LAPORAN PRAKTIKUM

PRAKTIKUM Ke-3

MANAJEMEN PROSES DALAM SISTEM OPERASI LINUX

Disusun oleh:

Hadrian Shandhy Yudha

24060124140207

PRAKTIKUM SISTEM OPERASI

LAB A2

DEPARTEMEN INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

2025

KATA PENGANTAR

Segala puji syukur ke hadirat Allah SWT atas segala rahmat dan karunia-Nya yang diberikan, sehingga "Manajemen Proses Dalam Sistem Operasi Linux" ini dapat terselesaikan dengan baik. Adapun laporan ini kami susun sebagai bagian dari penugasan praktikum Sistem Operasi..

Adapun maksud dan tujuan adanya laporan praktikum ini adalah dapat mengenali dan menerapkan perintah pada lingkungan linux, menjelaskan pembahasan dari command line yang dibuat, serta pembahasan hasil atau keluaran dari penyelesaian menggunakan linux untuk menampilkan command line yang telah dibuat.

Dalam penyusunan laporan ini, saya mengucapkan terima kasih kepada Kak Unggul Adimulia dan Kak Siriel Wafa Nuriel Fahri 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 maupun semua pihak yang membutuhkan.

Semarang, 24 November 2025

Hadrian Shandhy Yudha

BAB I
PENDAHULUAN

1.1. Rumusan Masalah

1.1.1 Bagaimana proses pada sistem operasi linux menggunakan perintah ps, jobs, fg, bg, dan top?

1.1.2 Bagaimana mekanisme sistem operasi linux dalam mengatur proses nice dan renice?

1.1.3 Bagaimana pembuatan dan eksekusi proses terjadi pada studi kasus?

1.2. Tujuan

1.2.1 Mahasiswa mampu mengetahui proses pada sistem operasi linux menggunakan perintah ps, jobs, fg, bg, dan top.

1.2.2 Mahasiswa mampu mengenali sistem operasi linux dalam mengatur proses nice dan renice.

1.3.3 Mahasiswa mampu menjelaskan pembuatan dan eksekusi proses terjadi pada studi kasus.

BAB II
LANDASAN TEORI

2.1 Dasar Teori

Di Linux, proses itu ibarat “makhluk hidup” yang lagi jalan di sistem. Setiap proses punya ID sendiri (PID) dan bisa dilihat atau pantau pakai perintah seperti ps, top, jobs, fg, dan bg. Linux juga ngatur prosesnya pakai multitasking, jadi banyak proses bisa hidup berdampingan dengan state yang beda-beda seperti running, sleeping, atau bahkan zombie. Selain itu, sistem juga punya mekanisme prioritas proses lewat nice value, yang membuat CPU lebih atau kurang “memberi perhatian” ke proses tertentu. Nilai ini bisa diubah pakai nice atau renice.

Dalam pemrograman C di lingkungan UNIX/Linux, proses baru dibuat pakai fork()**,** yaitu fungsi yang men-duplicate proses parent jadi proses child yang mirip banget namun memiliki jalannya sendiri. Dari sinilah muncul situasi menarik kayak proses orphan (anak tanpa parent), zombie (child yang selesai tapi statusnya belum diambil), dan dua proses yang jalan bersamaan setelah fork.

BAB III
PEMBAHASAN

3.1 Studi Kasus

3.1.1 Pembuatan Banyak Proses

A. Model Lab1.c

a. Isi File Lab1.c

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

int main(void)
{
    printf("Hello World\n");

    fork();

    printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");
    printf("I am after forking\n");
    printf("\tI am process %d.\n", getpid());
    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ gcc lab1.c -o lab1
ryk@HadrianShandhyYudha:~$ ./lab1
Hello World
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 530.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 531.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

c. Penjelasan

Proses awal (parent) mencetak "Hello World" memanggil fork(), yang membuat salinan identik bernama child → kemudian melanjutkan eksekusi dari baris setelah fork() secara independen, parent menerima PID child sebagai nilai return, child menerima 0 sehingga masing-masing mencetak blok pesan dengan PID sendiri.

B. Model Lab2.c

a. Isi File Lab2.c

```c
#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 is: %d .\n", getpid());

    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab2.c
ryk@HadrianShandhyYudha:~$ gcc lab2.c -o lab2
ryk@HadrianShandhyYudha:~$ ./lab2
Hello World!
I am the parent process and pid is : 543 .
Here i am before use of forking
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
I am the parent process and pid is: 543 .
I am the child process and pid is : 544.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

c. Penjelasan

Ada dua proses yang menjalankan semua baris setelah fork().. Karena fork() menyalin seluruh state proses, dan tidak ada pengecekan PID sebelum mencetak, keduanya parent dan child mencetak baris itu masing-masing satu kali, sehingga muncul dua kali. Setelah itu, if pid = 0 memisahkan jalur child mencetak dirinya dengan PID 544, parent dengan PID 543.

C. Model Lab3.c

a. Isi File Lab3.c

```c
#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 is: %d .\n", getpid());

    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab3.c
ryk@HadrianShandhyYudha:~$ gcc lab3.c -o lab3
ryk@HadrianShandhyYudha:~$ ./lab3
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 621!
Here I am just after second forking statement
                Hello World from process 623!
Here I am just after second forking statement
Here I am just after second forking statement
                Hello World from process 622!
                Hello World from process 624!
```

c. Penjelasan

Program dimulai dengan satu proses yang mencetak pesan awal, lalu fork() pertama menghasilkan 2 proses (parent + child), sehingga pesan setelahnya dicetak dua kali. Kemudian masing-masing dari kedua proses itu memanggil fork() kedua, menghasilkan total 4 proses akibatnya, kode setelah fork() kedua dijalankan oleh keempat proses tersebut, sehingga pesan “after second forking” dan “Hello World” muncul empat kali, masing-masing dengan PID unik (621–624).

3.1.2 Penyelesaian Proses dan Sleep

A. Model Lab4.c

a. Isi File Lab4.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>    /* berisi prototype fork */
#include <sys/wait.h>  /* mengandung fungsi wait */

int 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");
    }

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab4.c
ryk@HadrianShandhyYudha:~$ gcc lab4.c -o lab4
ryk@HadrianShandhyYudha:~$ ./lab4
Hello World!
I am the child process.
I am the parent process.
ryk@HadrianShandhyYudha:~$
```

c. Penjelasan

Program dimulai dengan mencetak "Hello World!" sekali, lalu memanggil fork() yang membuat proses child. Jika berhasil, child (pid == 0) langsung mencetak "I am the child process.", sedangkan parent (pid > 0) menunggu child selesai berkat wait(&status) sebelum mencetak "I am the parent process.", sehingga output selalu muncul dalam urutan: child lalu parent karena wait() memaksa parent berhenti sementara sampai child berakhir.

B. Model Lab5.c

a. Isi File Lab5.c

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

int main(void)
{
    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 */
        printf("%d: Hi! I am the child.\n", getpid());
    }

    printf("%d: like father like son.\n", getpid());

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab5.c
ryk@HadrianShandhyYudha:~$ gcc lab5.c -o lab5
ryk@HadrianShandhyYudha:~$ ./lab5
710: I am the parent. Remember my number!
710: I am now going to fork ...
################################
710: My child's pid is 711
710: like father like son.
################################
711: Hi! I am the child.
711: like father like son.
```

c. Penjelasan

Program dimulai sebagai proses parent (PID 710) yang mencetak pesan awal, lalu memanggil fork() untuk membuat child (PID 711) setelah fork(), kedua proses mencetak garis ###, lalu parent (forkresult != 0) mencetak PID child-nya (711), sedangkan child (forkresult == 0) mencetak sapaan sebagai child kemudian keduanya mencetak pesan "like father like son" masing-masing dengan PID-nya, menghasilkan dua baris akhir yang berbeda.

3.1.3 Proses Orphan

A. Model Lab6.c

a. Isi File Lab7.c

```c
#include <stdio.h>
#include <unistd.h>   /* untuk fork(), getpid(), getppid(), sleep() */

int main(void)
{
    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 parent mati duluan */
        printf("I'm the child with PID %d and PPID %d.\n",
               getpid(), getppid());
    }

    printf("PID %d terminates.\n", getpid());

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab6.c
ryk@HadrianShandhyYudha:~$ gcc lab6.c -o lab6
ryk@HadrianShandhyYudha:~$ ./lab6
I'am the original process with PID 720 and PPID 503.
################################################
I'am the parent with PID 720 and PPID 503.
################################################
My child's PID is 721
PID 720 terminates.
ryk@HadrianShandhyYudha:~$ I'm the child with PID 721 and PPID 499.
PID 721 terminates.
```

c. Penjelasan

Program dimulai sebagai proses parent (PID 720) yang membuat child (PID 721) via fork(), parent langsung mencetak informasinya lalu berakhir, sedangkan child menunda 4 detik dengan sleep(4) akibatnya, saat child akhirnya mencetak PPID-nya, parent-nya sudah mati, sehingga sistem operasi mengadopsi child tersebut ke proses(PID 1), tetapi karena PPID child menjadi 499 (bukan 1), menunjukkan bahwa child kini menjadi proses orphan yang diadopsi.

3.1.4 Proses Zombie

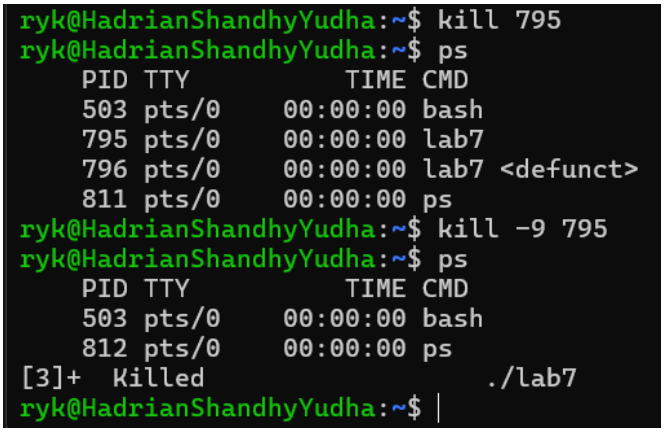
A. Model Lab7.c

a. Isi File Lab7.c

```c
#include <stdio.h>
#include <unistd.h>   /* untuk fork(), getpid(), getppid(), sleep() */

int main(void)
{
    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 parent mati duluan */
        printf("I'm the child with PID %d and PPID %d.\n",
               getpid(), getppid());
    }

    printf("PID %d terminates.\n", getpid());

    return 0;
}
```

b. Output

```
ryk@HadrianShandhyYudha:~$ vi lab7.c
ryk@HadrianShandhyYudha:~$ gcc lab7.c -o lab7
ryk@HadrianShandhyYudha:~$ ps
    PID TTY          TIME CMD
    503 pts/0    00:00:00 bash
    734 pts/0    00:00:00 as
    737 pts/0    00:00:00 as
    794 pts/0    00:00:00 ps
ryk@HadrianShandhyYudha:~$ ./lab7
^Z
[3]+  Stopped                 ./lab7
ryk@HadrianShandhyYudha:~$ ps
    PID TTY          TIME CMD
    503 pts/0    00:00:00 bash
    734 pts/0    00:00:00 as
    737 pts/0    00:00:00 as
    795 pts/0    00:00:00 lab7
    796 pts/0    00:00:00 lab7 <defunct>
    797 pts/0    00:00:00 ps
```



c. Penjelasan

Program ini membuat proses zombie: child (PID 796) langsung exit(42) setelah fork(), tetapi parent (PID 795) tidak pernah memanggil wait() dan malah berjalan terus dalam loop sleep(100), sehingga sistem tidak bisa menghapus entri child dari tabel proses child tetap ada sebagai zombie (<defunct>) meski sudah berhenti, setelah parent dipaksa berhenti dengan kill -9 795, sistem mengadopsi zombie ke init (atau systemd) sehingga zombie akhirnya hilang dari daftar proses.

BAB IV
PENUTUP

4.1 Kesimpulan

Dari praktikum ini, mengelola proses melalui mekanisme forking di mana satu proses dapat melahirkan proses baru yang awalnya merupakan salinan identik tetapi segera berjalan secara independen. Perbedaan antara parent dan child berdasarkan nilai kembalian fork(), serta sinkronisasi menggunakan wait() agar parent tidak berlanjut sebelum child selesai. Selain itu, mengamati dua kondisi khusus proses orphan, yang terjadi ketika parent berhenti lebih dulu sehingga child diadopsi oleh sistem dan proses zombie yang muncul ketika child sudah berhenti tetapi parent tidak memanggil wait() untuk mengambil status keluarnya. Sehingga entry prosesnya tetap tersisa di sistem hingga parent berakhir atau secara eksplisit membersihkannya. Praktikum ini memperkuat pemahaman kita tentang hierarki proses, manajemen sumber daya, dan pentingnya penanganan proses child secara bertanggung jawab dalam pemrograman sistem.

4.2 Saran

Dalam pembelajaran sistem operasi , sebagai mahasiswa informatika tidak hanya memahami konsep perintah command line, tetapi juga mencoba mengimplementasikan secara langsung dalam OS Linux. Dengan demikian, saya berharap kepada pembaca bersedia memberikan saran kritik mengenai makalah ini.

DAFTAR PUSTAKA

Ariata, C. (2025, April 10). *Manajemen proses Linux dengan command line*. Hostinger. https://www.hostinger.com/id/tutorial/manajemen-proses-linux-dengan-command-line