Sistemas Operacionais – Atividade 1.2
Manual de Funções de Processos em C

David Osvaldo Caldas Pereira[1], **Tales Lima de Oliveira**[1]

[1]Instituto Federal de Brasília (IFB)
Taguatinga – DF – Brasil

{david.pereira3, tales.oliveira}@estudante.ifb.edu.br

1. Introdução

Este manual faz parte de um projeto desenvolvido para a disciplina de Sistemas Operacionais (2024/2), ministrada pelo professor João Oliveira.

O objetivo é demonstrar a criação e o controle de processos utilizando funções da linguagem C, como: **getpid**, **getppid**, **fork**, **execv**, **execve**, **wait** e **waitpid**.

Todos os códigos produzidos para este projeto estão incluídos junto com este manual, permitindo a execução dos exemplos. Além disso, é possível acessar o repositório no GitHub, onde o código está disponível: *Repositório do Projeto*.

2. Funções de Processos em C

2.1. Getpid e Getppid

Essas funções retornam o identificador de processo (PID) do processo atual e o PID do processo pai, respectivamente. O uso dessas informações é fundamental para o rastreamento e controle de processos específicos dentro do sistema operacional.

getpid() retorna o PID do *processo chamador*.

getppid() retorna o PID do *processo pai*.

Exemplo das funções **getpid**() e **getppid**() abaixo:

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

int main(void){
    printf("Current Process ID (PID): %d\n", getpid());
    printf("Parent Process ID (PPID): %d\n", getppid());
    return 0;
}
```

2.2. Fork

Esta função permite a criação de um novo processo filho, que é uma cópia exata do processo pai no momento da chamada. O processo filho herda o mesmo espaço de memória, variáveis e contexto, o que possibilita operações paralelas.

No *processo pai*, **fork**() retorna o PID (Process ID) do *filho*.

No *processo filho*, retorna 0.

Se ocorrer um erro, retorna -1.

Exemplo da função **fork()** abaixo:

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

int main(void){
    pid_t pid = fork();

    if (pid < 0){
        perror("Fork falhou");
        return 1;
    }

    //Processo Filho
    if (pid == 0){
        printf("Filho com PID: %d e ", getpid());
        printf("Pai com PID: %d\n", getppid());
    }

    //Processo Pai
    else{
        printf("Pai com PID: %d ", getpid());
        printf("e Filho com PID: %d\n", pid);
    }

    return 0;
}
```

2.3. Wait e Waitpid

Estas funções permitem que um processo pai aguarde o término de um processo filho, proporcionando sincronização entre processos. Isso é crucial para evitar condições de corrida e garantir que o processo pai obtenha o resultado final do processo filho antes de continuar sua execução.

A função **wait()** faz com que o processo *pai* aguarde a finalização de um processo *filho*.

A função **waitpid()** é uma versão mais flexível, permitindo especificar qual processo *filho* aguardar.

Retorna o PID do *filho* que terminou ou -1 em caso de erro.

Exemplo das funções **wait()** e **waitpid()** abaixo:

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

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

int main(void){
    pid_t pid = fork();

    if (pid < 0){
        perror("Fork falhou");
        return 1;
    }

    //Processo Filho
    if (pid == 0) {
        sleep(2); // Simula Trabalho
        return 0; // Fim do Filho
    }

    //Processo Pai
    int status;
    pid_t waited_pid = wait(&status);
    printf("Filho com PID %d terminou.\n", waited_pid);

    return 0;
}
```

2.4. Execv e Execve

Essas funções substituem o programa em execução em um processo por outro programa, permitindo a execução de novos comandos ou aplicações. Isso é essencial quando um processo precisa alterar seu comportamento para executar tarefas diferentes ou lançar outras aplicações.

A função **execv()** substitui o processo atual por um novo processo. Recebe como argumentos o caminho do executável e um vetor de argumentos.

A função **execve()** é semelhante, mas aceita um vetor de **variáveis de ambiente**.

Em caso de sucesso, não retornam. Em caso de erro, retornam -1.

Exemplo das funções **execv()** e **execve()** abaixo:

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

int main(void){
    // Comando para listar arquivos (ls -l)
    char *args[] = {"/bin/ls", "-l", NULL};

    //Substitui o processo pelo comando 'ls -l'
    execv("args[0]", args);

```

```
11      //Em caso de sucesso o programa não executara essa
        ↪  linha
12      //Mas em caso de erro...
13      printf("execv falhou");
14      return 1;
15  }
```

3. Compilação e Execução do Código

Este projeto utiliza um `Makefile` para simplificar a compilação e execução.

- Para **compilar** os programas, utilize:
  - `make all`
- Para **executar** os programas, utilize:
  - `make run`

  ou
  - `./build/bin/NOME-DO-PROGRAMA`