Universidade Presbiteriana Mackenzie
Ciência da Computação
Sistemas Operacionais – Turma 04P11
Alan Meniuk Gleizer – 10416804
Caio Vinicius Corsini Filho – 10342005

Relatório Lab 03 – Criação de Processos

Exercício 01
código fonte

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // Para fork() e sleep()
#include <sys/types.h> // Para pid_t

int main() {
    int pid;

    pid = fork(); // Cria processo filho
    // No processo pai, retorna o PID do processo filho criado
    // No processo filho, retorna 0
    // Erro: retorna -1 (e nenhum processo é criado)
    if (pid < 0) {
        printf("Erro na criação do filho\n");
        return 1;
    }

    if (pid == 0) {
        // se estamos no processo filho
        // imprimir PID do filho e valor da variável pid
        printf("Processo Filho: PID = %d, valor do PID (var) = %d.\n", getpid(), pid);

        // Loop (5x) do processo filho
        for (int i = 0; i < 5; i++) {
            printf("Mensagem %d - processo filho\n", i + 1);
            sleep(1); // Espera por 1 segundo
        }
    } else {
        // se pid > 0, imprimimos PID do processo pai
        printf("Processo Pai: PID = %d, PID do Filho = %d\n", getpid(), pid);
    }

    return 0;
}
```

print execução

```
@agleizer ➜ /workspaces/OSlabs/lab03/ex1 (main) $ gcc ex1_ag.c -o ex1_ag
@agleizer ➜ /workspaces/OSlabs/lab03/ex1 (main) $ ./ex1_ag
Processo Pai: PID = 9117, PID do Filho = 9118
Processo Filho: PID = 9118, valor do PID (var) = 0.
Mensagem 1 - processo filho
@agleizer ➜ /workspaces/OSlabs/lab03/ex1 (main) $ Mensagem 2 - processo filho
Mensagem 3 - processo filho
Mensagem 4 - processo filho
Mensagem 5 - processo filho
```

Exercício 02

código fonte

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // Para fork() e sleep()
#include <sys/types.h> // Para pid_t, não usado
#include <sys/wait.h> // Para wait()

int main() {
    int pid;

    pid = fork(); // Cria processo filho
    // No processo pai, retorna o PID do processo filho criado
    // No processo filho, retorna 0
    // Erro: retorna -1 (e nenhum processo é criado)

    if (pid < 0) {
        printf("Erro na criação do filho\n");
        return 1;
    }

    if (pid == 0) {
        // se estamos no processo filho
        // imprimir PID do filho e valor da variável pid
        printf("Processo Filho: PID = %d, valor do PID (var) = %d.\n", getpid(), pid);

        // Loop (5x) do processo filho
        for (int i = 0; i < 5; i++) {
            printf("Mensagem %d - processo filho\n", i + 1);
            sleep(1); // Espera por 1 segundo
        }
    } else {
        // se pid > 0, imprimimos PID do processo pai
```

```
        printf("Processo Pai: PID = %d, PID do Filho = %d\n", getpid(), pid);

        wait(NULL);
        // Espera o término do processo filho
        // fonte: https://stackoverflow.com/questions/42426816/in-what-way-does-
waitnull-work-exactly-in-c
        printf("Estamos no processo pai!\nO processo filho já terminou!\n");
    }

    return 0;
}
```

print execução

```
@agleizer ➜/workspaces/OSlabs/lab03/ex2 (main) $ gcc ex2_ag.c -o ex2_ag
@agleizer ➜/workspaces/OSlabs/lab03/ex2 (main) $ ./ex2_ag
Processo Pai: PID = 11949, PID do Filho = 11950
Processo Filho: PID = 11950, valor do PID (var) = 0.
Mensagem 1 - processo filho
Mensagem 2 - processo filho
Mensagem 3 - processo filho
Mensagem 4 - processo filho
Mensagem 5 - processo filho
Estamos no processo pai!
O processo filho já terminou!
@agleizer ➜/workspaces/OSlabs/lab03/ex2 (main) $ cd
```

Exercício 3

código fonte

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> // Para fork(), exec() e sleep()
#include <sys/types.h> // Para pid_t, não usado
#include <sys/wait.h> // Para wait()

int main() {
    int pid;

    pid = fork(); // Cria processo filho
    // No processo pai, retorna o PID do processo filho criado
    // No processo filho, retorna 0
    // Erro: retorna -1 (e nenhum processo é criado)

    if (pid < 0) {
```

```c
        printf("Erro na criação do filho\n");
        return 1;
    }

    if (pid == 0) {
        // se estamos no processo filho
        // imprimir PID do filho e valor da variável pid
        printf("Processo Filho: PID = %d, valor do PID (var) = %d.\n", getpid(), pid);

        // Substitui o processo filho pelo comando 'ls -l'
        execlp("ls", "ls", "-l", NULL);

        // erro se exec() falhar
        printf("Falha ao executar execlp\n");
        return 1;
    } else {
        // Código do processo pai
        printf("Processo Pai: PID = %d, PID do Filho = %d\n", getpid(), pid);

        // Espera o término do processo filho
        wait(NULL);
        printf("Estamos no processo pai!\nO processo filho já terminou!\n");
    }

    return 0;
}
```

print execução