

**LAPORAN PRAKTIKUM**  
**PRAKTIKUM 1:**  
**MANAJEMENT PROSES DI LINUX**



**Disusun oleh:**  
**Misbachul Munir**  
**24060124120031**

**PRAKTIKUM SISTEM OPERASI**  
**LAB A2**

**DEPARTEMEN INFORMATIKA**  
**FAKULTAS SAINS DAN MATEMATIKA**  
**UNIVERSITAS DIPONEGORO**  
**2025**

## **KATA PENGANTAR**

Puji dan syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan makalah yang berjudul "Manajemen Proses di Linux" ini dengan baik. Makalah ini disusun sebagai bentuk pemahaman dan pendalaman materi mengenai sistem operasi Linux, khususnya dalam penggunaan antarmuka baris perintah (CLI) yang menjadi salah satu kekuatan utama Linux dalam pengelolaan sistem.

Dalam penyusunan laporan ini, saya mengucapkan terima kasih kepada Kak Siril Wafa dan Kak Unggul Adi Kusuma yang telah membantu kami dalam pemahaman materi dan dasar – dasar praktikum.

Dengan ini, saya menyadari bahwa laporan praktikum ini masih jauh dari kata sempurna. Untuk itu, saya dengan sangat terbuka menerima kritik dan saran dari para pembaca. Semoga laporan praktikum ini bermanfaat untuk para pembaca dan saya khususnya dalam usaha meningkatkan kemampuan memprogram, berfikir dan mencari nilai pada interval dengan menggunakan penyelesaian interpolasi.

Semarang, 25 November 2025

Penulis

# BAB I

## PENDAHULUAN

### 1.1. Rumusan Masalah

### 1.1.1. Soal

Praktekkan dari direktori home anda:

**Lab1.c**[illegible]

### 1.1.2. Soal

**Lab2.c**[illegible]

### 1.1.3. Soal

### Lab3.c ( Banyak Fork )

```
#include <stdio.h>
#include <unistd.h> /* Berisi prototype fork */

int main(void)
{
    printf("Here I am just before first forking statement\n");
    fork();
    printf("#####\n");
    printf("Here I am just after first forking statement\n");
    fork();
    printf("Here I am just after second forking statement\n");
    printf("\t\tHello World from process %d!\n", getpid());
}
```

#### 1.1.4. Soal

### Lab4.c: Menjamin proses child akan mencetak pesannya sebelum proses parent.

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h> /* mengandung fungsi wait */

void main(void)
{
    int pid;
    int status;

    printf("Hello World!\n");
    pid = fork();
    if (pid == -1) /* kondisi jika fork error */
    {
        perror("bad fork");
        exit(1);
    }
    if (pid == 0) printf("I am the child process.\n");
    else {
        wait(&status); /* parent menunggu child selesai */
        printf("I am the parent process.\n");
    }
}
```

#### 1.1.5. Soal

### Lab5.c

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

main() {
    int forkresult;

    printf("%d: I am the parent. Remember my number!\n", getpid());
    printf("%d: I am now going to fork ... \n", getpid());
    forkresult = fork();
    printf("#####\n");
    if (forkresult != 0) {
        /* proses parent akan mengeksekusi kode di bawah */
        printf("%d: My child's pid is %d\n", getpid(), forkresult);
    }
    else /* hasil fork == 0 */
    { /* proses child akan mengeksekusi kode di bawah */
        printf("%d: Hi! I am the child.\n", getpid());
    }
    printf("%d: like father like son. \n", getpid());
}
```

### 1.1.6. Soal

#### Lab6.c

```
#include <stdio.h>

main()
{
    int pid ;

    printf("I'am the original process with PID %d and PPID %d.\n",
        getpid(), getppid()) ;
    pid = fork () ; /* Duplikasi proses, child dan parent */
    printf("#####\n");
    if ( pid != 0 ) /* jika pid tidak nol, artinya saya parent*/
    {
        printf("I'am the parent with PID %d and PPID %d.\n",
            getpid(), getppid()) ;
        printf("My child's PID is %d\n", pid ) ;
    }
    else /* jika pid adalah nol, artinya saya child */
    {
        sleep(4); /* memastikan supaya parent lebih dulu di terminasi*/
        printf("I'm the child with PID %d and PPID %d.\n",
            getpid(), getppid()) ;
    }
    printf ("PID %d terminates.\n", getpid()) ;
}
```

### 1.1.7. Soal

### Lab7.c

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

int main ()
{
    int pid;

    pid = fork(); /* Duplikasi proses, Child dan parent */
    if (pid != 0) /* jika pid tidak nol, artinya saya parent */
    {
        while (1) /*Tidak Terminate dan tidak mengeksekusi wait()*/
            sleep(100); /* berhenti selama 100 detik */
    }
    else /* pid adalah nol, artinya saya child */
    {
        exit(42); /* exit dengan angka berapapun */
    }
}
```

## 1.2. Tujuan

Praktikum ini bertujuan untuk:

- a) Memahami konsep dasar Command Line Interface (CLI) sebagai antarmuka teks dalam sistem operasi Linux.
- b) Mengenal dan menggunakan perintah-perintah dasar Linux melalui terminal, seperti navigasi direktori, manipulasi file, dan manajemen proses.
- c) Melatih keterampilan penggunaan shell (misalnya Bash) untuk menjalankan perintah, membuat skrip sederhana, dan mengelola sistem secara efisien.
- d) Membandingkan efisiensi CLI dengan GUI dalam konteks administrasi sistem dan otomasi tugas.
- e) Meningkatkan kemampuan troubleshooting dan konfigurasi sistem melalui perintah-perintah CLI yang umum digunakan dalam lingkungan profesional.

## BAB II

### LANDASAN TEORI

#### 1. Pengertian

Manajemen proses adalah aspek fundamental dalam sistem operasi, khususnya Linux, yang berkaitan dengan penciptaan, penghentian, dan pengaturan proses atau program yang sedang berjalan. Sebuah **proses** adalah sebuah program yang sedang dieksekusi.

#### 2. Status dan Informasi Proses

Proses diidentifikasi menggunakan PID (Nomor Identitas Proses). Untuk melihat proses, digunakan beberapa utilitas dasar:

- a) `ps` (process status): Digunakan untuk melihat proses pada sesi terminal aktif atau semua proses (`ps x`). Informasi yang ditampilkan meliputi PID, TTY (terminal), dan COMMAND (instruksi). Informasi lain yang dapat dilihat menggunakan opsi `-u` (user) meliputi %CPU, %MEM, SIZE, RSS, dan waktu START.
- b) `pstree`: Digunakan untuk melihat struktur hierarki proses.
- c) `top`: Menyediakan tampilan dinamis dan *real-time* dari proses yang sedang berjalan.

#### 2. Foreground dan Background

Proses dapat dijalankan dalam dua mode:

- a) **Foreground Process**: Proses yang secara langsung berinteraksi dengan terminal dan membutuhkan terminal untuk berjalan (misalnya `xlogo` tanpa `&`).
- b) **Background Process**: Proses yang berjalan di belakang layar, melepaskan kontrol terminal, ditandai dengan penggunaan simbol `&` (misalnya `xlogo &`).
- c) `jobs`: Digunakan untuk melihat *job* yang aktif pada terminal, termasuk status dan lokasinya (foreground atau background).
- d) `fg`: Mengembalikan proses dari background ke foreground.

#### 3. Nice dan Renice

Setiap proses memiliki nilai *nice* yang menentukan tingkat prioritasnya. Nilai *nice* memengaruhi bagaimana *scheduler* CPU mengalokasikan waktu pemrosesan.

- a) `renice`: Perintah yang digunakan untuk mengubah nilai prioritas (*nice value*) dari proses yang sudah berjalan. Rentang nilai *nice* umumnya adalah dari -20 (prioritas tertinggi) hingga +19 (prioritas terendah). Mengubah ke nilai yang lebih rendah (prioritas lebih tinggi) biasanya memerlukan izin *superuser* (`sudo`).

#### 4. Pembuatan Proses (fork)

Fungsi `fork()` digunakan untuk menciptakan proses baru, yang disebut proses *child* (anak).

- a) Proses *Child* adalah salinan identik dari proses *parent* (induk), termasuk nilai variabel, kode, dan deskriptor file.
- b) **Nilai Kembalian `fork()`:**
  - 1) Mengembalikan 0 ke proses *child*.
  - 2) Mengembalikan ke proses *parent*.
  - 3) Mengembalikan -1 jika ID proses *child* pembuatan proses gagal.
- c) Proses *child* dan *parent* menjadi independen setelah `fork`, dan proses *child* mulai eksekusi pada pernyataan setelah pemanggilan `fork()`.

#### 5. Jenis Proses Khusus

- a) Proses Orphan: Terjadi ketika proses *parent* berakhir (*terminate*) sebelum proses *child*-nya. Proses *child* tersebut secara otomatis diadopsi oleh proses "init" yang memiliki PID 1.
- b) Proses *Zombie*: Proses yang telah terminasi tetapi masih ada dalam tabel proses karena proses *parent*-nya belum menerima kode pengembaliannya melalui fungsi `wait()`. Proses *zombie* akan hilang ketika *parent*-nya berakhir dan diadopsi oleh proses "init".
- c) `wait()`: Fungsi yang memaksa proses *parent* untuk menunggu proses *child* berhenti, menjamin *child* dieksekusi sebelum *parent* melanjutkan.

## BAB III

## PEMBAHASAN

### 3.1. Soal

Praktekkan dari direktori home anda :

**Lab1.c**[illegible]**Lab2.c**[illegible]

### Lab3.c ( Banyak Fork )

```
#include <stdio.h>
#include <unistd.h> /* Berisi prototype fork */

int main(void)
{
    printf("Here I am just before first forking statement\n");
    fork();
    printf("#####\n");
    printf("Here I am just after first forking statement\n");
    fork();
    printf("Here I am just after second forking statement\n");
    printf("\t\tHello World from process %d!\n", getpid());
}
```

### Lab4.c: Menjamin proses child akan mencetak pesannya sebelum proses parent.

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h> /* mengandung fungsi wait */

void main(void)
{
    int pid;
    int status;

    printf("Hello World!\n");
    pid = fork();
    if (pid == -1) /* kondisi jika fork error */
    {
        perror("bad fork");
        exit(1);
    }
    if (pid == 0) printf("I am the child process.\n");
    else {
        wait(&status); /* parent menunggu child selesai */
        printf("I am the parent process.\n");
    }
}
```

### Lab5.c

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

main() {
    int forkresult;

    printf("%d: I am the parent. Remember my number!\n", getpid());
    printf("%d: I am now going to fork ... \n", getpid());
    forkresult = fork();
    printf("#####\n");
    if (forkresult != 0) {
        /* proses parent akan mengeksekusi kode di bawah */
        printf("%d: My child's pid is %d\n", getpid(), forkresult);
    }
    else /* hasil fork == 0 */
    { /* proses child akan mengeksekusi kode di bawah */
        printf("%d: Hi! I am the child.\n", getpid());
    }
    printf("%d: like father like son. \n", getpid());
}
```

### Lab6.c

```
#include <stdio.h>

main()
{
    int pid ;

    printf("I'am the original process with PID %d and PPID %d.\n",
        getpid(), getppid()) ;
    pid = fork () ; /* Duplikasi proses, child dan parent */
    printf("#####\n");
    if ( pid != 0 ) /* jika pid tidak nol, artinya saya parent*/
    {
        printf("I'am the parent with PID %d and PPID %d.\n",
            getpid(), getppid()) ;
        printf("My child's PID is %d\n", pid ) ;
    }
    else /* jika pid adalah nol, artinya saya child */
    {
        sleep(4); /* memastikan supaya parent lebih dulu di terminasi*/
        printf("I'm the child with PID %d and PPID %d.\n",
            getpid(), getppid()) ;
    }
    printf ("PID %d terminates.\n", getpid()) ;
}
```

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

int main ()
{
    int pid;

    pid = fork(); /* Duplikasi proses, Child dan parent */
    if (pid != 0) /* jika pid tidak nol, artinya saya parent */
    {
        while (1) /*Tidak Terminate dan tidak mengeksekusi wait()*/
            sleep(100); /* berhenti selama 100 detik */
    }
    else /* pid adalah nol, artinya saya child */
    {
        exit(42); /* exit dengan angka berapapun */
    }
}
```

## Soal 1

[illegible]

## Soal 2

```

misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <unistd.h> /* Berisi Prototype fork */

int main(void)
{
    int pid;

    printf("Hello World!\n");
    printf("I am the parent process and pid is : %d .\n", getpid());
    printf("Here I am before use of forking\n");
    pid = fork();
    printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");
    printf("Here I am just after forking\n");
    if (pid == 0)
        printf("I am the child process and pid is :%d.\n", getpid());
    else
        printf("I am the parent process and pid id: %d .\n",getpid());
    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");
}

```

```

misbah@LAPTOP-V410AQSA:~$ vi Lab2.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab2.c
misbah@LAPTOP-V410AQSA:~$ ./a.out
Hello World!
I am the parent process and pid is : 1554 .
Here I am before use of forking
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
I am the parent process and pid id: 1554 .
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
I am the child process and pid is :1555.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

```

Soal 3

```

misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <unistd.h> /* Berisi Prototype fork */

int main(void)
{
    printf("Here I am just before first forking statement\n");
    fork();
    printf("#####\n");
    printf("Here I am just after first forking statement\n");
    fork();
    printf("Here I am just after second forking statement\n");
    printf("\t\tHello World from process %d!\n", getpid());
}

```

```

misbah@LAPTOP-V410AQSA:~$ vi Lab3.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab3.c
misbah@LAPTOP-V410AQSA:~$ ./a.out
Here I am just before first forking statement
#####
Here I am just after first forking statement
#####
Here I am just after first forking statement
Here I am just after second forking statement
        Hello World from process 1564!
Here I am just after second forking statement
        Hello World from process 1563!
Here I am just after second forking statement
Here I am just after second forking statement
        Hello World from process 1565!
        Hello World from process 1566!

```

Soal 4

```
misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h> /* mengandung fungsi wait */

void main (void)
{
    int pid;
    int status;

    printf("Hello World!\n");
    pid = fork();
    if (pid == -1) /* kondisi jika fork error */
    {
        perror("bad fork");
        exit(1);
    }
    if (pid == 0) printf ("I am the child process.\n");
    else {
        wait (&status); /*parent menunggu child selesai */
        printf("I am the parent process.\n");
    }
}
```

```
misbah@LAPTOP-V410AQSA:~$ vi Lab4.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab4.c
misbah@LAPTOP-V410AQSA:~$ ./a.out
Hello World!
I am the child process.
I am the parent process.
```

Soal 5

```

misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>

int _main() {
    int forkresult;

    printf("%d: I am the parent. Remember my number!\n", getpid());
    printf("%d: I am now going to fork ... \n", getpid());
    forkresult = fork();
    printf("#####\n");
    if (forkresult != 0) {
        /* proses parent akan mengeksekusi kode di bawah */
        printf("%d: My child's pid is %d\n", getpid(), forkresult);
    }
    else /* hasil fork == 0 */
    { /* proses child akan mengeksekusi kode di bawah */
        printf("%d: Hi! I am the child. \n", getpid());
    }
    printf("%d: like father like son. \n", getpid());
}

```

```

I am the parent process.
misbah@LAPTOP-V410AQSA:~$ vi Lab5.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab5.c
misbah@LAPTOP-V410AQSA:~$ ./a.out
1586: I am the parent. Remember my number!
1586: I am now going to fork ...
#####
1586: My child's pid is 1587
1586: like father like son.
#####
1587: Hi! I am the child.
1587: like father like son.

```

Soal 6

```

misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <unistd.h>

int main()
{
    int pid;

    printf("I'am the original process with PID %d and PPID %d.\n", getpid(), getppid());
    pid = fork (); /*Duplikasi proses, child dan pareng */
    printf("#####\n");
    if (pid != 0) /* jika pid tidak nol, artinya saya parent */
    {
        printf("I'am the parent with PID %d and PPID %d.\n", getpid(), getppid());
        printf("My child's PID is %d\n", pid) ;
    }
    else /* jika pid adalah nol, artinya saya child */
    {
        sleep(4); /* memastikan supata parent lebih dulu di terminasi */
        printf("I'm the child with PID %d and PPID %d.\n", getpid(), getppid());
    }
    printf ("PID %d terminates.\n", getpid()) ;
}

```

```

misbah@LAPTOP-V410AQSA:~$ vi Lab6.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab6.c
misbah@LAPTOP-V410AQSA:~$ ./a.out
I'am the original process with PID 1595 and PPID 1439.
#####
I'am the parent with PID 1595 and PPID 1439.
My child's PID is 1596
PID 1595 terminates.
#####
misbah@LAPTOP-V410AQSA:~$ I'm the child with PID 1596 and PPID 1435.
PID 1596 terminates.

```

## Soal 7

```

misbah@LAPTOP-V410AQSA: ~
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main()
{
    int pid;

    pid = fork(); /* Duplikasi proses, Child dan parent */
    if (pid != 0) /* jika pid tidak nol, artinya saya parent */
    {
        while (1) /* tidak terminate dan tidak mengeksekusi wait()*/
            sleep(100); /* berhenti selama 100 detik */
    }
    else /* pid adalah nol, artinya saya child */
    {
        exit(42); /* exit dengan angka berapapun */
    }
}

```

```

misbah@LAPTOP-V410AQSA:~$ vi Lab7.c
misbah@LAPTOP-V410AQSA:~$ gcc Lab7.c
misbah@LAPTOP-V410AQSA:~$ ./a.out &
[1] 1606
misbah@LAPTOP-V410AQSA:~$ ps
  PID TTY          TIME CMD
 1439 pts/2    00:00:00 bash
  1606 pts/2    00:00:00 a.out
  1607 pts/2    00:00:00 a.out
  1608 pts/2    00:00:00 ps
misbah@LAPTOP-V410AQSA:~$ kill 1606
misbah@LAPTOP-V410AQSA:~$ ps
  PID TTY          TIME CMD
 1439 pts/2    00:00:00 bash
 1609 pts/2    00:00:00 ps
[1]+  Terminated                  ./a.out

```

### 3.3. Penjelasan

#### Soal 1:

Soal nomor satu menunjukkan fungsi sistem `fork()` untuk manajemen proses. Program dimulai dengan satu proses yang mencetak "Hello World" satu kali. Setelah itu memanggil fungsi `fork()`. sistem operasi menciptakan proses identik baru (child), yang merupakan salinan dari parent, dan kedua proses melanjutkan ke baris selanjutnya. Karena fungsi `fork()` ini, baris mulai <<<< sampai >>>>, kalimat "I am after forking", dan PID (Process ID) akan dieksekusi dua kali (satu oleh parent dan satu oleh child), masing-masing menampilkan PID uniknya sendiri.

#### Soal 2:

Soal kedua menunjukkan fungsi `fork()` dan mekanisme untuk membedakan proses parent dan child menggunakan nilai kembalinya. Program dimulai dengan satu proses yang mencetak "Hello World!" beserta PID-nya, lalu memanggil `fork()`. Nilai kembalian `fork()` disimpan dalam variabel `pid`: 0 untuk proses child, dan ID proses child untuk proses parent. Kedua proses (parent dan child) melanjutkan eksekusi ke pernyataan `if/else` untuk mencetak blok pemisah (<<<<...>>>>) dan mengidentifikasi diri mereka menggunakan kondisi `if (pid == 0)`: jika nilainya 0 (child), ia mencetak bahwa ia adalah proses child; jika tidak (parent), ia mencetak bahwa ia adalah proses parent, sebelum akhirnya kedua proses mencetak batas penutup (>>>>...>>>>).

#### Soal 3:

Soal ketiga menunjukkan pembuatan banyak proses melalui dua kali pemanggilan fungsi `fork()`. Program dimulai dengan satu proses(parent) yang mencetak pesan

sebelum `fork()` pertama. Pemanggilan `fork()` yang pertama menciptakan satu proses anak (child yang pertama), sehingga ada dua proses yang berjalan. Kedua proses tadi kemudian mencetak batas (#####...) dan pesan setelah `fork()` pertama. Selanjutnya, ketika `fork()` kedua dipanggil, setiap proses yang ada (parent dan child yang pertama) akan membuat satu proses anak lagi, sehingga totalnya menjadi empat proses yang berbeda. Keempat proses ini kemudian melanjutkan eksekusi dan mencetak pesan setelah `fork()` kedua, masing-masing menampilkan PID uniknya.

**Soal 4:**

Soal keempat menunjukkan penggunaan fungsi `fork()` yang dikombinasikan dengan fungsi `wait()` untuk memastikan proses child selesai dieksekusi sebelum proses parent melanjutkannya. Setelah mencetak "Hello World!" dan memanggil `fork()`, program memeriksa apakah `fork()` gagal (`pid = -1`). Jika berhasil, proses child (`pid = 0`) segera mencetak pesannya. Sebaliknya, proses parent (else block) memanggil `wait(&status)`. Fungsi `wait()` ini akan menghentikan sementara proses parent hingga proses child-nya berakhir, sehingga menjamin proses child mencetak pesannya terlebih dahulu sebelum proses parent mencetak pesannya sendiri.

**Soal 5:**

Soal kelima menunjukkan bagaimana proses parent menggunakan nilai kembalian `fork()` untuk mengidentifikasi dan mencatat PID dari proses child yang baru dibuat. Setelah proses parent mencetak PID-nya dua kali dan memanggil `fork()`, nilai kembalian disimpan dalam `forkresult`. Melalui pernyataan `if/else`, kedua proses dipisahkan: proses parent akan masuk ke blok `if (forkresult != 0)` dan mencetak PID child-nya, sementara proses child akan masuk ke blok `else (forkresult = 0)` dan menyebut dirinya sebagai child. Kemudian kedua proses secara independen akan mencetak pesan penutup "like father like son" dengan menggunakan PID-nya sendiri

**Soal 6:**

Soal keenam dirancang untuk mengilustrasikan pembentukan proses orphan (seperti yang sudah dijelaskan di modul). Program dimulai dengan satu proses (parent) yang mencetak PID dan PPID-nya (Parent Process ID), kemudian memanggil `fork()`. Kedua proses (parent dan child) melanjutkan eksekusi setelah `fork()`. Proses child (`pid == 0`) dipaksa untuk sleep selama 4 detik, memastikan bahwa proses parent (`pid != 0`) akan terminasi lebih dulu. Setelah parent mencetak pesan identitasnya dan PID child-nya, parent berakhir, menjadikan child sebagai orphan. Ketika child bangun dari sleep, child

akan mencetak PID-nya, namun PPID-nya akan berubah menjadi 1, yang merupakan PID dari proses "init" (proses adopsi). Kemudian kedua proses mencetak pesan terminasi.

**Soal 7:**

Soal ketujuh dirancang untuk membuat proses zombie(yang sudah dijelaskan di modul). Program dimulai dengan memanggil `fork()`, menduplikasi proses menjadi parent dan child. Proses child (`pid = 0`) segera berakhir menggunakan `exit(42)`, menjadi zombie karena parent-nya masih hidup dan belum memanggil `wait()` untuk menerima kode pengembaliannya. Sementara itu, proses parent (`pid != 0`) masuk ke dalam *loop* tak terbatas `while(1)` dan tidak terminasi dan tidak mengeksekusi `wait()`, melainkan hanya `sleep` selama 100 detik secara terus-menerus. Selama proses parent masih berjalan, proses child yang sudah berakhir akan tetap berada di sistem sebagai proses zombie (ditunjukkan dengan status Z pada utilitas `ps`).

## BAB IV

### PENUTUP

#### 4.1. Kesimpulan

Praktikum Manajemen Proses di Linux berhasil memberikan pemahaman mendalam tentang siklus hidup proses, dari penciptaan hingga terminasi, serta interaksi antar proses.

- 1) Identifikasi dan Pemantauan: Perintah `ps`, `pstree`, dan `top` adalah utilitas kunci untuk memantau status, hierarki, dan penggunaan sumber daya oleh proses. Utilitas `top` secara khusus penting untuk pemantauan *real-time* dan interaksi seperti mengubah prioritas (*renice*).
- 2) Kontrol Proses: Pengguna dapat mengelola eksekusi proses dengan memindahkannya antara *foreground* (`fg`) dan *background* (`&`), serta melihat statusnya menggunakan `jobs`.
- 3) Prioritas Proses: Nilai *nice*, yang dapat diubah menggunakan `renice`, memungkinkan pengguna untuk mengalokasikan sumber daya CPU secara diskresioner, memastikan proses kritis memiliki prioritas yang lebih tinggi (nilai *nice* lebih rendah).
- 4) Mekanisme `fork()`: Fungsi `fork()` adalah inti dari penciptaan proses baru, di mana proses *child* dibuat sebagai salinan independen dari *parent*. Proses *child* dibedakan dari *parent* melalui nilai kembalian `fork()` yang unik (0 untuk *child* dan ID *child* untuk *parent*).
- 5) Pengakhiran Proses: Pengelolaan pengakhiran proses melalui `wait()` sangat penting untuk menghindari pembentukan proses *zombie*. Sebaliknya, jika *parent* berakhir lebih dulu, *child* akan menjadi proses *orphan* yang diadopsi oleh proses "init" (PID 1).

#### 4.2. Saran

- 1) Eksplorasi Parameter Lanjutan `ps` dan `top`: Disarankan untuk mempelajari lebih lanjut opsi-opsi pada `ps` (misalnya `ps -eo` untuk format output khusus) dan perintah interaktif di dalam `top` (selain `n`, `d`, `h`, `u`, `r`) untuk mendapatkan kendali dan visibilitas yang lebih rinci terhadap proses sistem.

- 2) Implementasi `wait()` yang Lebih Kompleks: Eksplorasi mendalam pada implementasi `wait()` dan fungsi terkait (`waitpid()`) dalam program *fork* untuk mengelola status keluar dari proses *child* secara eksplisit dan mencegah *zombie*, terutama dalam kasus di mana *parent* menciptakan banyak *child*.
- 3) Integrasi dengan *Shell Scripting*: Praktikum selanjutnya dapat mencakup pembuatan *shell script* yang secara otomatis menggunakan perintah manajemen proses (`ps`, `kill`, `renice`) untuk melakukan tugas administrasi sistem, seperti menghentikan proses yang mengonsumsi terlalu banyak memori atau mengubah prioritas *daemon* tertentu.
- 4) Analisis Performa: Gunakan `top` dan `ps` untuk menganalisis dampak nyata dari perubahan nilai *nice* terhadap waktu eksekusi proses lain yang berjalan secara bersamaan, memberikan pemahaman praktis tentang penjadwalan proses.

## DAFTAR PUSTAKA

- Shotts, W. E. (2019). *The Linux Command Line: A Complete Introduction* (2nd ed.). No Starch Press.
- Sobell, M. G. (2017). *A Practical Guide to Linux Commands, Editors, and Shell Programming* (4th ed.). Pearson Education.
- Nemeth, E., Snyder, G., Hein, T. R., Whaley, B., & Mackin, D. (2017). *UNIX and Linux System Administration Handbook* (5th ed.). Pearson.
- Robbins, A., & Beebe, N. H. F. (2005). *Classic Shell Scripting*. O'Reilly Media.
- Linux Documentation Project. (2023). *Bash Guide for Beginners*. Retrieved from <https://tldp.org/>