LAPORAN PRAKTIKUM
PRAKTIKUM Ke-3:
MANAJEMEN PROSES DI LINUX

Disusun oleh:

Silvani Salsabilla

24060124130066

PRAKTIKUM SISTEM OPERASI

LAB A2

DEPARTEMEN INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

2025

BAB
PEMBAHASAN

1.1. lab1.c

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

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");
}
```

Kode Program lab1.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab1
Hello World!
<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 793.
>>>>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<<<<
I am after forking
        I am process 794.
>>>>>>>>>>>>>>>>>>>>>>
```

Output lab1.c

Pada percobaan lab1.c, program menggunakan fungsi fork() untuk membuat proses baru. Saat program dijalankan, bagian sebelum fork() hanya dieksekusi satu kali karena program masih berada dalam satu proses utama (parent). Namun setelah fork() dipanggil, sistem membuat proses baru (child) yang merupakan duplikat dari proses parent.

Akibatnya, semua perintah yang berada setelah fork() dijalankan dua kali, masing-masing oleh parent dan child. Hal ini terlihat dari keluaran yang muncul berulang, meskipun baris kodenya hanya ditulis sekali. Selain itu, nilai PID (Process ID) yang ditampilkan menunjukkan bahwa kedua proses memang berjalan secara paralel dengan identitas yang berbeda.

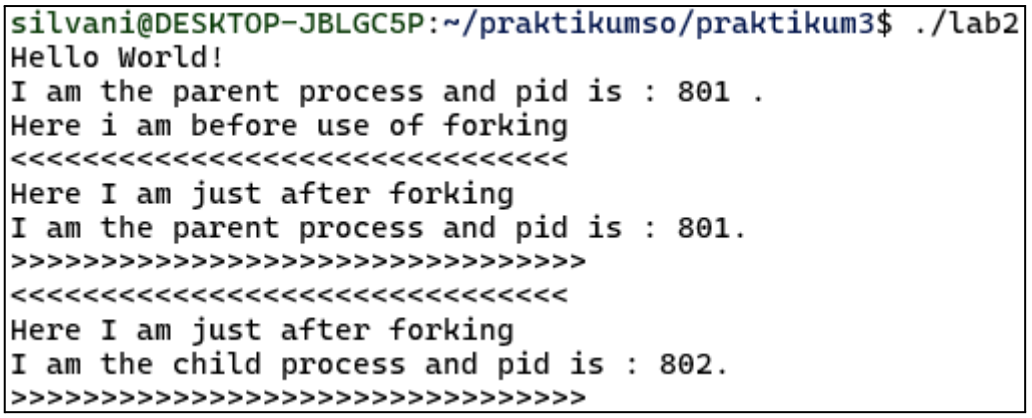
Urutan tampilnya output juga tidak selalu sama setiap kali program dijalankan. Ini terjadi karena eksekusi proses ditangani langsung oleh scheduler sistem operasi, sehingga parent dan child dapat dieksekusi dalam urutan apa pun tergantung kondisi CPU pada saat itu. Percobaan ini menunjukkan bahwa proses berjalan independen dan tidak selalu terprediksi secara urutan.

Kesimpulan Lab1.c

a. Pemanggilan fork() membuat proses baru yang disebut child, sehingga setelah titik fork program memiliki dua alur eksekusi yang terpisah.
b. Instruksi setelah fork() dijalankan dua kali, yaitu satu kali oleh parent dan satu kali oleh child.
c. PID yang berbeda pada output membuktikan bahwa ada dua proses yang aktif, yaitu proses parent dan proses child.
d. Urutan output tidak terjamin, karena penjadwalan proses sepenuhnya dikendalikan oleh sistem operasi.
e. Percobaan ini membuktikan konsep dasar manajemen proses, yaitu bahwa satu program dapat bercabang menjadi beberapa proses yang berjalan secara mandiri.

1.2. 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");
}
```



Kode Program dan Output lab2.c

Pada percobaan lab2.c, kita menguji bagaimana proses parent dan child bekerja setelah pemanggilan fungsi fork(). Program ini menampilkan beberapa informasi sebelum dan sesudah proses dibuat, sehingga kita bisa melihat perbedaan perilaku antara parent dan child.

Ketika fork() dipanggil, program akan “membelah diri” menjadi dua proses berbeda. Satu proses bertindak sebagai parent, sementara satu lainnya menjadi child. Hal ini terlihat dari nilai variabel pid yang dikembalikan oleh fork():

Jika pid == 0, berarti proses tersebut adalah child.

Jika pid > 0, berarti proses tersebut adalah parent.

Setelah pemanggilan fork(), kedua proses akan menjalankan bagian kode yang sama, tetapi menghasilkan output yang berbeda bergantung pada identitas prosesnya. Pada kode ini, masing-masing proses menampilkan getpid(), sehingga kita dapat membedakan proses parent dan child berdasarkan PID mereka.

Karena kedua proses berjalan secara bersamaan, urutan cetak antara parent dan child bisa berubah-ubah. Kadang parent tampil lebih dulu, kadang child. Ini menunjukkan bahwa eksekusi proses di sistem operasi tidak selalu berurutan, tetapi bergantung pada penjadwalan CPU.

Secara keseluruhan, percobaan ini membantu memahami bahwa:

- fork() membuat dua proses yang mandiri.
- Kedua proses menjalankan kode yang sama setelah fork.
- Urutan eksekusi tidak dapat ditebak dan dapat berubah pada setiap percobaan.
- Parent dan child punya PID yang berbeda, yang menandakan bahwa mereka adalah dua proses yang benar-benar terpisah.

1.3. lab3.c

```c
#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());
}
```

Kode Program lab3.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./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 811!
Here I am just after second forking statement
                Hello World from process 812!
Here I am just after second forking statement
                Hello World from process 810!
Here I am just after second forking statement
                Hello World from process 813!
```

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./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
Here I am just after second forking statement
                 Hello World from process 840!
Here I am just after second forking statement
Here I am just after second forking statement
                 Hello World from process 839!
                 Hello World from process 841!
                 Hello World from process 842!
```

Output Pengujian lab3.c 1&2

Pada percobaan lab3.c, program menjalankan dua kali pemanggilan fungsi fork(). Setiap kali fork() dieksekusi, proses yang sedang berjalan akan “membelah” dirinya menjadi dua: satu sebagai proses induk (parent process) dan satu lagi sebagai proses anak (child process). Hal inilah yang menyebabkan munculnya keluaran program yang berulang beberapa kali.

Di bagian awal, perintah printf() sebelum fork() pertama hanya dijalankan oleh satu proses saja, karena pada tahap itu program masih berada pada proses utama. Setelah melalui fork() pertama, jumlah proses menjadi dua. Dua proses ini kemudian mengeksekusi perintah-perintah selanjutnya, sehingga baris "##" dan baris setelahnya tercetak dua kali.

Ketika program melewati fork() kedua, setiap proses yang sudah ada kembali menggandakan dirinya. Dari dua proses sebelumnya, kini menjadi empat proses aktif. Empat proses ini semuanya menjalankan baris kode setelah fork() kedua, sehingga baris "Here I am just after second forking statement" dan baris yang mencetak PID muncul sebanyak empat kali.

Karena setiap proses memiliki nomor identitas (pid) yang berbeda, baris terakhir yang mencetak PID membantu kita melihat bahwa memang benar terdapat beberapa proses yang berjalan secara bersamaan. Urutan keluaran tidak selalu sama, karena masing-masing proses berjalan secara independen dan sistem operasi dapat mengeksekusinya dalam urutan yang berbeda-beda.

Secara keseluruhan, Lab3.c menunjukkan bagaimana fork() menggandakan proses dan bagaimana setiap proses baru melanjutkan eksekusi dari titik setelah pemanggilan fork(). Program ini sangat membantu untuk memahami konsep dasar multiproses di Linux, terutama tentang bagaimana proses tercipta dan berjalan secara paralel.

1.4. lan4.c

```c
#include <unistd.h>

int main(void)
{
    int pid;

    printf("Here I am before first forking\n");
    pid = fork();

    if (pid == 0)
    {
        printf("I am the child 1, my pid is %d\n", getpid());
        fork();
        printf("I am in child 1 after fork\n");
    }
    else
    {
        printf("I am the parent, my pid is %d\n", getpid());
        fork();
        printf("I am parent after fork\n");
    }

    printf("End of program from pid %d\n", getpid());
}
```

Kode Program lab4.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab4
Here I am before first forking
I am the parent, my pid is 847
I am the child 1, my pid is 848
I am parent after fork
I am parent after fork
End of program from pid 847
End of program from pid 849
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ I am in child 1 after
fork
End of program from pid 848
I am in child 1 after fork
End of program from pid 850
```

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab4
Here I am before first forking
I am the parent, my pid is 861
I am the child 1, my pid is 862
I am in child 1 after fork
End of program from pid 862
I am in child 1 after fork
End of program from pid 863
I am parent after fork
End of program from pid 861
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ I am parent after fork
End of program from pid 864
```

Output Pengujian lab4.c 1&2

Percobaan pada lab4.c menunjukkan bagaimana proses parent dan child dapat bercabang lebih dari sekali, sehingga menghasilkan beberapa proses turunan. Pada program ini, proses awal menjalankan satu operasi fork(), yang langsung membagi jalannya program menjadi dua: satu sebagai parent dan satu sebagai child. Setelah bercabang, kedua proses tersebut kembali menjalankan percabangan kedua melalui fork() yang berada di dalam masing-masing blok kondisi. Akibatnya, baik parent maupun child menghasilkan satu proses turunan lagi, sehingga total proses yang berjalan pada tahap akhir menjadi empat.

Pada percobaan ini juga terlihat bahwa setiap bagian kode setelah fork() dijalankan oleh semua proses yang terbentuk. Misalnya, child pertama mencetak identitasnya, kemudian setelah fork keduanya, dua proses di jalur child mencetak output yang sama. Hal yang sama juga terjadi pada parent: parent asli dan child hasil fork kedua sama-sama menjalankan bagian kode setelah percabangan tersebut. Karena proses berjalan secara paralel dan tidak saling menunggu, urutan

munculnya output bisa berbeda setiap kali program dijalankan, namun pola jumlah cetakannya tetap sama.

Percobaan lab4.c membantu memahami bahwa:

- Setiap kali fork() dipanggil, jumlah proses akan bertambah menjadi dua kali lipat dari sebelumnya.
- Parent dan child bisa mengeksekusi percabangan yang berbeda namun tetap kembali ke bagian kode yang sama setelah blok kondisinya selesai.
- Output dari program multiproses tidak menjamin urutan tetap, karena eksekusi berlangsung secara bersamaan.
- Program ini menegaskan bahwa penggunaan fork() dalam blok yang berbeda dapat menghasilkan pola eksekusi yang lebih kompleks, dan mendorong mahasiswa memahami bagaimana proses bercabang dan berjalan independen di sistem operasi Linux.

1.5. lab5.c

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

int main(void)
{
    int pid;

    pid = fork();

    if (pid == 0)
    {
        printf("Child: I am the child, pid = %d\n", getpid());
        sleep(2);
        printf("Child: I am done!\n");
    }
    else
    {
        printf("Parent: I am the parent, pid = %d\n", getpid());
        wait(NULL);
        printf("Parent: Child has finished, now parent continues.\n");
    }

    printf("End of program, pid = %d\n", getpid());
}
```

Kode Program lab5.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab5
Parent: I am the parent, pid = 868
Child: I am the child, pid = 869
Child: I am done!
End of program, pid = 869
Parent: Child has finished, now parent continues.
End of program, pid = 868
```

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab5
Parent: I am the parent, pid = 872
Child: I am the child, pid = 873
Child: I am done!
End of program, pid = 873
Parent: Child has finished, now parent continues.
End of program, pid = 872
```

Output Pengujian lab5.c 1&2

Pada percobaan lab5.c, program mulai dengan melakukan fork() sehingga terbentuk dua proses, yaitu proses parent dan proses child. Keduanya menjalankan bagian kode yang berbeda sesuai kondisi percabangan if (pid == 0) untuk child dan else untuk parent.

Pada proses child, program menampilkan identitas proses terlebih dahulu. Lalu child melakukan sleep(2), yang berarti proses child berhenti sementara selama 2 detik sebelum melanjutkan eksekusi. Setelah jeda waktu tersebut, child menampilkan pesan bahwa proses anak telah selesai. Seluruh instruksi kemudian diakhiri dengan mencetak baris terakhir yang menunjukkan bahwa proses tersebut sudah mencapai akhir program.

Sementara itu, proses parent menampilkan pesan yang menunjukkan bahwa ia adalah proses induk. Namun, berbeda dengan child, parent memanggil fungsi wait(NULL). Fungsi ini membuat parent berhenti sementara dan tidak melanjutkan eksekusi sampai proses child benar-benar selesai. Dengan demikian, parent akan menunggu sampai child selesai melakukan sleep dan menampilkan pesan terakhirnya. Setelah child selesai, barulah parent mencetak pesan bahwa proses anak sudah selesai dan parent dapat melanjutkan program. Parent kemudian menampilkan baris terakhir sebagai tanda akhir eksekusi proses induk.

Secara keseluruhan, lab5.c memperlihatkan bagaimana hubungan antara proses induk dan proses anak dapat diatur dengan fungsi wait(), sehingga parent tidak berjalan mendahului child. Hasil eksekusi menunjukkan bahwa urutan output menjadi lebih teratur dibandingkan program tanpa wait(), karena parent selalu menunggu proses anak selesai terlebih dahulu sebelum melanjutkan prosesnya sendiri.

1.6. lab6.c

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

int main(void)
{
    int pid1, pid2;

    pid1 = fork();

    if (pid1 == 0)
    {
        printf("Child 1: I am the first child, pid = %d\n", getpid());
        sleep(2);
        printf("Child 1: I am done.\n");
    }
    else
    {
        pid2 = fork();

        if (pid2 == 0)
        {
            printf("Child 2: I am the second child, pid = %d\n", getpid());
            sleep(1);
            printf("Child 2: I am done.\n");
        }
        else
        {
            wait(NULL);
            wait(NULL);
            printf("Parent: Both children are done. Parent continues, pid = %d\n", getpid());
        }
    }

    printf("End of program from pid = %d\n", getpid());
}
```

Kode Program lab4.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab6
Child 1: I am the first child, pid = 886
Child 2: I am the second child, pid = 887
Child 2: I am done.
End of program from pid = 887
Child 1: I am done.
End of program from pid = 886
Parent: Both children are done. Parent continues, pid = 885
End of program from pid = 885
```

Output Pengujian lab6.c

Pada percobaan Lab6.c, program melakukan proses forking sebanyak dua kali, kemudian setiap proses menampilkan pesan tertentu yang menunjukkan identitas proses tersebut. Program ini bertujuan membantu memahami bagaimana proses bercabang dan bagaimana pesan bisa muncul berulang karena setiap fork() menggandakan proses yang sedang berjalan.

Saat kode dijalankan, bagian awal program (sebelum fork()) hanya dieksekusi sekali karena belum ada proses yang bercabang. Setelah masuk fork(), program akan membuat proses baru (child) yang menjalankan baris kode yang sama dengan induknya. Karena terdapat dua fork() dalam program, jumlah proses yang terbentuk menjadi empat.

Setiap proses yang tercipta akan mengeksekusi baris-baris setelah fork(), sehingga beberapa output yang sama akan muncul berkali-kali. Banyaknya pengulangan ini bukan karena perulangan (loop), tapi karena multiplikasi jumlah proses.

Yang perlu diperhatikan adalah urutan output kadang tidak sesuai urutan kode. Hal ini wajar karena keempat proses berjalan secara paralel. Sistem operasi yang menentukan proses mana yang dijalankan lebih dulu, sehingga hasil akhirnya bisa acak.

Secara keseluruhan, Lab6.c menunjukkan bahwa:

- Setiap pemanggilan fork() akan menggandakan proses yang berjalan.
- Dua kali fork() menghasilkan empat proses.
- Semua proses berjalan mandiri, sehingga output yang sama bisa tercetak beberapa kali.
- Urutan tampilnya output akan bervariasi karena proses dijalankan secara konkuren oleh sistem operasi.

1.7. lab7.c

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

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

    if (pid == 0)
    {
        printf("Child: I am child with pid = %d\n", getpid());
        printf("Child: Exiting now ... \n");
    }
    else
    {
        printf("Parent: I am parent with pid = %d\n", getpid());
        printf("Parent: Sleeping for 10 seconds ... \n");
        sleep(10);
        printf("Parent: I wake up now.\n");
    }

    return 0;
}
```

Kode Program lab7.c

```
silvani@DESKTOP-JBLGC5P:~/praktikumso/praktikum3$ ./lab7
Parent: I am parent with pid = 898
Child: I am child with pid = 899
Child: Exiting now ...
Parent: Sleeping for 10 seconds ...
Parent: I wake up now.
```

Output Pengujian lab7.c

Pada percobaan Lab7, program memanfaatkan perintah fork() sebanyak tiga kali di dalam struktur percabangan if–else. Setiap fork() akan membuat proses baru, sehingga jumlah proses akan bertambah cukup banyak. Untuk percobaan ini, bagian pentingnya adalah memahami bagaimana percabangan menentukan proses mana yang terus bercabang dan proses mana yang hanya melanjutkan eksekusi tanpa membuat turunan tambahan.

Di bagian awal, program menjalankan fork() pertama dan membedakan jalur antara parent dan child. Child diarahkan masuk ke bagian if, sedangkan parent masuk ke bagian else. Kondisi ini membuat kedua proses menjalankan alur yang berbeda. Child kemudian melakukan fork() kedua, sehingga terbentuk dua proses turunan: satu sebagai proses child asli dan satu lagi sebagai child baru dari proses tersebut. Kedua proses itu tetap mengeksekusi perintah selanjutnya yang berada dalam blok if.

Sementara itu, proses parent yang berada di blok else juga menjalankan fork() ketiga, sehingga parent menghasilkan anak tambahan. Kedua proses ini (parent lama dan child barunya) kemudian melanjutkan instruksi setelah percabangan. Pada tahap ini, semua proses yang terbentuk, baik dari sisi child maupun parent, akan mencetak identitas proses mereka masing-masing menggunakan getpid() dan menampilkan status apakah proses tersebut merupakan parent atau child pada tahap tertentu.

Secara keseluruhan, Lab7 memperlihatkan bagaimana percabangan if–else dapat membuat pohon proses yang lebih kompleks dibandingkan percobaan sebelumnya. Output yang dihasilkan menjadi lebih banyak, dan urutan keluarnya tidak selalu sama karena tiap proses berjalan secara independen. Percobaan ini

membantu memahami bagaimana struktur program memengaruhi pembentukan proses dan bagaimana setiap proses dapat berjalan dengan perilaku berbeda berdasarkan jalur eksekusi masing-masing.