Penerapan Algoritma DFS dan BFS pada Perintah *Shell* ‘find’ dalam Sistem File FAT32 Berdasarkan Modul Tugas Besar Sistem Operasi IF2230

Panji Sri Kuncara Wisma - 13522028[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522028@std.stei.itb.ac.id

Abstract*—Makalah ini mengkaji mengenai penerapan algoritma *Depth-First Search* (DFS) dan Breadth-First Search (BFS) pada perintah *shell* ‘find’ dalam penelusuran struktur folder pada sistem file FAT32 dan merupakan salah satu modul yang harus dikerjakan dalam tugas besar sistem operasi IF2230. Tugas besar tersebut mengharuskan mahasiswa membuat sistem operasi sendiri dan membuat perintah *shell*. Sistem file FAT32 yang masih cukup sering digunakan pada berbagai perangkat penyimpanan dan memiliki struktur direktori yang kompleks, memerlukan teknik penelusuran yang efisien untuk manajemen dan pencarian berkas. Algoritma DFS dan BFS adalah dua metode penelusuran graf yang umum digunakan dalam ilmu komputer untuk menjelajahi dan mencari struktur data seperti pohon dan graf. Dalam konteks FAT32, algoritma ini diterapkan untuk mengakses dan memanipulasi folder dan file secara efisien. Makalah ini menjelaskan bagaimana masing-masing algoritma dapat diimplementasikan dalam modul sistem operasi untuk meningkatkan kinerja penelusuran folder. Hasil penelitian menunjukkan bahwa dengan beberapa penyesuaian algoritma DFS dan BFS dapat diterapkan pada perintah *find* untuk penelusuran folder pada sistem file FAT32.

***Keywords*— FAT32, DFS, BFS, penelusuran folder, sistem file, algoritma penelusuran, struktur direktori**

I. PENDAHULUAN

Sistem File FAT32 merupakan salah satu sistem file yang masih cukup banyak digunakan pada berbagai perangkat penyimpanan seperti hard drive, flash drive, kartu memori, dan sistem operasi sederhana buatan sendiri. Sistem file ini memiliki kompatibilitas yang luas dan menjadi metode pengelolaan file utama yang diterapkan dalam tugas besar mata kuliah sistem operasi IF2230. Struktur direktori FAT32 cukup kompleks sehingga memerlukan teknik penelusuran yang efisien untuk mengelola dan mencari file secara efektif. Oleh karena itu, penerapan algoritma penelusuran *Depth-First Search* (DFS) dan *Breadth-First Search* (BFS) layak untuk dipertimbangkan dalam konteks ini.

Algoritma DFS dan BFS adalah dua metode penelusuran graf yang umum digunakan dalam *Computer Science*. DFS menelusuri graf dengan menjelajahi setiap cabang hingga kedalaman tertentu sebelum beralih ke cabang berikutnya, sementara BFS menelusuri graf dengan menjelajahi semua tetangga terdekat terlebih dahulu sebelum bergerak lebih jauh. Kedua algoritma ini memiliki kelebihan dan kekurangan masing-masing, tergantung pada struktur data yang dihadapi dan kebutuhan spesifik dari penelusuran tersebut.

Sistem file FAT32 memiliki struktur direktori yang hierarkis menyerupai *tree* atau pohon, dimana direktori *root* berfungsi sebagai akar dan setiap folder atau file sebagai *node*. Setiap *node* ini dapat mengarah ke beberapa *node* lainnya, menciptakan jaringan yang kompleks. Terkait penerapan sistem file FAT32 secara detail, makalah ini akan menggunakan modul tugas besar sistem operasi sebagai pedoman utama yang menggunakan struct dan linked list sebagai struktur data utama.

Dalam konteks pengembangan sistem operasi, implementasi algoritma DFS dan BFS paling jelas dapat dilihat pada salah satu perintah *shell* yang disebut *find*. Perintah *find* akan diikuti oleh nama folder dan melakukan pencarian terhadap folder tersebut dari direktori *root*. Perintah ini akan menjadi sangat relevan pada sistem file FAT32 dan algoritma DFS atau BFS karena struktur direktori FAT32 yang hierarkis dan kompleks.

Makalah ini bertujuan untuk mengkaji penerapan algoritma DFS dan BFS dalam penelusuran direktori pada sistem file FAT32. Hasil kajian diharapkan dapat memberikan wawasan kepada pembaca khususnya mahasiswa yang akan mengambil mata kuliah sistem operasi IF2230 agar dapat mengerjakan tugas besar sistem operasi modul *shell* dan *filesystem* dengan lebih mudah.

II. LANDASAN TEORI

A. Algoritma DFS

Depth-First Search (DFS) atau pencarian mendalam adalah salah satu algoritma pencarian graf yang digunakan untuk menjelajahi dan mencari simpul atau *edge* dalam sebuah graf. DFS dimulai dengan memilih satu simpul acak dan menandainya sebagai “dikunjungi”. Pada setiap langkah, algoritma ini mengunjungi simpul yang belum dikunjungi dan bersebelahan dengan simpul yang sedang diperiksa. Proses ini berlanjut sampai menemui jalan buntu, yaitu saat tidak ada lagi simpul tetangga yang belum dikunjungi. Kemudian, algoritma akan melakukan *backtracking* ke simpul sebelumnya untuk

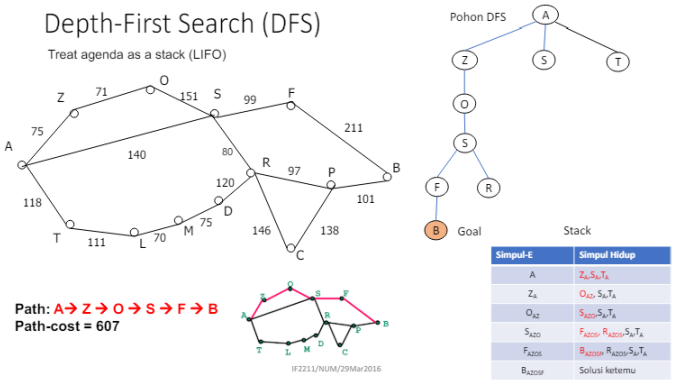
mencoba mengunjungi simpul yang belum dikunjungi dari sana.[1][2]

DFS akan berhenti setelah mundur kembali ke simpul awal dan menemui jalan buntu. Pada titik ini, semua simpul dan komponen graf yang sama dengan simpul awal sudah dikunjungi. Jika masih ada simpul yang belum dikunjungi, pencarian harus dimulai lagi dari salah satu simpul di komponen graf lainnya.[1][2]

Algoritma DFS menggunakan struktur data *stack* dengan prinsip LIFO (*Last-In First-Out*). *Stack* digunakan untuk melacak simpul yang harus dikunjungi kembali setelah mencapai jalan buntu. Implementasi DFS juga dapat menggunakan rekursif yang secara implisit menggunakan *stack* rekursif.[1][2]

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis ke layar
}
Deklarasi
   w : integer

Algoritma:
   write(v)
   dikunjungi[v]←true
   for w←1 to n do
      if A[v,w]=1 then {simpul v dan simpul w bertetangga }
         if not dikunjungi[w] then
            DFS(w)
         endif
      endif
   endfor
```

Gambar 1 *Pseudocode* Umum Algoritma DFS
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf)



Gambar 2 Ilustrasi Algoritma DFS
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf)

B. Algoritma BFS

Breadth-First Search (BFS) atau pencarian melebar adalah salah satu algoritma pencarian graf yang digunakan untuk menjelajahi dan mencari simpul atau *edge* dalam sebuah graf. BFS bekerja dengan mengeksplorasi simpul-simpul graf secara bertahap, dimulai dari simpul awal, kemudian mengunjungi semua simpul tetangga terdekat sebelum berpindah ke simpul-simpul yang lebih jauh. Pendekatan ini menjadikan algoritma BFS terlihat "berhati-hati" dibandingkan dengan algoritma DFS yang cenderung "berani" dalam menjelajahi simpul-simpul yang lebih dalam terlebih dahulu. Algoritma BFS sangat berguna dalam menemukan jalur terpendek pada graf tak berbobot.[1][2]

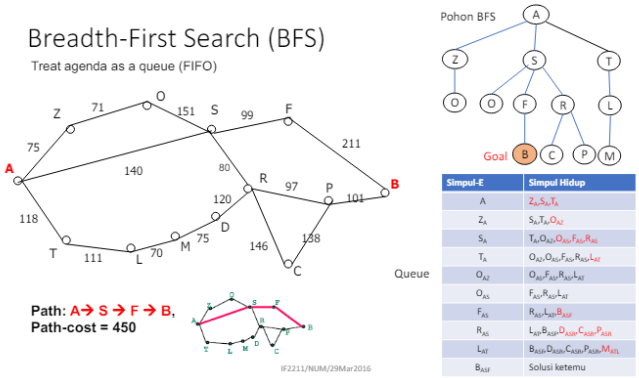
Inti dari algoritma BFS adalah penggunaan struktur data *queue* (antrean) yang beroperasi dengan prinsip *First-In, First-Out* (FIFO). Proses dimulai dengan menandai simpul awal sebagai dikunjungi dan memasukannya ke dalam *queue*. Kemudian, simpul di depan *queue* diambil, dan semua simpul tetangga yang belum dikunjungi dimasukkan ke dalam *queue*. Simpul-simpul tersebut kemudian ditandai sebagai dikunjungi untuk mengurangi pengulangan. Proses ini terus berlanjut hinga *queue* kosong atau simpul tujuan telah ditemukan. [1][2]

Keuntungan utama dari BFS adalah kemampuannya untuk menemukan jalur terpendek dalam graf tidak berbobot, karena BFS memastikan bahwa simpul yang lebih dekat dengan simpul awal dijelajahi terlebih dahulu. Hal ini membuat BFS sangat berguna dalam aplikasi pencarian rute di peta, penyelesaian puzzle, dan penyelesaian masalah yang melibatkan pemrograman graf. BFS juga dapat digunakan untuk menentukan apakah sebuah graf terhubung atau tidak dan untuk mencari semua komponen yang terhubung dalam sebuah graf.[1][2]

BFS juga memiliki beberapa kelemahan. Salah satu kelemahan utama adalah penggunaan memori yang tinggi, terutama pada graf dengan banyak simpul dan tepi karena semua simpul yang belum dieksplorasi perlu disimpan dalam antrian. Selain itu, BFS juga tidak efisien untuk graf dengan kedalaman yang sangat besar atau masalah yang memerlukan eksplorasi lebih dalam sebelum solusi ditemukan.[1][2]

```
procedure BFS(input v:integer)
{ Traversal graf dengan algoritma pencarian BFS.

  Masukan: v adalah simpul awal kunjungan
  Keluaran: semua simpul yang dikunjungi dicetak ke layar
}
Deklarasi
   w : integer
   q : antrian;

   procedure BuatAntrian(input/output q : antrian)
   { membuat antrian kosong, kepala(q) diisi 0 }

   procedure MasukAntrian(input/output q:antrian, input v:integer)
   { memasukkan v ke dalam antrian q pada posisi belakang }

   procedure HapusAntrian(input/output q:antrian,output v:integer)
   { menghapus v dari kepala antrian q }

   function AntrianKosong(input q:antrian) → boolean
   { true jika antrian q kosong, false jika sebaliknya }

Algoritma:
   BuatAntrian(q)          { buat antrian kosong }

   write(v)                { cetak simpul awal yang dikunjungi }
   dikunjungi[v]←true      { simpul v telah dikunjungi, tandai dengan
                             true}
   MasukAntrian(q,v)       { masukkan simpul awal kunjungan ke dalam
                             antrian}

  { kunjungi semua simpul graf selama antrian belum kosong }
  while not AntrianKosong(q) do
        HapusAntrian(q,v)   { simpul v telah dikunjungi, hapus dari
                              antrian }
        for tiap simpul w yang bertetangga dengan simpul v do
            if not dikunjungi[w] then
               write(w)     {cetak simpul yang dikunjungi}
               MasukAntrian(q,w)
               dikunjungi[w]←true
            endif
     endfor
  endwhile
  { AntrianKosong(q) }
```

Gambar 3 *Pseudocode* Umum Algoritma BFS
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf)



Gambar 4 Ilustrasi Algoritma DFS
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf)

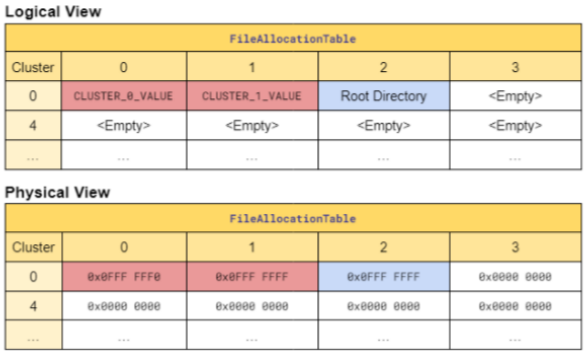
C. FAT32

Istilah	Definisi dalam konteks FAT32
Direktori	Sinonim dengan folder yaitu sebuah struktur yang digunakan untuk mengkatalogkan file dan direktori lain. Struktur ini akan menyimpan referensi ke file dan folder lain.
File	Sekelompok data yang memiliki identifier dan metadata. File adalah unit penyimpanan data terkecil pada sistem file.
Metadata	Data yang mendeskripsikan file seperti nama file, ukuran file, atribut, dan lain-lain.
Cluster	Unit penyimpanan data terkecil pada sistem file.
Block	Unit penyimpanan data terkecil pada *storage device*.

FAT32 merupakan sistem file yang menggunakan sebuah tabel bernama *File Allocation Table* untuk menyimpan informasi alokasi. Sementaara itu metadata akan disimpan pada tabel khusus milik direktori. Setiap sistem file FAT32 akan memiliki *File Allocation Table* yang telah dibuat sejak awal pembuatan sistem file FAT32, misalnya pada tahap formatting flashdisk. Struktur data ini akan digunakan untuk semua operasi *read* dan *write* pada sistem file FAT32.[3]

Dua struktur yang menjadi *building block* sistem file adalah file dan direktori. Hanya dua struktur ini yang bersifat transparan ke pengguna. Pengguna hanya dapat melihat dan memanipulasi file dan direktori melalui antarmuka yang disediakan. Semua file dan folder akan memiliki *Ancestor Node* yang sama dengan *root*. Pembacaan sistem file FAT32 dimulai dengan membaca direktori *root* yang akan memiliki subdirektori dan hubungan ke semua file dan folder lain.[3]

D. File Allocation Table



Logical View

FileAllocationTable

Cluster	0	1	2	3
0	CLUSTER_0_VALUE	CLUSTER_1_VALUE	Root Directory	<Empty>
4	<Empty>	<Empty>	<Empty>	<Empty>
...	...	...	...	...

Physical View

FileAllocationTable

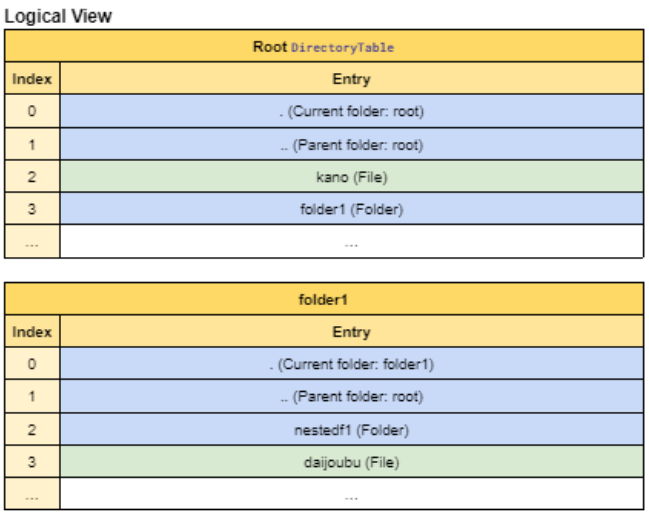
Cluster	0	1	2	3
0	0x0FFF FFF0	0x0FFF FFFF	0x0FFF FFFF	0x0000 0000
4	0x0000 0000	0x0000 0000	0x0000 0000	0x0000 0000
...	...	...	...	...

Gambar 5 Ilustrasi *File Allocation Table*
(Sumber:https://docs.google.com/document/d/1EafdqpKWpYpU08w8AmKrEDCedrh8PvnGJ3bJWZEeFPU/edit?usp=drive_link)

FAT32 menggunakan struktur data FAT untuk menyimpan informasi alokasi cluster kosong dan susunan linked list file. Setiap *entry* dari FAT berukuran 4 bytes (32-bit) yang direpresentasikan dengan 8 hexadecimal digit seperti pada gambar 5. Semua FAT yang valid akan memiliki 3 *reserved entry*. Cluster 0 dan 1 masing-masing akan diisi oleh suatu nilai konstanta. Cluster 2 akan menjadi direktori *root*. Cluster kosong akan bernilai FAT32_FAT_EMPTY_FILE_ENTRY (0x00000000). Pada FAT32, cluster 1 akan menyimpan *File Allocation Table* dan selalu dimodifikasi atau dibaca untuk semua operasi CRUD. Struktur data ini tidak menyimpan isi data dari sebuah file, tetapi menyimpan cluster-cluster partisi file yang dituliskan dengan cara linked list dengan pointer ke list selanjutnya adalah informasi nomor cluster.[3]

E. File dan Direktori

Direktori *root* berperan sebagai *ancestor node* semua file dan direktori. *Root* dijamin selalu ada dan berperan pada sistem file. Pembacaan awal untuk membentuk pohon direktori sistem file dimulai dari membaca direktori *root* terlebih dahulu. Setelah itu digunakan algoritma traversal rekursif ke semua sub direktori untuk mendapatkan satu level pohon dibawah hingga terminasi.[3]



Logical View

Root DirectoryTable

Index	Entry
0	. (Current folder: root)
1	.. (Parent folder: root)
2	kano (File)
3	folder1 (Folder)
...	...

folder1

Index	Entry
0	. (Current folder: folder1)
1	.. (Parent folder: root)
2	nestedf1 (Folder)
3	daijoubu (File)
...	...

Gambar 6 Ilustrasi Tabel Direktori
(Sumber:https://docs.google.com/document/d/1EafdqpKWpYpU08w8AmKrEDCedrh8PvnGJ3bJWZEeFPU/edit?usp=drive_link)

F. Shell

Shell adalah bagian terluar dari sistem operasi yang menggunakan layanan kernel untuk menyediakan antarmuka pengguna. *Shell* dijalankan pada mode pengguna dan tidak memiliki akses penuh ke *resource* sistem. Dalam konteks sistem file, *shell* membutuhkan bantuan *system calls* untuk melakukan operasi CRUD. *System calls* adalah cara utama kernal untuk menyediakan layanan I/O dan hardware kepada program pengguna.[3][4]

Find adalah salah satu perintah yang dapat diimplementasikan pada *shell. Find* akan menerima argumen nama folder atau file dan akan melakukan pencarian folder atau file tersebut dalam sistem file. Apabila ditemukan, *path* dari folder tersebut dengan referensi *root* akan ditampilkan. Penelusuran selalu dimulai dari *root.*[3]

III. ANALISIS ALGORITMA

A. Desain dan Batasan

Boot sector hanya menyimpan *fs_signature.* Cluster 1 digunakan secara penuh untuk FAT32 *File Allocation Table.* Cluster 2 digunakan untuk tabel direktori *root.* Dalam kaitannya dengan tabel direktori, folder (termasuk *root*) dijamin hanya 1 cluster. Tabel direktori disebut kosong jika dan hanya jika memiliki 2 *entry*. *Entry* 0 adalah direktori itu sendiri dan menyimpan nama dari direktori itu sendiri. *Entry* 1 adalah direktori *parent* dan harus menyimpan *parent* cluster yang sesuai.[3]

Khusus untuk *root*, *Entry* 1 akan diisi oleh informasi cluster dirinya sendiri. *Entry* disebut kosong jika dan hanya jika tidak memiliki flag UATTR_NOT_EMPTY pada user_attribute. *Entry* bertipe folder akan memiliki atribut flag ATTR_SUBDIRECTORY. Maksimum dari nama file atau folder adalah 8 karakter dan maksimum ukuran ekstensi adalah 3 karakter.[3]

B. Analisis Umum

Dalam konteks sistem file FAT32, *entry* dari tabel direktori akan dianggap sebagai simpul. *Entry* dari tabel direktori yang berupa file tidak memiliki tabel direktori di dalamnya. Oleh karena itu, file dianggap sebagai simpul yang tidak memiliki anak. Sementara itu, direktori dapat memiliki tabel direktori lagi di dalamnya, yang berarti suatu direktori dapat dianggap sebagai simpul yang memiliki anak-anak simpul dengan jumlah tertentu.

Setiap *entry* memiliki atribut cluster_high dan cluster_low yang masing masing merupakan *unsigned integer* 16 bit. Oleh karena itu nomor cluster dari file atau folder yang bersangkutan dapat diperoleh dengan operasi cluster_high << 16 | cluster_low.

Terkait dengan file, nomor cluster yang diperoleh dari operasi tersebut dapat digunakan sebagai garis awal untuk menjelajahi *File Allocation Table* untuk mendapatkan informasi cluster dari file yang sudah dipartisi dan mendapatkan isi atau data dari file tersebut. Terkait dengan folder, nomor cluster yang diperoleh juga dapat digunakan untuk mendapatkan tabel direktori dari folder yang dimaksud. Dengan kata lain, nomor cluster adalah penghubung antara suatu simpul dengan simpul yang lainnya.

Dalam modul tugas besar sistem operasi IF2230, ukuran satu block adalah 512 byte dan jumlah block dalam satu cluster ada 4. Oleh karena itu, ukuran satu cluster adalah 512 * 4 = 2048 byte. Diketahui juga ukuran satu *entry* dalam tabel direktori adalah 32 bytes, maka dalam satu tabel direktori maksimal akan berisi 2048 / 32 = 64 *entry*.

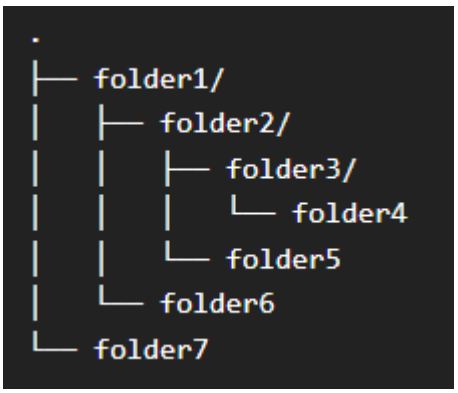
Dengan mengabaikan *entry 0* yang merujuk pada direktori itu sendiri dan juga *entry* 1 yang merujuk pada direktori *parent*, dapat diketahui bahwa jumlah anak paling banyak yang dimiliki oleh suatu simpul adalah 62.

C. Analisis dan Implementasi Algoritma DFS

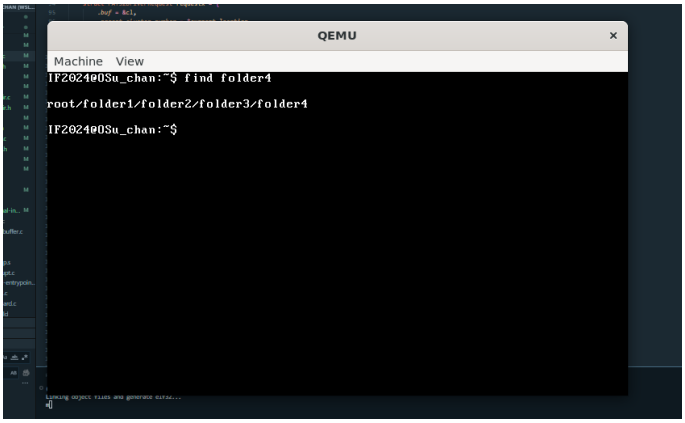
Implementasi DFS dalam FAT32 melibatkan beberapa langkah utama. Pertama, membuat buffer (ClusterBuffer cl) untuk menyimpan data sementara. Selanjutnya, membuat request (FAT32DriverRequest requestx) yang akan digunakan untuk membaca direktori, dan mengatur tabel direktori (FAT32DirectoryTable dir_table) untuk menyimpan *entry* direktori yang dibaca. Langkah berikutnya adalah menggunakan syscall untuk membaca *entry* direktori dari FAT32 dan menyimpannya dalam dir_table. Syscall ini mengirimkan request ke kernel untuk membaca data dari direktori yang ditentukan.

Jika *entry* yang dibaca adalah subdirektori (ATTR_SUBDIRECTORY), DFS dipanggil kembali dengan memperbarui current_location ke lokasi subdirektori tersebut. Ini memungkinkan DFS untuk menjelajahi setiap subdirektori secara mendalam sebelum kembali ke direktori sebelumnya. Jika *entry* yang dibaca cocok dengan nama file (nama) dan ekstensi (ext) yang dicari, maka pencarian dihentikan. Setelah file yang sesuai ditemukan, algoritma menandai pencarian sebagai berhasil dan menghentikan pencarian lebih lanjut.

Program diuji pada struktur folder berikut dengan tujuan folder4 :



Gambar 7 Struktur Folder
(Sumber:Dokumentasi Pribadi Penulis)



Gambar 8 Tangkapan Layar Pengujian DFS
(Sumber:Dokumentasi Pribadi Penulis)

Gambar 8 adalah tangkapan layar dari sistem operasi yang dibuat oleh penulis dan kelompok tugas besar sistem operasi IF2230. Dalam sistem operasi tersebut, perintah *shell find* yang diimplementasikan dengan menggunakan algoritma DFS berhasil dijalankan dan mendapatkan jalur sesuai ekspektasi.

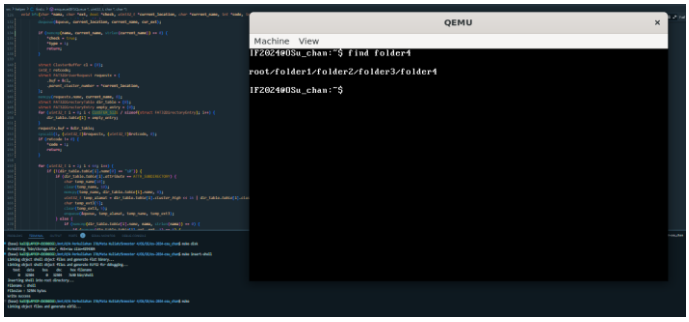
D. Analisis dan Implementasi Algoritma BFS

Implementasi BFS dalam FAT32 melibatkan beberapa langkah utama. Pertama, membuat struktur antrian (BFSQueue) untuk menyimpan lokasi direktori, nama direktori, dan ekstensi yang akan diperiksa. Antrian ini digunakan untuk menyimpan direktori yang akan dieksplorasi dan memastikan semua direktori pada level yang sama diperiksa sebelum melanjutkan ke level berikutnya.

Selanjutnya, fungsi BFS menginisialisasi antrian dengan direktori root dan memulai proses eksplorasi. Proses ini melibatkan dequeue direktori dari antrian, membaca *entry* direktori menggunakan request (FAT32DriverRequest requestx), dan menyimpan hasilnya dalam tabel direktori (FAT32DirectoryTable dir_table). Syscall digunakan untuk membaca *entry* direktori dari FAT32 dan menyimpannya dalam dir_table. Syscall ini mengirimkan request ke kernel untuk membaca data dari direktori yang ditentukan.

Jika *entry* yang dibaca adalah subdirektori (ATTR_SUBDIRECTORY), entry tersebut dimasukkan ke dalam antrian dengan memperbarui current_location ke lokasi subdirektori tersebut. Ini memungkinkan BFS untuk menjelajahi setiap subdirektori setelah semua direktori pada level yang sama telah diperiksa. Jika entry yang dibaca cocok dengan nama file (nama) dan ekstensi (ext) yang dicari, maka pencarian dihentikan. Setelah file yang sesuai ditemukan, algoritma menandai pencarian sebagai berhasil dan menghentikan pencarian lebih lanjut.

Program diuji pada struktur folder yang sama dengan kasus DFS. Berikut adalah hasilnya:



Gambar 9 Tangkapan Layar Pengujian BFS
(Sumber:Dokumentasi Pribadi Penulis)

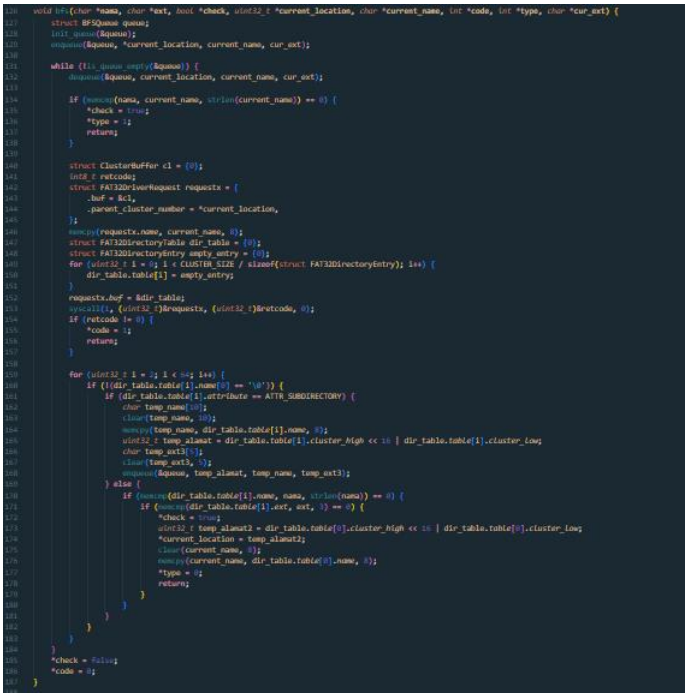
Gambar 9 adalah tangkapan layar dari sistem operasi yang dibuat oleh penulis dan kelompok tugas besar sistem operasi IF2230. Dalam sistem operasi tersebut, perintah *shell find* yang diimplementasikan dengan menggunakan algoritma BFS berhasil dijalankan dan mendapatkan jalur sesuai ekspektasi

IV. KESIMPULAN

Algoritma DFS dan BFS dapat diterapkan secara efektif dalam sistem file FAT32 untuk meningkatkan efisiensi penelusuran file dan folder. DFS lebih cocok untuk penelusuran mendalam, sementara BFS lebih efektif untuk penelusuran lebar dan menemukan jalur terpendek. Implementasi algoritma ini pada perintah shell 'find' membantu penelusuran file atau folder yang lebih baik dan berperan signifikan pada pengembangan sistem operasi yang merupakan bagian dari tugas besar IF2230.

V. KODE PROGRAM

- DFS
  https://github.com/PanjiSri/os-2024-osu_chan/blob/main/src/helper/find.c
- BFS



Gambar 10 Implementasi algoritma BFS
(Sumber:Dokumentasi Pribadi Penulis)

VI. UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan Yang Maha Esa karena berkat rahmat dan kasih karunia-Nya penulis dapat menyelesaikan makalah yang berjudul “Penerapan Algoritma DFS dan BFS pada Perintah Shell ‘find’ dalam Sistem File FAT32 Berdasarkan Modul Tugas Besar Sistem Operasi IF2230” dengan baik. Tak lupa juga penulis mengucapkan terima kasih kepada dosen-dosen pengampuh mata kuliah IF2211 terutama kepada Dr. Nur Ulfa Maulidevi, S.T., M. Sc. selaku dosen pengampuh mata kuliah IF2211 untuk kelas 02 karena telah memberikan pengetahuan yang dapat digunakan penulis untuk menulis makalah ini. Penulis berharap makalah ini juga bisa memberi manfaat baik bagi pelajar maupun masyarakat secara umum.

REFERENSI

[1] Munir, Rinaldi dan Nur Ulfa Maulidevi. Breadth/Depth First Search (BFS/DFS) (Bagian 1). (Diakses pada tanggal 12 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf

[2] Munir, Rinaldi dan Nur Ulfa Maulidevi. Breadth/Depth First Search (BFS/DFS) (Bagian 2). (Diakses pada tanggal 12 Juni 2024 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag2.pdf

[3] 13519214 dan Asisten Laboratorium Sistem Paralel dan Terdistribusi. Book I: Protected Mode x86 IF2230 - Sistem Operasi (Diakses pada tanggal 12 Juni 2024 dari https://docs.google.com/document/d/1EafdqpKWpYpU08w8AmKrEDCedrh8PvnGJ3bJWZEeFPU/edit#heading=h.aszi94vnjo5v

[4] Silberschatz, Abraham and Galvin, Peter B. and Gagne, Greg, Operating System Concepts, vol. 9. Wiley Publishing, 2012.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Panji Sri Kuncara Wisma
13522028