(0);
  }

  return 0;
}
```

Primeiro, criamos o vetor de 10 posições e inicializamos com o valor zero. Depois, a função fork() é chamada para criar um novo processo, o processo filho. Diferenciamos o processo pai do filho pelo valor retornado por fork(). O processo pai é quando o valor é diferente de zero e o filho é quando é igual a zero. O processo pai incrementa cada elemento do vetor em 1. Após modificar o vetor, o processo pai imprime o vetor e espera a conclusão do processo filho usando waitpid() e, por fim, é finalizado com exit(0). Já o processo filho subtrai 1 cada elemento do vetor, imprimindo o estado do vetor logo em seguida e finalizando com exit(0).

1.4. Conclusão

```
luisa@NoteLu:~/inf1316/lab1$ gcc -Wall -o ex1 ex1.c
luisa@NoteLu:~/inf1316/lab1$ ./ex1
Saída do Pai:
0 1
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
Saída do filho:
0 -1
1 -1
2 -1
3 -1
4 -1
5 -1
6 -1
7 -1
8 -1
9 -1
```

Na saída, cada elemento do vetor do processo pai foi 1 e do processo filho foi -1. Isso ocorre, pois processo pai e o processo filho possuem cópias independentes do vetor (quando a função fork() é chamada, o processo pai e o filho recebem, cada um, cópias do espaço de memória, e as mudanças feitas por um processo em uma cópia não afetam a outra). Então, como o vetor é zero no início, o processo pai acrescenta 1, resultando em 1 para todas as posições do vetor no processo pai. Já o processo filho subtrai 1, resultando em -1 para todas as posições do vetor no processo filho.

2. EXERCÍCIO 2

2.1. Objetivo

Programar funcionalidades dos utilitários do unix - “echo”
Ex: $meuecho bom dia /* exibe os parâmetros de meuecho */ bom dia

2.2. Estrutura do Programa

O módulo principal do programa interage com os argumentos passados na linha de comando e imprime, a partir do índice 1, os argumentos na saída

Funções implementadas:

- printf
- EXIT_SUCCESS

2.3. Solução

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

int main(int argc, char *argv[]) {
  int i;
  for (i = 1; i < argc-1; i++)
  {
    (void) printf("%s%s", argv[i], " ");
  }
  (void) printf("%s%s", argv[argc-1], "\n");
  return EXIT_SUCCESS;
}
```

Primeiro é preciso entender que os parâmetros recebidos na função main. O argc é um inteiro que representa o número total de argumentos passados para o programa na linha de comando. Já o `argv` é um array de strings que contém os argumentos passados para o programa. O argv[0] é o nome do programa que quer utilizar.

Então, voltando para a explicação do código, inicialmente, é feito loop para processar os argumentos fornecidos na linha de comando, percorrendo argumentos começando do índice 1, já que argv[0] é o próprio nome do programa, até o penúltimo índice (argc-1 é o último, pois o índice começa no zero).

Cada argumento é impresso seguido por um espaço e o último argumento é impresso sem espaço seguido de uma nova linha.

Por fim, o programa é finalizado com EXIT_SUCCESS, mostrando que a operação foi bem-sucedida.

2.4. Conclusão

Saída do programa:

```
luisa@NoteLu:~/inf1316/lab1$ ./meuecho bom dia!
bom dia!
luisa@NoteLu:~/inf1316/lab1$ ./meuecho Desenvolvedores:LuísaSilveira e Rafael Ribeiro
Desenvolvedores:LuísaSilveira e Rafael Ribeiro
```

Dessa forma, pode-se observar que o programa ocorreu sem erros.

3. EXERCÍCIO 3

3.1. Objetivo

Programar funcionalidades do utilitário do unix “cat”
Ex: $meucat echo.c cat.c /* exibe os arquivos echo.c e cat.c */

3.2. Estrutura do Programa

O programa é composto por duas funções:

Função print_file: responsável por abrir um arquivo, ler seu conteúdo e imprimi-lo. Utiliza as funções fopen, fgetc e fclose

Função main: responsável por processar os argumentos da linha de comando e chama print_file para cada arquivo especificado na linha de comando.

3.3. Solução

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

void print_file(const char* filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Unable to open file %s\n", filename);
        return;
    }
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);
    }
    fclose(file);
}

int main(int argc, char* argv[]) {
    for (int i = 1; i < argc; i++) {
        print_file(argv[i]);
        printf("\n");
    }
    return 0;
}
```

A função print_file abre o arquivo no modo leitura ("r"). Se o arquivo não puder ser aberto, a função imprime uma mensagem de erro e retorna. Se o arquivo é aberto com sucesso, a função lê o arquivo caractere por caractere usando a função fgetc e imprime cada caractere usando putchar até o arquivo acabar, fechando-o com fclose.

Na função main, seus parâmetros são a quantidade de argumentos passados passados na linha de comando (int argc) e um array de strings (char*argv[]). O argv[0] é o nome do programa, e os argumentos seguintes são nomes de arquivos .

Para cada nome de arquivo fornecido (a partir do índice 1), o programa chama a função print_file para imprimir o conteúdo do arquivo.

Depois de imprimir o conteúdo de cada arquivo, é impresso um “\n” a fim de separar os conteúdos dos arquivos e tornar mais fácil a visualização.

3.4. Conclusão

Saída do programa:

- Conteúdo dos arquivos meucat.c e meuecho.c

```
luisa@NoteLu:~/inf1316/lab1$ ./meucat meucat.c meuecho.c
#include <stdio.h>
#include <string.h>

void print_file(const char* filename) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        printf("Unable to open file %s\n", filename);
        return;
    }
    char ch;
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);
    }
    fclose(file);
}

int main(int argc, char* argv[]) {
    for (int i = 1; i < argc; i++) {
        print_file(argv[i]);
        printf("\n");
    }
    return 0;
}
```

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

int main(int argc, char *argv[]) {
  int i;
  for (i = 1; i < argc-1; i++)
  {
    (void) printf("%s%s", argv[i], " ");
  }
  (void) printf("%s%s", argv[argc-1], "\n");
  return EXIT_SUCCESS;
}
```

- Conteúdo binário dos arquivos echo e cat (usando/bin para acessar os utilitários do unix, indicando em que diretório estão). Como, provavelmente, /bin/echo.c e /bin/cat.c são arquivos binários e o programa meucat foi feito para tratar arquivos de texto, ele lê o conteúdo do arquivo binário e tenta imprimí-lo, resultando em caracteres não legíveis na tela .

```
luisa@NoteLu:~/inf1316/lab1$ ./meucat /bin/echo /bin/cat
@@@@◆   6161```◆◆◆|◆◆◆◆l◆◆|ÒÔ◆8880hhhDDS◆td8880P◆td◆m◆m◆m◆◆Q◆tdR◆td◆|◆◆◆◆XX/lib64/ld-linux-x86-64.so.2 GNU◆◆◆GNU◆k◆)   p
◆◆""__cxa_finalize__libc_start_main__cxa_atexitdcgettexterrorabortnl_langinfo__fpendingfileno__freadingfflush__errno_loc
ationfcloselseekfseekosetlocalestrlen__memcpy_chkstrcmp__stack_chk_fail__ctype_get_mb_cur_maxmemcmp__ctype_b_locmbrtowci
swprintmbsinitstdout_exitstderr__fprintf_chkfputc_unlockedgetenvstrrchrstrncmpprogram_invocation_name__progname_fullbind
textdomainfputs_unlocked__overflowprogram_invocation_short_name__prognamefwrite__printf_chklibc.so.6GLIBC_2.3GLIBC_2.3.4
◆tiBC_2.4GLIBC_2.34GLIBC_2.2.5_ITM_deregisterTMCloneTable__gmon_start___ITM_registerTMCloneTable◆ii

@@@@◆xx   6<6<```◆◆◆z◆◆◆◆@|@◆@◆◆8880hhhDDS◆td8880P◆td◆m◆m◆m◆◆Q◆tdR◆td◆z◆◆  /lib64/ld-linux-x86-64.so.2 GNU◆◆◆GNUD◆◆1wu7;
ʕw◆◆◆◆</◆◆nGNU>◆>◆e◆m4FP9n◆ RYg@◆◆◆◆/◆◆◆◆f◆?◆◆b ◆◆◆◆◆ vrK◆Vy7◆J◆◆◆"◆Z?◆◆ /&◆_"__cxa_finalize__libc_start_main__cxa_atexi
tdcgettext__errno_locationerrorabortnl_langinforead__fpendingfileno__freadingfflushfcloselseekfseeko__fprintf_chkfputc_u
nlocked__stack_chk_failsetlocalestrlen__memcpy_chkstrcmp__ctype_get_mb_cur_maxmemcmp__ctype_b_locmbrtowciswprintmbsinitr
eallocmemsetfreemallocstdout_exitstderrstrrchrstrncmpprogram_invocation_name__progname_fullbindtextdomaingetopt_long__pr
intf_chkfputs_unlockedfstatgetpagesizeoptindopenposix_fadvisestpcpymemmoveioctlprogram_invocation_short_name__prognamefw
ritelibc.so.6GLIBC_2.3GLIBC_2.33GLIBC_2.3.4GLIBC_2.4GLIBC_2.34GLIBC_2.2.5_ITM_deregisterTMCloneTable__gmon_start___ITM_r
q◆◆◆{uʕieiioneTa◆◆a ◆a@b`b◆#b◆-b◆7b@b◆H◆◆P@◆X,◆`1◆◆◆◆◆◆◆◆◆#◆◆$ô7Ų9ᵹ>◆◆◆◆ ◆(◆   0◆
8◆
  @◆
P◆X◆`◆h◆p◆x◆◆◆◆◆◆◆◆◆◆◆◆ʕ◆◆◆◆◆Ôў₄ ◆!◆"◆%◆◆&◆◆(◆)◆* ◆+(◆,0◆-8◆.@◆0H◆1P◆2X◆3`◆4h◆5p◆6x◆8◆◆:◆◆;◆◆<◆◆H◆H◆◆oH◆◆t
```

4. EXERCÍCIO 4

4.1. Objetivo

Programar uma shell e executar os seus programas meuecho, meucat e os utilitários do Unix echo, cat, ls.
Ex: $minhashell meuecho alo alo Realengo aquele abraço /* executa meuecho */ alo alo Realengo aquele abraço.
Ex> $minhashell meucat echo.c cat.c /* executa meucat */ Para executar os utilitários do unix é necessário indicar os diretórios onde eles estão.

4.2. Estrutura do Programa

Esse programa é composto por duas funções:

- separar_comandos: Essa função foi criada apenas para tratar a string recebida do usuário e preencher o vetor args, que é onde ficam as palavras escritas na nossa linha de comandos da shell.

- main: Aqui é onde a nossa shell permanece rodando e aguardando por novos comandos dentro dela, sendo que podemos utilizar os nossos próprios “meucat” e “meuecho” criados anteriormente.

4.3. Solução

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

#define MAX_ARGS 80
#define MAX_LINE 80

void separar_comandos(char *input, char *args[]) {
    int i = 0;
    char *token = strtok(input, " \n");
    while (token != NULL) {
        args[i] = token;
        i++;
        token = strtok(NULL, " \n");
    }
    args[i] = NULL;
}
```

```c
int main() {
    char *args[MAX_ARGS];
    char input[MAX_LINE];
    pid_t pid;
    int status;
    char caminho[MAX_LINE];

    while (1) {
        printf("minhashell> ");
        fgets(input, MAX_LINE, stdin);
        separar_comandos(input, args);

        if (args[0] == NULL) {
            continue;
        }

        if (strcmp(args[0], "exit") == 0) {
            break;
        }

        pid = fork();

        /*processo filho*/
        if (pid == 0) {
            if (strcmp(args[0], "meucat") == 0 || strcmp(args[0], "meuecho") == 0) {
                strcpy(caminho, "./");
                strcat(caminho, args[0]);
                execvp(caminho, args);
            }
            else {
                strcpy(caminho, "/bin/");
                strcat(caminho, args[0]);
                execvp(caminho, args);
                execvp(args[0], args);
            }
            exit(1);
        }
        /*Processo filho*/
        else {
            waitpid( -1, &status, 0);
        }
    }

    return 0;
}
```

O programa serve para funcionar como uma shell do Linux, mas personalizada para esse laboratório. Nós recebemos o input do usuário e tratamos ele na função separar_comandos. Após isso, verificamos se o comando é “exit”, caso a pessoa queira sair da shell e parar de executar o programa. Depois, abrimos o processo com fork() e verificamos o comando inserido pelo usuário para, finalmente, executá-lo com o caminho correto (caso o usuário escolha o nosso comando, acessamos o diretório local; senão, utilizamos o /bin). Esse programa ficará em execução até que o usuário comande que o programa pare, o que encerra o seu funcionamento naquele momento.

4.4. Conclusão

Abaixo estão algumas saídas do nosso programa funcionando:

```
rafaelribeiro@DESKTOP-JUHACO3:~/inf1316$ ./minhashell
minhashell> meuecho oi seibel
oi seibel
minhashell> _
```

```
minhashell> meucat meuecho.c
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int i;
  for (i = 1; i < argc-1; i++)
  {
    (void) printf("%s%s", argv[i], " ");
  }
  (void) printf("%s%s", argv[argc-1], "\n");
  return EXIT_SUCCESS;
}
```

```
minhashell> echo botafogo
botafogo
minhashell> cat meuecho.c
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
  int i;
  for (i = 1; i < argc-1; i++)
  {
    (void) printf("%s%s", argv[i], " ");
  }
  (void) printf("%s%s", argv[argc-1], "\n");
  return EXIT_SUCCESS;
}
minhashell> ls
ex1.c  meucat  meucat.c  meuecho  meuecho.c  minhashell  minhashell.c  minhashell.c:Zone.Identifier
minhashell> _
```

```
minhashell> meuecho vou sair
vou sair
minhashell> exit
rafaelribeiro@DESKTOP-JUHACO3:~/inf1316$
```