**OPERASI SISTEM DENGAN MENGGUNAKAN**

**BAHASA C DAN ASSEMBLER DI QEMU**

Rifqy Adli Damhuri1, Vika Putri Ariyanti2

Universitas Gunadarma

Jl. Margonda Raya No. 100 Depok

E-mail : [rifqy.adli@gmail.com](mailto:rifqy.adli@gmail.com)1, [vikaputriariyanti@gmail.com2](mailto:vikaputriariyanti@gmail.com2)

**Abstrak**

Pembuatan operasi sistem bukanlah hal mudah karena diperlukan sebuah pengetahuan tentang C dan Assembler. Dalam diperlukan metode pengujian alternatif agar tidak me-Reboot ulang setiap pengujian pada saat terjadi perubahan. Namun ada sistem operasi sederhana dengan menggunakan bahasa C dan Assembler serta Qemu.

**Kata Kunci** : Sistem Operasi, bahasa C, Assembler, Qemu

**1. Pendahuluan**

Sebuah sistem operasi merupakan bagian dari perangkat lunak untuk mengelola dan berinteraksi dengan hardware secara efektif dan memberikan shell untuk program lainnya untuk berjalan. Jadi, dengan adanya sistem operasi, komputer atau perangkat keras (hardware) dapat digunakan seefisien mungkin. Selain itu, tujuan dari sistem  operasi adalah agar komputer dapat bekerja dalam kode biner yang mudah dipahami dan digunakan oleh pemakai atau user. Dengan begitu membuat sistem operasi bukanlah yang mudah karena dalam pembuatan sistem operasi harus dapat digunakan secara efektif dan mudah dimengerti oleh user.

Tidak hanya dari sisi tujuan, namun dari sisi pengetahuan juga. Untuk dapat membuat sistem operasi diperlukan sebuah pengetahuan tentang bahasa C dan juga Assembler terutama di cara kompilasi, pointer, dan lain-lain.

Dalam pembuatan sistem operasi juga diperlukan sebuah pengujian dan tentunya perlu beberapa kali pengujian. Agar tidak selalu me-Reboot komputer setiap pengujian karena membuat perubahan, maka perlu sebuah metode alternatif untuk menguji sistem operasi baru yaitu dengan menggunakan mesin virtual. Banyak sekali jenis mesin virtual yaitu Virtual Box, Parallels, VMWare, QEMU.

Untuk itu, kami akan membuat sebuah operasi sistem sederhana menggunakan bahasa C dan Assembler yang diuji dengan menggunakan mesin virtual Qemu.

**2. Landasan Teori**

1. Sistem Operasi

Sistem operasi adalah sebuah program yang bertindak sebagai perantara (interface) antara pemakai komputer (user) dengan komputer (perangkat keras). Pengertian lain dari sistem operasi adalah sebuah perangkat lunak sistem komputer yang membantu perangkat keras dalam menjalankan fungsi-fungsi Manajemen Proses.

2. Bahasa C

Bahasa C merupakan perkembangan dari bahasa BCPL yang dikembangkan oleh Martin Richards, pada tahun 1967. Selanjutnya bahasa ini memberikan ide kepada Ken Thompson yang kemudian mengembangkan bahasa yang disebut bahasa B pada tahun 1970. Perkembangan selanjutnya dari bahasa B adalah bahasa C oleh Dennis Ritchie sekitar tahun 1970-an di Bell Telephone Laboratories Inc. (sekarang adalah AT&T Bell Laboratories). Bahasa C pertama kali digunakan di komputer Digital Equipment Corporation PD-11 yang menggunakan sistem operasi UNIX.

1. QEMU

QEMU adalah open source sebuah mesin emulator yang memungkinkan Anda untuk menjalankan hampir semua sistem operasi saat anda sedang berjalan di dalam distribusi linux lainnya.

1. Assembler

Assembler adalah bahasa pemrograman yang instruksinya menggunakan abjad dan tanda-tanda numerik, contohnya: Assembly.

1. Kernel

Kernel adalah komponen inti pada sistem operasi yang bertindak sebagai jembatan antara aplikasi pengolahan data yang dilakukan komputer. Kernel komputer menjadi jembatan tiga komponen utama perangkat keras komputer yaitu perangkat keras input, perangkat keras output, dan perangkat keras proses. Fungsi kernel adalah untuk melayani berbagai program aplikasi mengakses perangkat keras komputer, mengatur penjadwalan suatu program memakai bagian perangkat keras dan membantu eksekusi aplikasi melakukan abstraksi perangkat keras. Kernel bertanggung jawab melakukan proses aplikasi manajemen memori, alokasi, dan manajemen perangkat input dan system call control.

1. Loader

Loader berguna untuk sebagai pemicu awal ketika OS di booting awal. Loader ini akan dipanggil pertama kali.

1. LD

LD adalah sebuah aplikasi yang mengkombinasikan beberapa kode objek dan pustaka yang ada untuk menjadi sebuah file yang dapat dieksekusi. LD diambil dari kata load’, dan merupakan perkakas standar bagi sistem operasi berbasis UNIX GNU. Program termasuk ke dalam gcc.

1. GCC

GCC merupakan singkatan dari GNU C Compiler, yaitu compiler yang digunakan untuk mengkompilasi program yang ditulis dalam bahasa C.

**3. Metode Penelitian**

1. Instalasi GCC dan make digunakan untuk mengkompilasi kernel OS yang dibuat dengan bahasa C.
2. Instalasi Qemu untuk virtual machine yang digunakan untuk menjalankan OS yang dibuat
3. Membuat Bootloader

Bootloader ini digunakan untuk memilih sistem operasi tertentu dan memulai prosesnya dan kemudian sistem operasi memuat dirinya ke dalam memori.

Simpan kode program bootloader ini dengan nama boot.s. Kemudian kompilasi file boot.s untuk menghasilkan boot.o

1. Membuat Kernel OS

Kernel yang dibuat ini menggunakan bahasa pemrograman C yang terdiri dari kernel.C dan kernel.h untuk menghasilkan kernal.h

1. Membuat Source Code Pendukung Kernel yaitu types.h dan utils.h
2. Membuat Linking semua objek

Untuk menghubungkan semua objek boot.o dan kernel.o ke dalam linking digunakan ld berikut ini perintah cara mengkompilasi file linker.ld yang akan menghasilkan file MyOS.bin

1. Testing

Setelah kita melakukan linking dan mendapatkan file MyOS.bin, untuk menguji apakah OS yang dibuat bener-bener berhasil maka cukup dengan menjalankan melalui qemu.

**4. Hasil dan Pembahasan**

1. Bootloader

Hasil dari pengkompailan dari file boot.s adalah file boot.o

1. Kernel

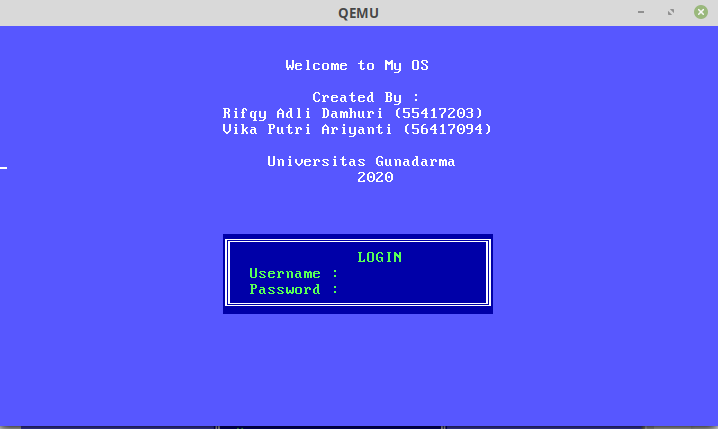
Hasil dari pengkompailan dari file kernel.c adalah file kernel.o

1. Linking Semua Objek

Hasil dari kompilasi bootloader dan kernel yang berupa boot.o dan kernel.o kemudian 2 file tersebut dihubungkan dengan menggunakan ld yang kemudian menghasilkan file yang berupa MyOS.bin

1. Testing

Untuk menguji apakah OS yang dibuat bener-bener berhasil maka dilakukan testing dengan menggunakan qemu dengan mengkompilasi file MyOS.bin dan hasilnya adalah sebagai berikut



**5. Kesimpulan**

Dari pembahasan di atas bahwa dapat disimpulkan bahwa membuat OS bukanlah hal yang sulit, namun untuk dapat membuat OS si pembuat harus memiliki dasar, bahkan harus paham paham tentang program assembler dan bahasa C. Selain itu, si pembuat harus memiliki dasar tentang command prompt atau perintah dasar terminal.

**6. Daftar Pustaka**

[1] Zope, Pritam. 2018. *Create Your Own Kernel In C* di <https://www.codeproject.com/Articles/1225196/Create-Your-Own-Kernel-In-C-2>

[2] Kurniawan, Agus Kurniawan. 2013. *Membuat Sistem Operasi Sederhana* di <http://blog.aguskurniawan.net/post/Membuat-Sistem-Operasi-Sederhana.aspx>

[3] Solichin, Achmad. 2003. *Pemograman Bahasa C dengan Turbo C* di [https://books.google.co.id/books?hl=en&lr=&id=9QgGBgAAQBAJ&oi=fnd&pg=PP2&dq=bahasa+c+adalah&ots=nmnIgZO9M1&sig=EmXtg23z4FYiDXiqcToGKJ4-SEI&redir\_esc=y#v=onepage&q=bahasa%20c%20adalah&f=false](https://books.google.co.id/books?hl=en&lr=&id=9QgGBgAAQBAJ&oi=fnd&pg=PP2&dq=bahasa+c+adalah&ots=nmnIgZO9M1&sig=EmXtg23z4FYiDXiqcToGKJ4-SEI&redir_esc=y" \l "v=onepage&q=bahasa c adalah&f=false)

[4] Zope, Pritam. 2018. *Create Your Own Kernel In C* di <https://www.c-sharpcorner.com/article/create-your-own-kernel/>.

[5] Zope, Pritam. 2019. *OS* di <https://github.com/pritamzope/OS>.

[6] Huda, Miftahul. TT. Teknologi Komputer : Pemanfaatan Teknologi Komputer Untuk Mempermudah Penyelesaian Tugas dan Pekerjaan Yang Dihadapi di [https://books.google.co.id/books?id=2DmnDwAAQBAJ&pg=PA28&dq=Kernel+pada+OS+adalah&hl=en&sa=X&ved=0ahUKEwjWyfLwiITnAhXQXSsKHUBRC\_MQ6AEIMzAB#v=onepage&q=Kernel%20pada%20OS%20adalah&f=false](https://books.google.co.id/books?id=2DmnDwAAQBAJ&pg=PA28&dq=Kernel+pada+OS+adalah&hl=en&sa=X&ved=0ahUKEwjWyfLwiITnAhXQXSsKHUBRC_MQ6AEIMzAB" \l "v=onepage&q=Kernel pada OS adalah&f=false).

[7] Aziz, Azminuddin I. S.,dkk. 2019. Fundamental Pemrograman. Budi Utama : Yogyakarta.

**Lampiran :**

**boot.s**

.set MAGIC, 0x1BADB002

.set FLAGS, 0

.set CHECKSUM, -(MAGIC + FLAGS)

.section .multiboot

.long MAGIC

.long FLAGS

.long CHECKSUM

stackBottom:

.skip 1024

stackTop:

.section .text

.global \_start

.type \_start, @function

\_start:

mov $stackTop, %esp

call kernel\_entry

cli

hltLoop:

hlt

jmp hltLoop

.size \_start, . - \_start

**kernel.c**

#include "kernel.h"

#include "utils.h"

uint32 vga\_index;

static uint32 next\_line\_index = 1;

uint8 g\_fore\_color = WHITE, g\_back\_color = BLUE;

int digit\_ascii\_codes[10] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39};

uint16 vga\_entry(unsigned char ch, uint8 fore\_color, uint8 back\_color) {

uint16 ax = 0;

uint8 ah = 0, al = 0;

ah = back\_color;

ah <<= 4;

ah |= fore\_color;

ax = ah;

ax <<= 8;

al = ch;

ax |= al;

return ax;

}

void clear\_vga\_buffer(uint16 \*\*buffer, uint8 fore\_color, uint8 back\_color){

uint32 i;

for(i = 0; i < BUFSIZE; i++){

(\*buffer)[i] = vga\_entry(NULL, fore\_color, back\_color);

}

next\_line\_index = 1;

vga\_index = 0;

}

void init\_vga(uint8 fore\_color, uint8 back\_color){

vga\_buffer = (uint16\*)VGA\_ADDRESS;

clear\_vga\_buffer(&vga\_buffer, fore\_color, back\_color);

g\_fore\_color = fore\_color;

g\_back\_color = back\_color;

}

void print\_new\_line(){

if(next\_line\_index >= 55){

next\_line\_index = 0;

clear\_vga\_buffer(&vga\_buffer, g\_fore\_color, g\_back\_color);

}

vga\_index = 80\*next\_line\_index;

next\_line\_index++;

}

void print\_char(char ch){

vga\_buffer[vga\_index] = vga\_entry(ch, g\_fore\_color, g\_back\_color);

vga\_index++;

}

void print\_string(char \*str){

uint32 index = 0;

while(str[index]){

print\_char(str[index]);

index++;

}

}

void print\_color\_string(char \*str, uint8 fore\_color, uint8 back\_color){

uint32 index = 0;

uint8 fc, bc;

fc = g\_fore\_color;

bc = g\_back\_color;

g\_fore\_color = fore\_color;

g\_back\_color = back\_color;

while(str[index]){

print\_char(str[index]);

index++;

}

g\_fore\_color = fc;

g\_back\_color = bc;

}

uint16 get\_box\_draw\_char(uint8 chn, uint8 fore\_color, uint8 back\_color){

uint16 ax = 0;

uint8 ah = 0;

ah = back\_color;

ah <<= 4;

ah |= fore\_color;

ax = ah;

ax <<= 8;

ax |= chn;

return ax;

}

void gotoxy(uint16 x, uint16 y){

vga\_index = 80\*y;

vga\_index += x;

}

void draw\_generic\_box(uint16 x, uint16 y,

uint16 width, uint16 height,

uint8 fore\_color, uint8 back\_color,

uint8 topleft\_ch,

uint8 topbottom\_ch,

uint8 topright\_ch,

uint8 leftrightside\_ch,

uint8 bottomleft\_ch,

uint8 bottomright\_ch){

uint32 i;

vga\_index = 80\*y;

vga\_index += x;

vga\_buffer[vga\_index] = get\_box\_draw\_char(topleft\_ch, fore\_color, back\_color);

vga\_index++;

for(i = 0; i < width; i++){

vga\_buffer[vga\_index] = get\_box\_draw\_char(topbottom\_ch, fore\_color, back\_color);

vga\_index++;

}

vga\_buffer[vga\_index] = get\_box\_draw\_char(topright\_ch, fore\_color, back\_color);

y++;

vga\_index = 80\*y;

vga\_index += x;

for(i = 0; i < height; i++){

vga\_buffer[vga\_index] = get\_box\_draw\_char(leftrightside\_ch, fore\_color, back\_color);

vga\_index++;

vga\_index += width;

vga\_buffer[vga\_index] = get\_box\_draw\_char(leftrightside\_ch, fore\_color, back\_color);

y++;

vga\_index = 80\*y;

vga\_index += x;

}

vga\_buffer[vga\_index] = get\_box\_draw\_char(bottomleft\_ch, fore\_color, back\_color);

vga\_index++;

for(i = 0; i < width; i++){

vga\_buffer[vga\_index] = get\_box\_draw\_char(topbottom\_ch, fore\_color, back\_color);

vga\_index++;

}

vga\_buffer[vga\_index] = get\_box\_draw\_char(bottomright\_ch, fore\_color, back\_color);

vga\_index = 0;

}

void draw\_box(uint8 boxtype,

uint16 x, uint16 y,

uint16 width, uint16 height,

uint8 fore\_color, uint8 back\_color){

switch(boxtype){

case BOX\_SINGLELINE :

draw\_generic\_box(x, y, width, height,

fore\_color, back\_color,

218, 196, 191, 179, 192, 217);

break;

case BOX\_DOUBLELINE :

draw\_generic\_box(x, y, width, height,

fore\_color, back\_color,

201, 205, 187, 186, 200, 188);

break;

}

}

void fill\_box(uint8 ch, uint16 x, uint16 y, uint16 width, uint16 height, uint8 color){

uint32 i,j;

for(i = 0; i < height; i++){

vga\_index = 80\*y;

vga\_index += x;

for(j = 0; j < width; j++){

vga\_buffer[vga\_index] = get\_box\_draw\_char(ch, 0, color);

vga\_index++;

}

y++;

}

}

void create\_dosbox\_ui(){

gotoxy(32, 2);

print\_color\_string("Welcome to My OS", WHITE, BRIGHT\_BLUE);

gotoxy(35, 4);

print\_color\_string("Created By : ", WHITE, BRIGHT\_BLUE);

gotoxy(25, 5);

print\_color\_string("Rifqy Adli Damhuri (55417203) ", WHITE, BRIGHT\_BLUE);

gotoxy(25, 6);

print\_color\_string("Vika Putri Ariyanti (56417094)", WHITE, BRIGHT\_BLUE);

gotoxy(30, 8);

print\_color\_string("Universitas Gunadarma", WHITE, BRIGHT\_BLUE);

gotoxy(40, 9);

print\_color\_string("2020", WHITE, BRIGHT\_BLUE);

fill\_box(0, 25, 13, BOX\_MAX\_WIDTH - 49, 5, BLUE);

draw\_box(BOX\_DOUBLELINE, 25, 13, BOX\_MAX\_WIDTH - 50, 3, WHITE, BLUE);

gotoxy(40, 14);

print\_color\_string("LOGIN", BRIGHT\_GREEN, BLUE);

gotoxy(28, 15);

print\_color\_string("Username :", BRIGHT\_GREEN, BLUE);

gotoxy(28, 16);

print\_color\_string("Password :", BRIGHT\_GREEN, BLUE);

}

void kernel\_entry(){

init\_vga(WHITE, BRIGHT\_BLUE);

create\_dosbox\_ui();

}

**kernel.h**

#ifndef KERNEL\_H

#define KERNEL\_H

typedef unsigned char uint8;

typedef unsigned short uint16;

typedef unsigned int uint32;

#define VGA\_ADDRESS 0xB8000

#define BUFSIZE 2200

#define BOX\_MAX\_WIDTH 78

#define BOX\_MAX\_HEIGHT 23

#define BOX\_SINGLELINE 1

#define BOX\_DOUBLELINE 2

uint16\* vga\_buffer;

#define NULL 0

enum vga\_color {

BLACK,

BLUE,

GREEN,

CYAN,

RED,

MAGENTA,

BROWN,

GREY,

DARK\_GREY,

BRIGHT\_BLUE,

BRIGHT\_GREEN,

BRIGHT\_CYAN,

BRIGHT\_RED,

BRIGHT\_MAGENTA,

YELLOW,

WHITE,

};

#endif

**types.h**

#ifndef TYPES\_H

#define TYPES\_H

typedef unsigned char uint8;

typedef unsigned short uint16;

typedef unsigned int uint32;

#endif

**utils.c**

#include "utils.h"

uint32 strlen(const char\* str){

uint32 length = 0;

while(str[length])

length++;

return length;

}

uint32 digit\_count(int num){

uint32 count = 0;

if(num == 0)

return 1;

while(num > 0){

count++;

num = num/10;

}

return count;

}

void vsnprintf(int num, char \*number){

int dgcount = digit\_count(num);

int index = dgcount - 1;

char x;

if(num == 0 && dgcount == 1){

number[0] = '0';

number[1] = '\0';

}else{

while(num != 0){

x = num % 10;

number[index] = x + '0';

index--;

num = num / 10;

}

number[dgcount] = '\0';

}

}

utils.h

#ifndef UTILS\_H

#define UTILS\_H

#include "types.h"

extern uint32 strlen(const char\*);

extern uint32 digit\_count(int);

extern void vsnprintf(int, char \*);

#endif

**linker.Id**

ENTRY(\_start)

SECTIONS{

. = 1M;

.text BLOCK(4K) : ALIGN(4K){

\*(.multiboot)

\*(.text)

}

.rodata BLOCK(4K) : ALIGN(4K){

\*(.rodata)

}

.data BLOCK(4K) : ALIGN(4K){

\*(.data)

}

.bss BLOCK(4K) : ALIGN(4K){

\*(COMMON)

\*(.bss)

}

}