MODUL PRAKTIKUM 03 (TUGAS TERSTRUKTUR 1)

TOPIK: PEMROGRAMAN CUSTOM LINUX SHELL C BERBASIS POSIX PROCESS API

Mata Kuliah: Sistem Operasi (INF-201) – Alokasi Waktu: 170 Menit – Bobot Evaluasi: 5%

Program Studi Informatika – Fakultas Teknik – Universitas Bengkulu

---

1. CAPAIAN PEMBELAJARAN MODUL (SUB-CPMK 3)

Setelah menyelesaikan Modul Praktikum 03 ini, mahasiswa diharapkan mampu:

1. Mengimplementasikan sistem call pembuat proses POSIX `fork()` dalam bahasa C.
2. Mengganti ruang alamat memori proses menggunakan `execvp()`.
3. Mengelola sinkronisasi proses induk dan mencegah timbulnya *Zombie Process* menggunakan `waitpid()`.
4. Merancang aplikasi terminal *Custom Linux Shell* interaktif sederhana.

2. PRASYARAT TUGAS MEMBACA

Sebelum memulai percobaan laboratorium ini, mahasiswa **wajib** telah membaca bab referensi berikut:

- **[OSTEP]** Chapter 5: *Interlude: Process API* (hal. 39–48).
- **[Think OS]** Chapter 2: *Processes* (hal. 9–16).
- **[xv6 RISC-V]** Chapter 1: *Code: System Calls (`fork`, `exec`)* (hal. 9–14).

3. LANDASAN TEORI RINGKAS

3.1 Dua Fase Pembuatan Proses POSIX: `fork()` & `execvp()`

Pada sistem operasi berbasis POSIX (UNIX/Linux/macOS), pembuatan proses dipisahkan menjadi dua sistem call utama:

1. `pid_t fork(void)`: Menduplikasi proses pemanggil. Proses baru (*child*) mendapatkan duplikat persis memori, register, dan *file descriptors*. Fungsi ini mengembalikan nilai `0` pada proses anak, dan `PID anak` (> 0) pada proses induk.
2. `int execvp(const char *file, char *const argv[])`: Mengganti seluruh citra memori proses saat ini dengan program eksekusi baru. Jika eksekusi berhasil, fungsi ini **tidak pernah kembali** (*never returns*).

3.2 Manajemen Zombie Process & `waitpid()`

Ketika proses anak selesai dieksekusi, statusnya diubah menjadi **ZOMBIE** hingga proses induk membaca kode keluarannya via `wait()` atau `waitpid()`. Jika induk tidak memanggil `wait()`, proses anak akan menggantung sebagai *Zombie* di tabel proses kernel.

4. KODE ACUAN UTUH (EXECUTABLE C LISTING: `myshell.c`)

Berikut adalah kode sumber C lengkap untuk aplikasi terminal *Custom Linux Shell* sederhana:

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

#define MAX_LINE 80 // Panjang maksimum perintah
#define MAX_ARGS 10 // Jumlah maksimum argumen

int main(void) {
    char input[MAX_LINE];
    char *args[MAX_ARGS];

    while (1) {
        printf("myshell> ");
        fflush(stdout);

        // Membaca perintah pengguna
        if (fgets(input, MAX_LINE, stdin) == NULL) break;

        // Menghilangkan karakter newline \n
        input[strcspn(input, "\n")] = 0;

        // Parsing token perintah
        int i = 0;
        args[i] = strtok(input, " ");
        while (args[i] != NULL && i < MAX_ARGS - 1) {
            i++;
            args[i] = strtok(NULL, " ");
        }

        if (args[0] == NULL) continue; // Baris kosong

        // Handling Built-in Command: exit
        if (strcmp(args[0], "exit") == 0) break;

        // Handling Built-in Command: cd
        if (strcmp(args[0], "cd") == 0) {
            if (args[1] == NULL || chdir(args[1]) != 0) {
                perror("myshell cd failed");
            }
            continue;
        }

        // Membuat proses anak (Forking)
        pid_t pid = fork();
        if (pid < 0) {
            perror("Fork failed");
        } else if (pid == 0) {
            // PROSES ANAK: Eksekusi program CLI via execvp
            if (execvp(args[0], args) < 0) {
                perror("Command execution failed");
                exit(EXIT_FAILURE);
            }
        } else {
```

```
            // PROSES INDUK: Tunggu proses anak selesai
            waitpid(pid, NULL, 0);
        }
    }
    return 0;
}
```

Listing 1: Program Custom Linux Shell C (`myshell.c`)

5. PROSEDUR LANGKAH KERJA PERCOBAAN

1. Buat berkas baru `myshell.c` dan salin kode pada Listing 3.1 di atas.
2. Kompilasi kode menggunakan `gcc` dengan opsi peringatan ketat:

```
$ gcc -Wall -Wextra myshell.c -o myshell
```

3. Jalankan executable `./myshell` dan uji perintah CLI standar seperti `ls -l`, `pwd`, `whoami`, `cd ..`, dan `exit`.

6. PANDUAN TROUBLESHOOTING ERROR UMUM

Pesan Error / Kendala	Penyebab Utama	Solusi Solutif
`Segmentation fault (core dumped)`	Argumen array `args` tidak diakhiri dengan pointer `NULL`.	Pastikan elemen terakhir `args[i] = NULL`.
`Command execution failed`	Perintah tidak ditemukan dalam `PATH` sistem.	Periksa ejaan perintah atau berikan path lengkap (misal `/bin/ls`).

7. TUGAS TERSTRUKTUR 1 (BOBOT: 5% NILAI AKHIR)

Kembangkan program `myshell.c` di atas dengan menambahkan 2 fitur lanjutan:

1. **Fitur Eksekusi Background (`&`)**: Jika perintah diakhiri dengan simbol `&` (misal `sleep 10 &`), proses induk **tidak memblokir** dan langsung menampilkan prompt `myshell>` .
2. **Fitur Perintah Built-in `history`**: Menampilkan 5 perintah terakhir yang diketikkan pengguna.