Implementasi Algoritma Pencocokan String dan backtracking untuk masalah penyelesaian dependensi pada instalasi paket Linux

Catur Wirawan Wijiutomo.13505020

Sekolah Teknik Elektro dan Informatika
Program Studi Teknik Informatika
Institut Teknologi Bandung
e-mail: if15020@students.if.itb.ac.id

ABSTRAK

Sistem operasi modern seperti Linux telah mengizinkan user untuk menambah sendiri program-program tambahan. Akan tetapi, instalasi di linux menjadi sangat istimewa, kareana adanya kegiatan untuk mengumpulkan semua semua pustaka pada suatu direktori yang telah disepakati. Di Linux program tidak membawa dependency sendiri apalagi menggunakan lokasi ala *program files* seperti *Windows*. Hal ini dimaksudkan untuk menghindari *redudancy* pustaka dan mengurangi terjadinya ketidakcocokan versi pustaka untuk program. Namun hal ini menyebabkan permasalahan bagi pengguna awam Linux. Karena itu perlu dibuat suatu program yang mengotomatisasi penyelesaian dependensi program di Linux saat instalasi.

Kata kunci: RPM,Bash,Dependensi,Distro,KMP

1. PENDAHULUAN

Saat ini Linux merupakan salah satu sistem operasi yang disukai tidak hanya oleh para teknisi tetapi para pengguna akhir sebagai pelengkap PC desktop atau *notebook.* Kinerjanya pun tak kalah dengan sistem operasi Windows. Akan tetapi sampai saat ini pun penetrasi Linux di kalangan pengguna awam tidaklah terlalu baik. Banyak masalah di Linux harus diselesaikan bak teknisi.
Salah satu alasan pengguna akhir tidak mau memakai Linux karena instalasi programnya yang sulit. Kesulitan utama yang sering dialami adalah linux meminta penggunanya untuk menyelesaikan masalah dependensi pada instalasi program. Tidak seperti *windows* yang membawa sendiri pustakanya pada paket instalasi. Alasan metode ini telah dijelaskan diabstrak.
Contoh masalah dependency ini adalah, jika kita menginstall suatu paket intalasi Linux, misalnya *bluefish.rpm*. sistem Linux akan mengecek bila file dependensi belum terdapat di pustaka utama linux. Linux akan meminta pengguna menginstall pustaka tersebut terlebih dahulu dan instalasi tidak dapat dilanjutkan tanpa pengguna terlebih dahulu menyelesaikan masalah dependensi ini. Misalnya *gtk.rpm*. tetapi terkadang bisa saja file dependensi tersebut juga membutuhkan pustaka yang lain. Terlebih lagi pada distro Linux tertentu tidak disebutkan paket tetapi file pustaka yang diperlukan padahal user hanya mempunyai file paket yang terkompres. Kasus terburuk pengguna harus melihat secara manual di tiap paket dengan mengekstrak terlebih dahulu, tentu saja masalah seperti ini terkesan rumit bagi pengguna linux yang awam.
Bahkan untuk Linux yang paling baru sekalipun jika kita ingin menginstall suatu paket yang biasanya datang dalam bentuk file *tar.gz*, *.rpm*, *.tgz*, dll. Kita masih harus berhadapan dengan masalah dependensi ini.
Penulis mencoba untuk menyelesaikan masalah ini dengan merancang suatu program yang mengimplementasi algoritma DFS pada suatu *scripting languange* di Linux yaitu *Bash*.
Bash (Bourne Again Shell) sendiri adalah salah satu jenis shell, shell sendiri adalah suatu program perantara antara sistem operasi dengan pemakai. *Bash* sendiri dipakai untuk dua tujuan sebagai *user environment* dan salah satu *programming enviroment* di Linux..salah satu alasan pemrogram *shell* yang digunakan adalah Linux secara tradisional melakukan instalasi program yang dijalankan di atas *shell*.

Gambar 1. shell bash

2. METODE

2.1 perintah Instalasi Linux

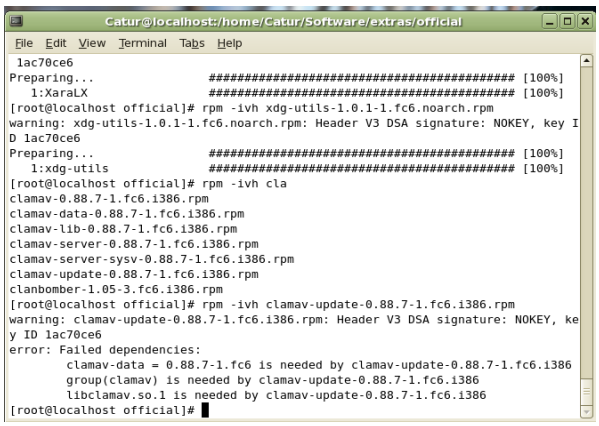
Pada makalah ini platform Linux yang digunakan adalah *Fedora Core 6*. Bash dapat dijalankan dari menu *acessories->terminal* setelah ini muncul sebuah jendela seperti *command-prompt* di *windows*, kalangan Linux menyebutnya console. Banyak konfigurasi Linux harus diatur melalui jendela ini. Karena sistem administrasi linux yang cukup solid ada beberapa perintah di konsole yang tidak bisa dijalankan jika kita masuk sebagai user biasa. Untuk melakukan instalasi program pengguna harus masuk terlebih dahulu sebagai *root* atau *super user*, di Windows kita mengenal *account Administrator* , untuk masuk sebagai root jalankan perintah :

```
$su
Password:
#
```

Setelah memasukkan password, maka kita dapat menjalankan perintah-perintah yang dibutuhkan untuk melakukan instalasi program. Di Fedora Core 6 paket instalasi yang dipakai adalah .rpm. perintah yang dipakai sama dengan nama ekstensinya. untuk melakukan instalasi kita memasukkan perintah:

```
#rpm –ivh nama_paket.rpm
```

Setelah perintah ini, akan muncul pesan-pesan kesalahan masalah dependensi yang tak terpecahkan. Jika pustaka tidak ditemukan di sistem linux. Maka Bash akan menampilkan pesan-pesan kesalahan yang menyeburtkan file-file yang harus diinstall terlebih dahulu.



Gambar 1. pesan kesalahan dalam instalasi

2.2 Standar I/O *Redirection* di Bash

Pesan-pesan kesalahan yang dikirmkan oleh bash akan dibutuhkan untuk membentuk himpunan solusi pada algoritma. Untuk menangkap pesan ini dipakai fitur I/O *redirection* pada bash perintah-perintah nya dapat dillihat di tabel 1:

Tabel 1 Redirection

Perintah	Keterangan
1> file	Redirect standard output ke file
2> >file	Redirect dan menerima standard output ke file
2>	Redirect standard error ke file
2>>	Redirect dan menerima standard error ke file
&>	Redirect standard output dan standard error ke file
2>&1	Redirect standard error ke standard output
i>&j	Redirect file deskriptot i ke j
>&j	Redirect file deksriptor 1 (standard output) ke j
J<>file	Redirect input dari file dan hasilnya dikembalikan ke file

Redirection yang dipakai adalah *redirect standar error* ke file. Dengan *redirection* ini maka pesan kesalahan yang biasanya ditampilkan ke layar akan dialihkan ke file yang kita sertakan setelah perintah *redirection.*

```
#rpm –ivh nama_paket.rpm 2> depedensi.txt
```

Dengan perintah tersebut maka semua pesan kesalahan akan dialihakan ke file dependesi.txt, jika kita membuka file ini maka kita akan membaca semua pesan kesalahan yang biasa kita lihat di layar.
Selain *error redirection* kita juga bisa memakai redirect standar *output* ke *file*. *Redirection* ini dipakai untuk *file* apa saja yang dibawa suatu paket .rpm. karena algoritma kita akan melakukan pengecekan ke semua file paket kandidat *resolver*. Perintah rpm menyediakan suatu perintah untuk menampilkan ke layar file-file yang ada di dalam paket.

```
#rpm –qp –requires file_paket.rpm 1>file.txt
```

Akan tetapi cara ini kurang praktis, karena program yang dibuat akan merespon dari kesalahan user. Bukan meminta user melakukan pengecekan file-file yang dibutuhkan.

2.3 Bash Script

File yang berisi pesan-pesan kesalahan pada proses instalasi mengandung nama-nama file yang dibutuhkan

pada instalasi. Untuk mengambil nama-nama file ini dibuat sebuah *daemon* dengan masukan parameter file yang memanfaatkan perintah *grep* sebuah perintah yang dapat mengerti masukan *regular expression* dan mengekstrasi nama-nama file dependensi yang biasanya berakhiran .so. nama file ini akan diredireksi lagi ke sebuah file nama_depend.txt. isi dari *daemon* tersebut:

```
#!/bin/bash
egrep so$ 1> $0
```

script bash ini disimpan dalam file dengan ekstensi .sh dan dijalankan di bash dengan mengetikkan

```
$./nama_file.sh
```

Kelebihan dari bash adalah kita dapat menyimpan serangkaian rutin perintah ke dalam file. Contohnya pada daemon yang dibuat *egrep* adalah sebuah perintah standar di konsole untuk menampilkan ke layar baris yang mengandung suatu string. Kelebihan *egrep* dapat diikuti metakarakter yang berfungsi seperti regular expression.

Tabel 2 metakarakter pada grep

Perintah	Keterangan
^	Diawali oleh kata setelah tanda ini
$	Diakhiri oleh kata sebelum tanda ini
.	Mewakili tepat satu karakter
*	Mewakili banyak karakter
[]	Mencari kata yang mengandung karakter didalamnya
{^}	Mencari kata yang tidak mengandung karakter didalamnya
+	Menggantikan tambahan karakter satu atau banyak
?	Menggantikan satu karakter atau tidak ada karakter
|	Untuk menyatakan atau
()	Menyatakan grup karakter
x{m}	Pengulangan karakter
x{m,}	M pengulangan terakhir
x{m,n}	Pengulangan anatara m dan n
\w	Karakter alphanumeric
\W	Karakter no alphanumeric
\b	Mengandung tepat kata di dalamnya
\<	Menyatakan string kosng diawal kata
\>	Menyatakan string kosng diakhir kata

Algoritma pencocokan string akan dipakai untuk mencari nama file yang sesuai di dalam file nama_depend.txt dengan file dalam kandidat-kandidat paket penyelesaian.
Penyelesaian dependensi yang dibuat tidaklah otomatis mencari paket di seluruh direktori penyimpanan data. Untuk menyelesaikan dependesi ini program tidak membuat file dependensi, tetapi mencari file-file dependensi pada paket-paket yang kita sediakan. Tentu saja suatu paket bisa berakhir tidak dapat diinstalasi jika paketnya tidak ada di paket-paket yang kita telah sediakan.

2.3 Penerapan Algoritma pencocokan string

Pertama sekali kita perlu membuat dua buah himpunan yaitu solusi dan ruang kandidat solusi. Untuk itu dibutuhkan dua buah daemon. Satu *daemon* merespon dari kesalahan user lalu mengecek file yang dibutuhkan dan memasukkan nama file yang dibutuhkan ke dalam suatu file teks list. *Daemon* yang lain akan melakukan iterasi pembuatan daftar file yang ada pada seluruh paket yang terdapat pada direktori yang kita siapkan. Daftar *file* ini akan disimpan dalam sebuah file teks sebagai kandidat ruang solusi.
Setelah dua daemon bekerja kita akan memiliki dua file yang berisi himpunan solusi yaitu nama_dependesi.txt dan ruang pencarian solusi yaitu kandidat_paket.txt.
Satu persatu nama di nama_dependesi.txt akan dicocokkan dengan kandidat_paket.txt. jika bertemu yang cocok, program akan memproses paket yang memilki file tersebut dengan otomatis menjalankan perintah

```
#rpm –ivh nama_paket.rpm <2 nama_depensi1.txt
```

Dalam program tetap diberikan *redirection* karena bisa saja paket yang kita butuhkan ternyata membutuhkan file-file lain. Sehingga algoritma akan dijalankan secara rekursif sampai pada akhirnya program yang diininkan terinstalasi di sistem.
Permasalah adalah dalam pemilihan algoritma pencocokan string. Dalam algoritma KMP telihat lebih mangkus jika terdapat banyak pengulangan huruf dalam string. Sehingga didapatkan fungsi pinggiran yang bernilai besar. Akan tetapi mayoritas file-file di Linux memiliki nama yang kemunculan suatu huruf yang sama cukup sering. Sehingga kompleksitas waktunya akan kurang lebih sama dengan dengan metode brute force.

2.4 Optimalisasi pencarian dengan menerapkan algoritma *backtracking*

Algoritma pencocokan string dipakai saat mengecek apakah terdapat nama file yang dicari dalam suatu kalang. Tentu saja dengan metode ini masalah kecepatan proses akan dikorbankan karena program mengecek sebanyak file kandidat dikali mengecek sebanyak huruf kandidat.

Untuk itu diperlukan sebuah algoritma *searching* yaitu *Backtracking* dengan ruang solusi berupa huruf-huruf pada nama file. Fungsi pembangkit akan mengiterasi huruf-huruf. Agar algoritma berjalan seakan mencari ruang solusi maka algoritma pencocokan string diletakkan pada fungsi pembatas.yang akan mengecek perhuruf. Jika ada satu huruf yang tidak cocok maka simpul tersebut

dimatikan dan tidak diperhitungkan kembali dalam pencarian.

3. KESIMPULAN

Kesulitan utama dalam instalasi program di Linux dapat diatasi dengan membuat sebuah program *resolver* otomatis. Program *resolver* ini membutuhkan suatu algoritma pencocokan string yang mencari nama-nama file dan mengeksekusi nama ile yang cocok untuk secara otomatis terinstall. Algoritma pencocokan string yang dipakai adalah *brute force* dan KMP. Penggunaan algoritma KMP akan berdampak tidak jauh berbeda dengan *brute force*, karena nama-nama file di linux yang jarang memilki pengulangan huruf didalamnya. Untuk itu dibutuhkan algoritma pencocokan string lain yang lebih mangkus.

Penggunaan algoritma backtracking digunakan untuk mempercepat waktu pencarian. Dengan ruang solusi adalah huruf-huruf dari nama file yang dicari, maka jika saat mencocokkan terjadi ketidakcocokan. Simpul akan dibuang dan tidak diperhitungkan lagi dalam pencarian. Cara ini akan mempercepat waktu pencarian.

Bash Scripting memberikan kelebihan dalam akses ke perintah-perintah dasar di Linux seperti grep, rpm dan lainya.akan tetapi tidak memberikan keleluasaan layaknya suatu bahasa pemrograman. Bahkan ada beberapa hal dalam pemrograman yang tidak boleh dipaksakan dalam Bash yaitu:

- *resourc*e sistem- tugas yang sangat intensif dimana kecepatan adalah sebuah faktor.
- *Cross platform* karena hanya ada di UNIX
- Kemanan data
- Proyek yang memilki subkomponen dengan dependensi yang saling mengunci
- *Array* multi-dimensi
- Struktur data kompleks
- Menipulasi *GUI*
- *Closed source*

Tetapi linux juga menyediakan alternatif lain dalam scripting yang lebih memilki fitur layaknya bahasa pemrograman tingkat tinggi. Seperti *perl, python*, dan *tcl*. Pemilihan bash dikarenakan kemudahannya dan interaksi awal pengguna di Linux dalam masalah konfigurasi adalah dengan *shell, shell* utama di Linux adalah *bash*..

Walaupun masih terdapat banyak kesulitan dalam instalasi. Pengembang Linux di seluruh dunia mulai memikirkan solusi untuk masalah ini. Agar Linux dapa diterima oleh pengguna yang awam sekalipun. Dan menciptakan suatu sistem operasi *open source* yang menjangkau semua lapisan user denagan beragam *skill*. Makalah ini adalah salah satu wujud kontribusi penulis dalam dunia *open-source*.

REFERENSI

Noprianto,"instalasi program di linux", InfoLinux, 05,2007,30

Mendel, Cooper. "Advanced Bash Scipting Guide".http://personal.riverusers.com/~thegrendel/abs-guide-1.0.tar.gz, 2001

Munir,Rinaldi "Strategi Algritmik",Program Studi Teknik Informatika ITB,2007

Sofyan,Ahmad. "membuat Distro Linux sendiri" Dian Rakyat,2006

Yuliadi,Rofiq "Bash Scripting–untuk administrasi sistem linux", PT Elex Media Komputindo:Jakarta,2003