LAPORAN PRAKTIKUM

PRAKTIKUM Ke-3

Manajemen Proses di Linux

Disusun oleh:

Kiyoshi Akila Tira

24060124130074

PRAKTIKUM Sistem Operasi

LAB A2

DEPARTEMEN INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

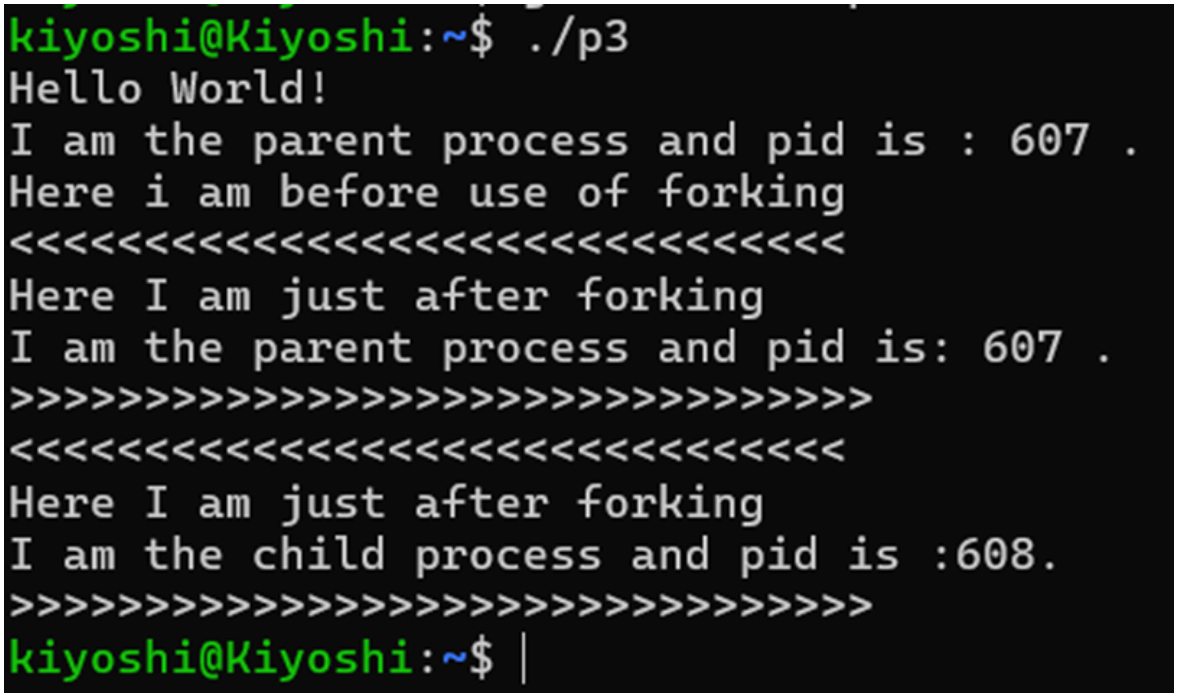
UNIVERSITAS DIPONEGORO

2025

1. Lab1.c

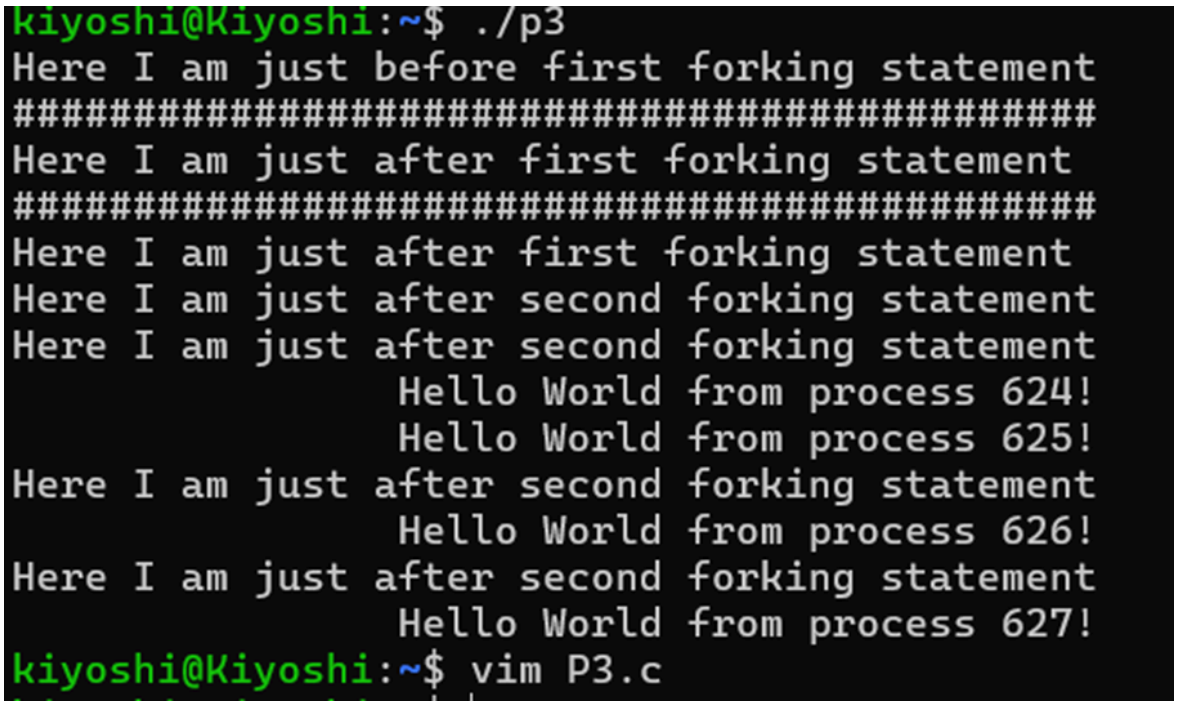
Program di atas bertujuan untuk menunjukkan bagaimana sistem operasi membuat proses baru menggunakan *system call* fork(). Ketika program dijalankan, perintah printf("Hello World\n"); dieksekusi satu kali oleh proses utama (parent process). Setelah itu, fungsi fork() dipanggil dan sistem operasi akan menduplikasi proses tersebut sehingga tercipta dua proses yang berjalan secara paralel: *parent process* dan *child process*. Kedua proses ini kemudian melanjutkan eksekusi dari baris setelah fork(). Akibatnya, rangkaian printf berikutnya akan dicetak dua kali, masing-masing oleh parent dan child. Setiap proses menampilkan teks yang sama, namun nilai yang ditampilkan oleh getpid() berbeda karena setiap proses memiliki Process ID (PID) yang unik. Dengan demikian, program ini mendemonstrasikan cara kerja fork() dalam menciptakan dua proses independen yang dieksekusi secara bersamaan oleh sistem operasi.

2. Lab2.c



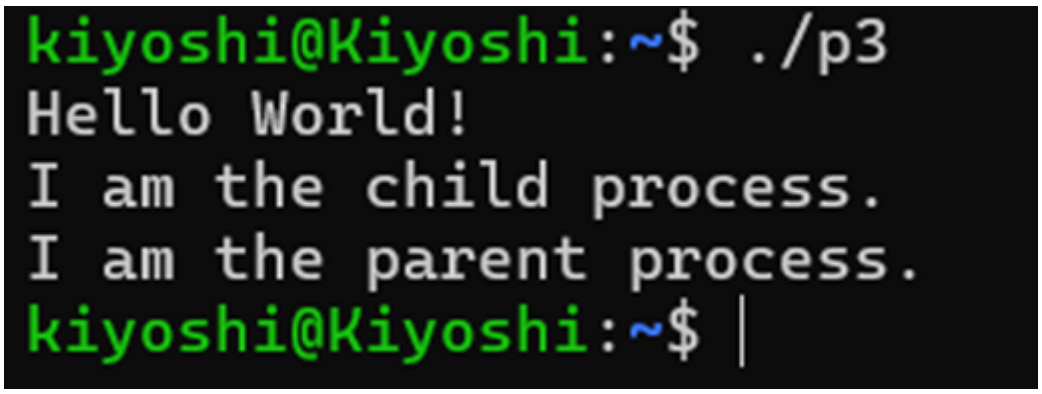
Program ini bertujuan untuk mengilustrasikan cara kerja *system call* fork() dalam menciptakan proses baru pada sistem operasi berbasis UNIX. Pada awal eksekusi, proses utama (parent process) menampilkan pesan identifikasi termasuk PID-nya dan menandakan posisi eksekusi sebelum pemanggilan fork(). Ketika fork() dipanggil, sistem operasi menduplikasi proses yang sedang berjalan sehingga terbentuk dua proses independen: proses parent dan proses child. Nilai pengembalian fork() disimpan dalam variabel pid. Pada proses child, pid bernilai 0, sedangkan pada proses parent, pid berisi PID dari child. Setelah pemanggilan fork(), kedua proses melanjutkan eksekusi kode yang sama dan menampilkan pesan bahwa mereka telah berada pada bagian setelah forking. Melalui percabangan if (pid == 0), program membedakan output antara child dan parent dengan mencetak PID masing-masing proses menggunakan getpid(). Hasil akhirnya menunjukkan bahwa setelah pemanggilan fork(), kedua proses berjalan secara paralel dan menjalankan instruksi yang sama namun sebagai entitas terpisah dengan PID yang berbeda, sehingga memperlihatkan mekanisme dasar manajemen proses dalam sistem operasi.

3. Lab3.c



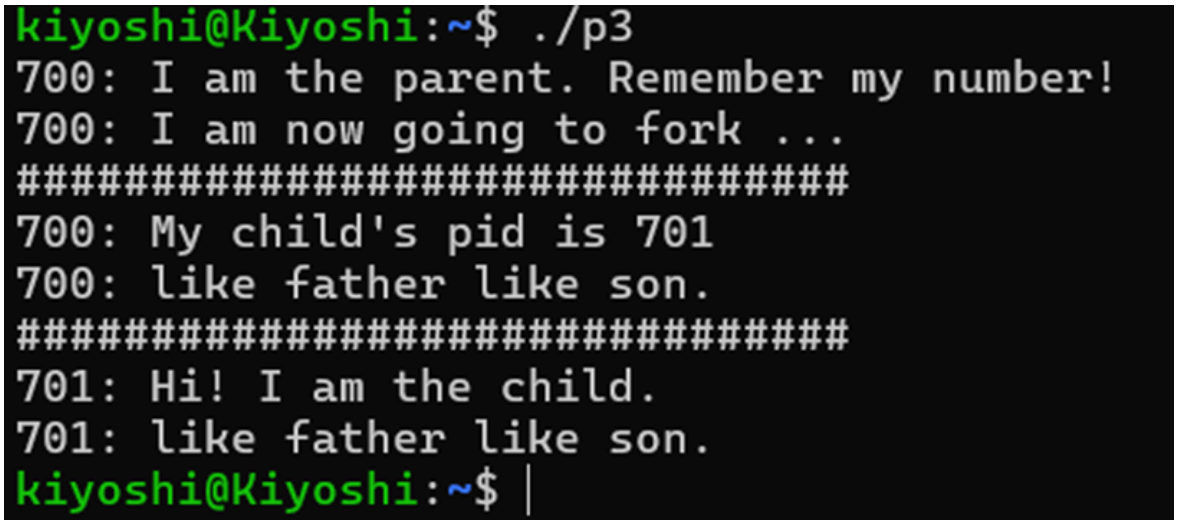
Program ini digunakan untuk mendemonstrasikan bagaimana pemanggilan fork() yang dilakukan lebih dari satu kali dapat menghasilkan multiplikasi proses dalam sistem operasi UNIX. Pada awal program, hanya satu proses yang mengeksekusi perintah dan mencetak pesan sebelum pemanggilan fork() pertama. Ketika fork() pertama dieksekusi, proses utama diduplikasi sehingga terdapat dua proses yang kini melanjutkan eksekusi ke baris berikutnya, masing-masing mencetak pesan bahwa mereka berada setelah forking pertama. Kedua proses tersebut kemudian mencapai pemanggilan fork() kedua, yang kembali menduplikasi masing-masing proses, sehingga total proses yang berjalan menjadi empat. Seluruh proses hasil duplikasi kemudian mengeksekusi perintah setelah forking kedua, mencetak pesan akhir serta PID-nya melalui getpid(). Dengan demikian, program ini memperlihatkan bahwa setiap pemanggilan fork() akan menggandakan jumlah proses yang ada, sehingga dua kali pemanggilan fork() menghasilkan $2^2 = 4$ proses yang berjalan paralel dan mengeksekusi instruksi yang sama tetapi sebagai entitas terpisah dengan PID unik.

4. Lab4.c



Program ini bertujuan untuk memperlihatkan mekanisme sinkronisasi antara *parent process* dan *child process* menggunakan *system call* wait(). Pada awal eksekusi, program mencetak pesan pembuka, kemudian memanggil fork() untuk menduplikasi proses. Jika fork() gagal, program akan menampilkan pesan kesalahan menggunakan perror() dan menghentikan eksekusi. Ketika fork() berhasil, proses child (ditandai dengan nilai pid == 0) akan langsung mengeksekusi perintah untuk mencetak pesan bahwa ia adalah proses anak. Sebaliknya, proses parent akan menjalankan wait(&status) yang menyebabkan parent berhenti sementara dan menunggu hingga child selesai sepenuhnya. Setelah child menyelesaikan tugasnya, parent melanjutkan eksekusi dan mencetak pesan bahwa ia adalah proses induk. Melalui alur ini, program menunjukkan bahwa dengan menggunakan wait(), proses parent dapat memastikan child mengeksekusi dan selesai lebih dulu sehingga urutan output menjadi teratur dan tidak tumpang tindih, yang merupakan konsep dasar sinkronisasi proses dalam sistem operasi berbasis UNIX.

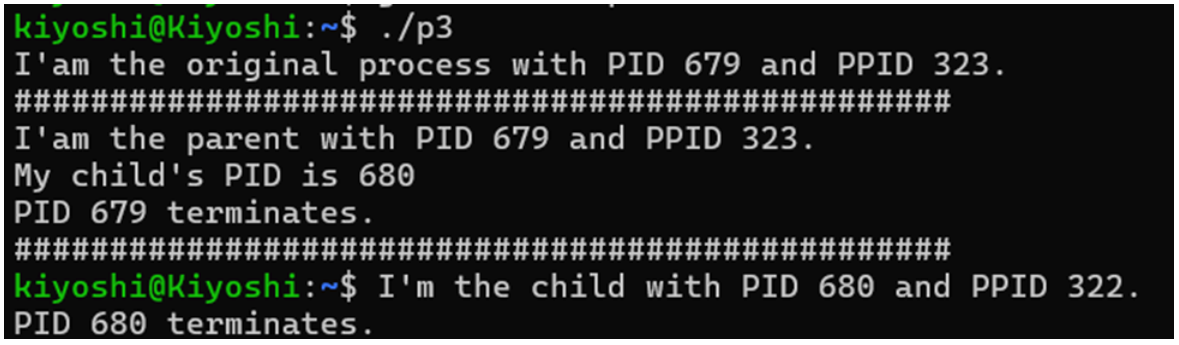
5. Lab5.c



Program ini digunakan untuk menggambarkan bagaimana proses parent dan child berperilaku setelah pemanggilan fork() pada sistem operasi UNIX. Pada awal eksekusi, proses utama (parent process) mencetak PID-nya sebagai identitas dan menampilkan pesan

bahwa proses akan melakukan *forking*. Ketika fork() dipanggil, sistem operasi menciptakan proses baru yang merupakan duplikasi dari parent, dan nilai pengembalian fork() disimpan di variabel forkresult. Setelah forking, kedua proses—parent dan child—akan mengeksekusi baris kode yang sama dimulai dari instruksi setelah pemanggilan fork(). Jika forkresult tidak bernilai nol, berarti proses tersebut adalah parent dan ia mencetak PID proses anak yang baru dibuat. Sebaliknya, jika forkresult bernilai nol, proses tersebut adalah child dan mencetak pesan identitasnya beserta PID sendiri melalui getpid(). Kedua proses kemudian mengeksekusi baris terakhir yang sama, menunjukkan bahwa setelah forking, parent dan child berjalan secara independen namun pada jalur eksekusi program yang identik. Program ini secara jelas mendemonstrasikan pemisahan eksekusi antara parent dan child pasca fork(), yang merupakan konsep fundamental dalam manajemen proses pada sistem operasi.

6. Lab6.c



Program ini bertujuan untuk mendemonstrasikan hubungan antara *parent process* dan *child process* menggunakan fork() serta bagaimana perubahan *parent process ID* (PPID) dapat diamati setelah proses induk berakhir. Pada awalnya, proses asli mencetak PID dan PPID-nya. Ketika fork() dipanggil, proses tersebut diduplikasi menjadi dua: parent dan child. Jika nilai pengembalian fork() tidak nol, proses tersebut adalah parent, dan ia mencetak informasi identitasnya serta PID proses anak. Sebaliknya, jika nilai pengembalian fork() adalah nol, proses tersebut merupakan child. Pada bagian child, terdapat *delay* dengan sleep(4) yang memastikan parent telah selesai dan benar-benar terminasi sebelum child melanjutkan eksekusi. Hal ini membuat child nantinya akan menunjukkan PPID baru, biasanya PID dari proses init/systemd, karena OS otomatis mengadopsi proses anak yang kehilangan induknya. Setelah masing-masing menyelesaikan bagiannya, baik parent maupun child mencetak pesan bahwa proses dengan PID tertentu akan terminasi. Program ini secara jelas menunjukkan bagaimana fork() membentuk dua proses independen, bagaimana perbedaan eksekusi terjadi berdasarkan nilai pengembalian fork(), serta bagaimana PPID pada child dapat berubah ketika parent sudah berakhir lebih dahulu.