LAPORAN PRAKTIKUM
PRAKTIKUM 3
MANAJEMEN PROSES DI LINUX

Disusun oleh:

Elang Fadila Ahmad

24060124130108

PRAKTIKUM SISTEM OPERASI

LAB A2

DEPARTEMEN INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

2025

A. PEMBAHASAN

1. Lab1.c

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab1.c
```

(2)

```
  GNU nano 7.2                  Lab1.c
#include <stdio.h>
#include <unistd.h> /* Library untuk fork */

int main(void)
{
    printf("Hello World\n");
    fork(); /* Disinilah proses membelah diri */
    printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");
    printf("I am after forking\n");
    printf("\tI am process %d.\n", getpid());
    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");
    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab1.c -o lab1
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab1
Hello World
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 764.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 765.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Penjelasan:

Program ini bermula sebagai satu proses tunggal yang mencetak kalimat "Hello World" satu kali saja. Ketika fungsi fork() dipanggil, sistem operasi secara langsung menduplikasi proses yang sedang berjalan tersebut menjadi dua entitas independen yang disebut sebagai proses induk dan proses anak. Akibatnya, kedua proses tersebut mengeksekusi sisa kode program secara bersamaan sehingga kalimat "I am after forking" muncul dua kali pada layar terminal. Meskipun menjalankan baris kode yang sama, kedua proses tersebut adalah entitas yang berbeda yang dibuktikan dengan adanya dua nomor Process ID atau PID yang berlainan pada hasil keluaran.

2. Lab2.c

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab2.c
```

(2)

```
 GNU nano 7.2                    Lab2.c
#include <stdio.h>
#include <unistd.h>

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(); /* Menyimpan hasil fork ke variabel pid */

    printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");
    printf("Here I am just after forking\n");

    if (pid == 0) {
        /* Jika pid 0, ini adalah kode untuk CHILD */
        printf("I am the child process and pid is %d.\n", getpid());
    } else {
        /* Jika pid > 0, ini adalah kode untuk PARENT */
        printf("I am the parent process and pid is: %d\n", getpid());
    }

    printf(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n");
    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab2.c -o lab2
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab2
Hello World!
I am the parent process and pid is 476
Here i am before use of forking
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
I am the parent process and pid is: 476
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
Here I am just after forking
I am the child process and pid is 477.
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

Penjelasan:

Program ini mendemonstrasikan metode untuk membedakan identitas antara proses induk dan proses anak setelah duplikasi terjadi. Pembedaan tersebut dilakukan dengan memeriksa nilai kembalian dari fungsi fork yang disimpan dalam variabel integer. Jika nilai variabel tersebut adalah nol maka kode yang dijalankan adalah milik proses anak sedangkan nilai selain nol menandakan proses induk.

Penggunaan struktur kendali logika memungkinkan kedua proses melakukan tugas yang berbeda secara spesifik meskipun berasal dari sumber kode yang sama . Pada keluaran program terlihat bahwa proses induk mencetak identitasnya sendiri serta ID anaknya sementara proses anak hanya mencetak identitas dirinya sendiri. Percobaan ini membuktikan bahwa programmer dapat mengontrol perilaku setiap proses secara terpisah segera setelah perintah fork dieksekusi.

3. **Lab3.c**

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab3.c
```

(2)

```
  GNU nano 7.2                  Lab3.c *
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    printf("Here I am just before first forking statement\n");
    fork(); /* Fork pertama */

    printf("####################\n");
    printf("Here I am just after first forking statement\n");

    fork(); /* Fork kedua */

    printf("Here I am just after second forking statement\n");
    printf("\t\tHello World from process %d!\n", getpid());

    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab3.c -o lab3
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./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 491!
Here I am just after second forking statement
Here I am just after second forking statement
                Hello World from process 493!
                Hello World from process 492!
Here I am just after second forking statement
                Hello World from process 494!
```

Penjelasan:

Program ini mengilustrasikan pertumbuhan jumlah proses secara eksponensial akibat pemanggilan fungsi fork secara berulang . Pemanggilan fungsi fork pertama memecah satu proses induk menjadi dua proses yang berjalan serentak. Kedua proses tersebut kemudian mengeksekusi perintah fork kedua sehingga masing masing membelah diri lagi dan menghasilkan total empat proses aktif di akhir program.

Dampak dari penggandaan proses ini terlihat jelas pada jumlah baris keluaran yang tercetak di layar terminal. Pengguna akan melihat kalimat "Hello World" muncul sebanyak empat kali dengan nomor identitas proses atau PID yang berbeda beda untuk setiap barisnya. Hal ini membuktikan bahwa setiap proses hasil duplikasi memiliki independensi untuk menjalankan sisa instruksi program secara mandiri hingga selesai.

4. **Lab4.c**

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab4.c
```

(2)

```
  GNU nano 7.2                    Lab4.c *
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h> /* Library untuk 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) {
        /* Kode untuk Child */
        printf("I am the child process.\n");
    } else {
        /* Kode untuk Parent */
        wait(&status); /* Parent menunggu child selesai di sini */
        printf("I am the parent process.\n");
    }

    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab4.c -o lab4
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab4
Hello World!
I am the child process.
I am the parent process.
```

Penjelasan:

Program ini memperkenalkan fungsi khusus bernama wait yang bertujuan untuk mensinkronisasi urutan eksekusi antara proses induk dan anak. Fungsi tersebut mewajibkan proses induk untuk menunda seluruh kegiatannya sementara waktu sampai proses anak menyelesaikan tugasnya dan berhenti sepenuhnya. Mekanisme penundaan ini menjamin bahwa kalimat keluaran dari proses anak akan selalu tercetak lebih dulu di layar terminal sebelum kalimat dari proses induk muncul.

5. **Lab5.c**

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab5.c
```

(2)

```
 GNU nano 7.2                    Lab5.c *
#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();

    if (forkresult != 0) {
        /* Proses PARENT mengeksekusi kode ini */
        printf("%d: My child's pid is %d\n", getpid(), forkresult);
    }
    else {
        /* Proses CHILD mengeksekusi kode ini */
        printf("%d: Hi! I am the child.\n", getpid());
    }

    /* Baris ini di luar if-else, jadi dijalankan oleh keduanya */
    printf("%d: like father like son. \n", getpid());

    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab5.c -o lab5
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab5
541: I am the parent. Remember my number!
541: I am now going to fork
541: My child's pid is 542
541: like father like son.
542: Hi! I am the child.
542: like father like son.
```

Penjelasan:

Program ini mengeksplorasi hubungan hirarkis yang unik antara proses induk dan anak dengan menampilkan nomor identitas proses secara eksplisit. Logika percabangan yang diterapkan memisahkan instruksi spesifik dimana induk bertugas mencatat nomor identitas anaknya sedangkan anak memperkenalkan dirinya sendiri . Kedua proses tersebut kemudian menjalankan baris kode terakhir yang sama untuk menunjukkan bahwa mereka berbagi segmen kode program yang serupa meskipun memiliki peran yang berbeda.

6. **Lab6.c**

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab6.c
```

(2)

```
  GNU nano 7.2                                  Lab6.c *
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h> /* Untuk fungsi sleep */

int main()
{
    int pid;

    printf("I'am the original process with PID %d and PPID %d.\n", getpid(), getppid());

    pid = fork(); /* Duplikasi proses */

    if (pid != 0) {
        /* KODE PARENT */
        printf("I'am the parent with PID %d and PPID %d.\n", getpid(), getppid());
        printf("My child's PID is %d\n", pid);
        printf("PID %d terminates.\n", getpid());
        /* Parent langsung selesai (mati) di sini */
    }
    else {
        /* KODE CHILD */
        sleep(4); /* Child tidur 4 detik untuk memastikan Parent mati duluan */

        /* Saat bangun, Parent aslinya sudah mati. Cek siapa PPID barunya */
        printf("I'm the child with PID %d and PPID %d.\n", getpid(), getppid());
        printf("PID %d terminates.\n", getpid());
    }

    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ gcc Lab6.c -o lab6
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab6
I'am the original process with PID 564 and PPID 417.
I'am the parent with PID 564 and PPID 417.
My child's PID is 565
PID 564 terminates.
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ I'm the child with PID 565 and PPID 297.
PID 565 terminates.
```

Penjelasan:

Program ini merancang skenario khusus di mana proses induk menyelesaikan tugasnya dan berhenti lebih awal daripada proses anak. Mekanisme penundaan menggunakan fungsi *sleep* membuat proses anak tertidur sementara waktu sehingga proses induk memiliki kesempatan untuk mati terlebih dahulu. Akibatnya hubungan hirarki antara induk dan anak terputus secara sepihak saat proses anak masih aktif berjalan di memori.

Sistem operasi secara otomatis menangani situasi proses yatim piatu ini dengan menugaskan proses *init* atau manajer sistem untuk mengadopsi proses anak yang tertinggal. Hal ini terlihat jelas pada hasil keluaran program di mana nomor ID induk milik proses anak berubah drastis setelah ia bangun dari tidurnya . Pengadopsian ini merupakan mekanisme pertahanan sistem yang krusial untuk mencegah kebocoran sumber daya akibat proses liar yang tidak memiliki pengawas.

7. **Lab7.c**

Langkah-langkah:

(1)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ nano Lab7.c
```

(2)

```
  GNU nano 7.2                          Lab7.c *
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    int pid;

    pid = fork();

    if (pid != 0) {
        /* KODE PARENT */
        /* Parent sengaja looping selamanya dan TIDAK melakukan wait() */
        while (1) {
            sleep(100);
        }
    }
    else {
        /* KODE CHILD */
        /* Child langsung mati (exit) */
        exit(42);
    }

    return 0;
}
```

(3)

```
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ./lab7 &
[2] 775
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1001     671     648  0  80   0 -  1648 do_wai pts/0    00:00:00 bash
0 T  1001     694     671  0  80   0 -   637 do_sig pts/0    00:00:00 lab7
1 Z  1001     695     694  0  80   0 -     0 -      pts/0    00:00:00 lab7
0 S  1001     775     671  0  80   0 -   637 hrtime pts/0    00:00:00 lab7
1 Z  1001     776     775  0  80   0 -     0 -      pts/0    00:00:00 lab7
0 R  1001     777     671  0  80   0 -  2078 -      pts/0    00:00:00 ps
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ kill 775
elangfa@LAPTOP-T33AH4PB:~/Pertemuan3$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S  1001     671     648  0  80   0 -  1648 do_wai pts/0    00:00:00 bash
0 T  1001     694     671  0  80   0 -   637 do_sig pts/0    00:00:00 lab7
1 Z  1001     695     694  0  80   0 -     0 -      pts/0    00:00:00 lab7
0 R  1001     778     671  0  80   0 -  2078 -      pts/0    00:00:00 ps
[2]-  Terminated              ./lab7
```

Penjelasan:
Program ini mensimulasikan kegagalan komunikasi antara dua proses di mana proses induk lalai menerima laporan status akhir dari anaknya. Kode program membuat proses anak berhenti bekerja dengan memanggil fungsi *exit* sedangkan proses induk terus sibuk dalam putaran tanpa henti dan tidak menjalankan fungsi *wait* . Kondisi ini menyebabkan sistem operasi tidak dapat menghapus data proses anak dari memori meskipun tugasnya sebenarnya sudah selesai.

Dampak dari kelalaian tersebut dapat diamati melalui instruksi pemantauan status proses yang menampilkan label *defunct* atau simbol huruf Z pada baris proses anak. Entri ini menandakan bahwa proses tersebut telah menjadi mayat hidup atau *zombie* yang hanya menyisakan nomor identitas tanpa menggunakan sumber daya komputasi aktif. Proses *zombie* tersebut akan terus bertahan mengotori tabel sistem sampai proses induknya dimatikan secara paksa oleh pengguna.

B. KESIMPULAN

Eksperimen pengelolaan proses menunjukkan bahwa fungsi sistem *fork* bekerja dengan menduplikasi satu program berjalan menjadi dua entitas independen yang disebut proses induk dan proses anak. Identifikasi kedua proses tersebut dilakukan melalui pemeriksaan nilai kembalian fungsi di mana nilai nol secara spesifik menandakan proses anak sedangkan nilai lainnya milik proses induk. Penerapan fungsi *wait* selanjutnya menjadi mekanisme vital untuk sinkronisasi waktu yang memastikan proses induk menunda aktivitasnya hingga proses anak menyelesaikan tugas sepenuhnya.

Pengujian terhadap anomali siklus hidup proses memperlihatkan penanganan sistem operasi terhadap kondisi *orphan* dan *zombie* yang terjadi akibat ketidaksinkronan waktu berhenti antara induk dan anak. Kondisi *orphan* tercipta saat proses induk berhenti lebih awal sehingga sistem secara otomatis menugaskan proses *init* untuk mengadopsi proses anak yang masih aktif demi mencegah kebocoran sumber daya. Sebaliknya fenomena *zombie* muncul ketika proses anak telah selesai namun status akhirnya tidak diambil oleh induk yang menyebabkan sisa data proses terus bertahan di memori sebagai entri *defunct*.