LAPORAN TUGAS BESAR I
IF2220-TEORI BAHASA FORMAL DAN AUTOMATA

“Implementasi DFA untuk Pemodelan UNIX Socket”

disusun oleh:

K03-Teknik Informatika 2016

Ricky Kennedy	13516105
Christian Wibisono	13516147

INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2017

1. Deskripsi Persoalan

Aplikasi yang dibuat dapat digunakan memodelkan cara kerja UNIX socket sederhana. Model yang dibuat harus melingkupi semua *state socket* yang mungkin serta method yang mungkin dipanggil dalam komunikasi client-server sederhana yaitu:

1) Socket dapat melakukan aksi *bind* ke alamat tertentu dan kemudian melakukan aksi *listen* untuk mendeteksi koneksi baru yang datang
2) Socket dapat digunakan untuk membuka koneksi ke socket lain, dimana nantinya akan dapat mengirimkan pesan kepada socket tersebut
3) Socket juga dapat digunakan untuk membaca pesan yang telah dikirimkan oleh socket lain yang telah terhubung (dimodelkan state blocking I/O)
4) Socket dapat diatur untuk melakukan blocking / unblocking read
5) Socket dapat menutup koneksi ke socket lain atau terhadap binding

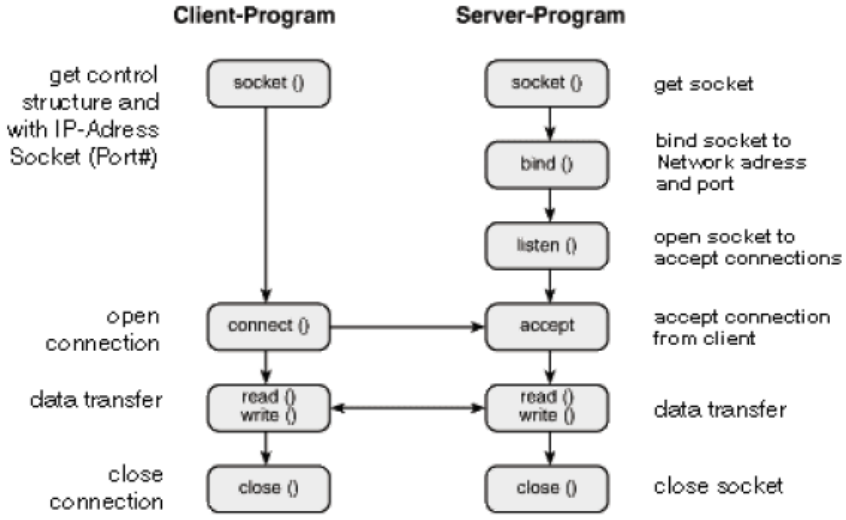
Aplikasi akan membuka file yang berisi informasi mengenai daftar state, daftar simbol, state awal, state akhir dan transition function. Informasi dari file tersebut akan digunakan untuk mengecek masukan dari pengguna. Program diwajibkan untuk membaca konfigurasi dari file eksternal, dan logika state machine tidak di-*hardcode* ke program secara langsung.

2. Landasan Teori

a. Socket

Socket adalah titik pertemuan atau komunikasi antar komputer yang sama atau berbeda untuk melakukan pertukaran data satu sama lain. Dan berbasis pada pemograman API.

Cara kerja Socket adalah seperti pada gambar di bawah ini :



Gambar 1. Cara kerja socket.

Pemrograman API ini bahkan lebih rumit daripada fungsi biasa yang memanggil integrasi port COM, karena keseluruhan kompleksitas mekanisme jaringan dimasukkan ke port soket.

Panah diagonal menunjukkan bagaimana program klien dan server bekerja sama. Yang perlu Anda lakukan adalah menerapkan satu program, tergantung pada apakah Com-Server akan dioperasikan dalam mode klien atau server. Dan beberapa domain dari socket adalah sebagai berikut :

1. Bind : assigning an address and port number to the socket
2. Listen : announce willingness to accept connections
3. Accept : block caller until a connection request arrives
4. Connect : actively attempt to establish a connection
5. Wait : wait the server to reply for the request
6. Close : release a connection
7. Blocking : the default mode of socket calls is blocking. A socket is in blocking mode when an I/O call waits for an event to complete. If the blocking mode is set for a socket, the calling program is suspended until the expected event completes
8. Request : request some data over the connection (usually client to server)
9. Receive : receive some data over the connection (server reply request to client)
10. Read : read data from the connection
11. Write : write data to the connection
12. Abort : return error

b. DFA

Deterministic Finite Automata (DFA) adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa regular) dan dapat diimplementasikan secara nyata dimana sistem dapat berada disalah satu dari beberapa konfigurasi internal yang disebut state.Sistem state merupakan ringkasan informasi yang berkaitan dengan beberapa input sebelumnya yang di perlukan untuk menentukan perilaku sistem pada inputan selanjutnya.

3. Deterministic Finite Automata Pemodelan UNIX Socket

Berdasarkan landasan teori tersebut dapat dimodelkan sebuah Deterministic Finite Automata untuk pemodelan UNIX Socket sebagai berikut:
Dalam memodelkan socket ini penulis menggunakan beberapa asumsi yaitu:

1) Perubahan mode blocking dan non-blocking dapat dilakukan kapan saja setelah socket terinisialisasi
2) Mode default untuk pemodelan ini adalah blocking sesuai UNIX Socket yang sesungguhnya
3) Perbedaan mode blocking dan non-blocking adalah pada mode blocking sebelum data didapatkan dengan aksi return/abort maka kita harus menunggu untuk bisa melakukan perintah lainnya, sedangkan pada mode non-blocking perintah dijalankan secara synchronous.
4) Perintah untuk menutup koneksi yaitu close() bisa dilakukan di *state* manapun

- Q ={closed, init, bound, listening, connecting, connected, waiting, init[non-blocking], bound[non-blocking], listening[non-blocking], connecting[non-blocking]}
- ∑={socket, bind, listen, block, unblock, accept, connect, write, wait, read, return, abort, close}
- q_0= closed
- F={ closed, init, bound, listening, connecting, connected, waiting, init[non-blocking], bound[non-blocking], listening[non-blocking], connecting[non-blocking]}
- Transition Function (δ):

	socket	bind	listen	block	unblock	accept	connect	write	wait	abort	return	read	close
--> closed	init	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected
init	rejected	bound	rejected	rejected	init[blocking]	rejected	connecting	rejected	rejected	closed	rejected	rejected	closed
init[non-blocking]	rejected	bound[non-blocking]	rejected	init	rejected	rejected	connecting[non-blocking]	rejected	rejected	init	rejected	rejected	closed
bound	rejected	rejected	listening	rejected	bound[non-blocking]	rejected	rejected	rejected	rejected	init	rejected	rejected	closed
bound[non-blocking]	rejected	rejected	listening[non-blocking]	bound	rejected	rejected	rejected	rejected	rejected	init[non-blocking]	rejected	rejected	closed
listening	rejected	rejected	rejected	rejected	listening[non-blocking]	connecting	rejected	rejected	rejected	bound	rejected	rejected	closed
connecting	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	connecting	rejected	connected	rejected	closed
connected	rejected	rejected	rejected	rejected	connecting[non-blocking]	rejected	rejected	waiting	rejected	listening	rejected	waiting	closed
waiting	rejected	rejected	rejected	rejected	rejected	rejected	rejected	rejected	waiting	connected	connected	rejected	closed
listening[non-blocking]	rejected	rejected	rejected	listening	rejected	connecting[non-blocking]	rejected	rejected	rejected	bound[non-blocking]	rejected	rejected	closed
connecting[nonblocking]	rejected	rejected	rejected	connecting	rejected	rejected	rejected	connecting [non-blocking]	rejected	connecting[non-blocking]	rejected	conneting[non-blocking]	closed

- Deskripsi State
    1. Closed: socket dalam kondisi tidak sedang melakukan koneksi
    2. Init: kondisi socket memulai untuk membuka koneksi
    3. Bound: kondisi socket yang telah diassign port dan addressnya

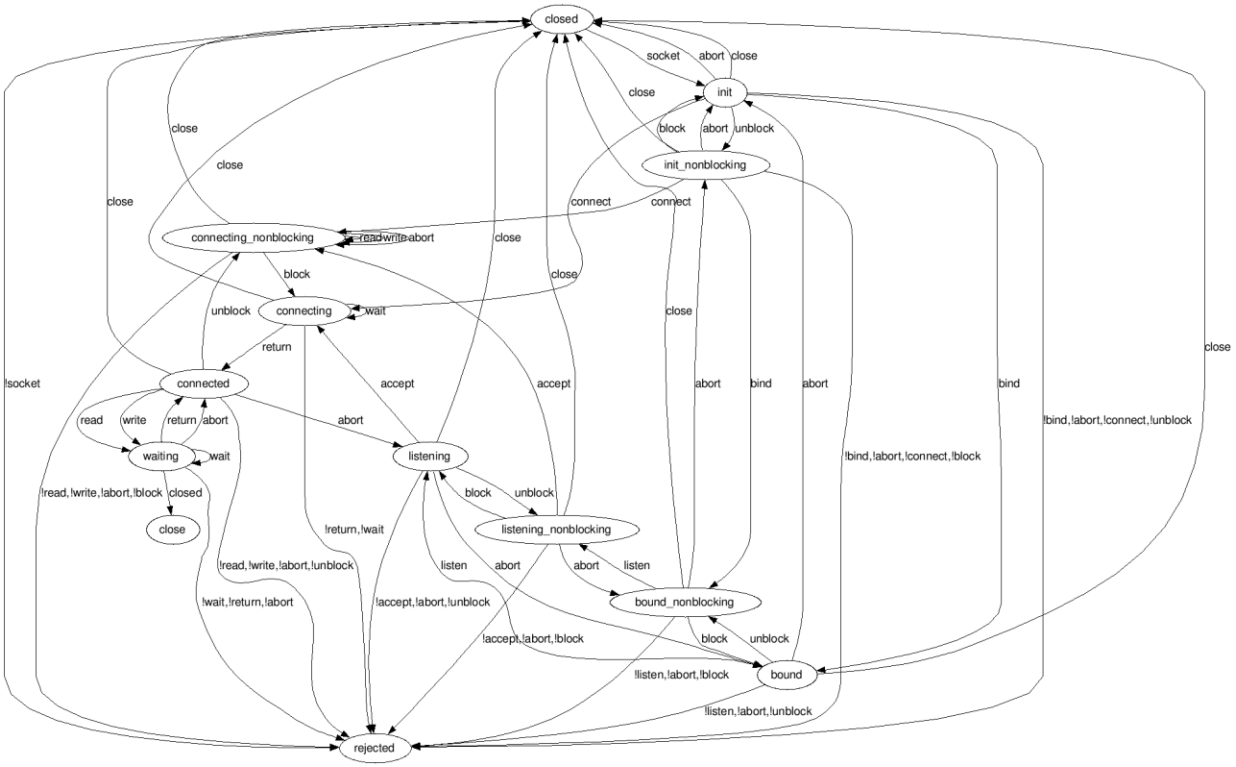
4. Listening: kondisi socket sedang membuka port untuk melakukan koneksi
5. Connecting: socket sedang berusaha melakukan koneksi
6. Connected: socket berhasil terkoneksi, transfer data dilakukan
7. Waiting: socket dalam posisi menunggu kembalian data dari server/client

[non-blocking] menunjukkan state yang sama tetapi untuk mode non-blocking I/O

- Deskripsi aksi
  1. Socket: menginisialisasi socket
  2. Bind: melakukan *assignment* port dan address ke socket
  3. Listen: memerintahkan socket untuk mendengarkan data di port yang ditentukan
  4. Block: melakukan SELECT() untuk berpindah ke mode blocking
  5. Unblock: melakukan SELECT() untuk berpindah ke mode non-blocking
  6. Accept: server menerima request koneksi dari client
  7. Connect: client mengirimkan request untuk terhubung dengan server
  8. Write: melakukan penulisan data
  9. Read: melakukan pembacaan data
  10. Wait: menunggu kembalian data dari server/client
  11. Abort: kembalian data error/gagal menuju state selanjutnya
  12. Return: kembalian data diterima
  13. Close: menutup koneksi pada port
- Delta List
  - δ(closed,socket) = init
  - δ(init,abort) = closed
  - δ(init,bind) = bound
  - δ(init,connect) = connecting
  - δ(init,unblock) = init[non-blocking]
  - δ(init[non-blocking],block) = init
  - δ(init[non-blocking],abort) = init
  - δ(init[non-blocking],bind) = bound[non-blocking]
  - δ(init[non-blocking],connect) = connecting[non-blocking]
  - δ(bound,abort) = init
  - δ(bound,listen) = listening
  - δ(bound,unblock) = bound[non-blocking]
  - δ(bound[non-blocking],block) = bound
  - δ(bound[non-blocking],abort) = init[non-blocking]
  - δ(bound[non-blocking],listen) = listening[non-blocking]
  - δ(listening,abort) = bound
  - δ(listening,accept) = connecting
  - δ(listening,unblock) = listening[non-blocking]

- δ(listening[non-blocking],block) = listening
- δ(listening[non-blocking],abort) = bound[non-blocking]
- δ(listening[non-blocking],accept) = connecting[non-blocking]
- δ(connecting,return) = connected
- δ(connecting,wait) = connecting
- δ(connected,abort) = listening
- δ(connected,read) = waiting
- δ(connected,write) = waiting
- δ(connected,unblock) = connecting[non-blocking]
- δ(connecting[non-blocking],block) = connecting
- δ(connecting[non-blocking],read) = connecting[non-blocking]
- δ(connecting[non-blocking],write) = connecting[non-blocking]
- δ(connecting[non-blocking],abort) = connecting[non-blocking]
- δ(waiting,wait) = waiting
- δ(waiting,return) = connected
- δ(waiting,abort) = connected
- δ(init,close) = closed
- δ(bound,close) = closed
- δ(listening,close) = closed
- δ(connected,close) = closed
- δ(connecting,close) = closed
- δ(waiting,close) = closed
- δ(init[non-blocking],close) = closed
- δ(bound[non-blocking],close) = closed
- δ(listening[non-blocking],close) = closed
- δ(connecting[non-blocking],close) = closed

- Representasi dalam Graph:



Gambar 2 Representasi Graph Pemodelan UNIX Socket

4. Implementasi DFA dalam Bahasa Pemrograman C

Struktur file txt

- Baris pertama berisi daftar State (Q)
- Baris kedua berisi state awal (q_0)
- Baris ketiga berisi daftar Final State (F)
- Baris keempat berisi input yang diterima ($\sum$)
- Baris selanjutnya berisi daftar delta state yang merupakan konfigurasi DFA

```
1  closed init bound listening connecting connected waiting init[non-
   blocking] bound[non-blocking] listening[non-blocking]
   connecting[non-blocking]
2  closed
3  closed init bound listening connecting connected waiting init[non-
   blocking] bound[non-blocking] listening[non-blocking]
   connecting[non-blocking]
4  socket bind listen block unblock accept connect write wait read
   return abort close return
   closed socket init
   init abort closed
   init bind bound
   init connect connecting
   init unblock init[non-blocking]
   init[non-blocking] block init
   init[non-blocking] abort init
   init[non-blocking] bind bound[non-blocking]
   init[non-blocking] connect connecting[non-blocking]
   bound abort init
   bound listen listening
   bound unblock bound[non-blocking]
   bound[non-blocking] block bound
   bound[non-blocking] abort init[non-blocking]
   bound[non-blocking] listen listening[non-blocking]
   listening abort bound
   listening accept connecting
   listening unblock listening[non-blocking]
   listening[non-blocking] block listening
   listening[non-blocking] abort bound[non-blocking]
   listening[non-blocking] accept connecting[non-blocking]
   connecting return connected
   connecting wait connecting
   connected abort listening
   connected read waiting
   connected write waiting
   connected unblock connecting[non-blocking]
   connecting[non-blocking] block connecting
   connecting[non-blocking] read connecting[non-blocking]
   connecting[non-blocking] write connecting[non-blocking]
```

```
connecting[non-blocking] abort connecting[non-blocking]
waiting wait waiting
waiting return connected
waiting abort connected
init close closed
bound close closed
listening close closed
connected close closed
connecting close closed
waiting close closed
init[non-blocking] close closed
bound[non-blocking] close closed
listening[non-blocking] close closed
connecting[non-blocking] close closed
```

1. Definisi Program

```
#define BLANK ' '
#define NEWLINE '\n'
#define NULLREF '\0'
#define NEXT '>'

typedef int IdxType;
typedef char string[1000];
typedef string state;
typedef struct{
      state Qawal;
   state Transition;
      state Qakhir;
}graph;

FILE *in;
string baris,initialState,input, namafile;;
graph TabTrans[100];
string
TabLintasan[100],TabState[100],TabFinalState[100],TabAcceptedInput[
100];
IdxType i,j,k;
```

2. Implementasi program membaca file text.

Pada bagian ini program akan membaca file ekternal yang berisi tentang state-state yang berlaku pada mesin

```
void bacaFileEksternal(string fileName)
//I.S menerima masukan berupa file yang berisi DFA
//F.S Membaca isi file dan menyimpan isi file didalam memory
internal
```

```
{
   in = fopen(fileName,"r");
   fgets(baris,sizeof(baris),in);
   getDaftarState(baris,TabState);
   fgets(baris,sizeof(baris),in);
   strtok(baris,"\n");
   strcpy(initialState,baris);
   fgets(baris,sizeof(baris),in);
   getDaftarState(baris,TabFinalState);
   while(strcmp(TabFinalState[lengthOfFinalState],"\0") != 0 &&
strcmp(TabFinalState[i]," ") != 0){
        lengthOfFinalState++;
    }
    fgets(baris,sizeof(baris),in);
    getDaftarState(baris,TabAcceptedInput);
    while(!feof(in)){
        fgets(baris,sizeof(baris),in);
        salinDelta(baris, row);

    row++;
    }
    fclose(in);
}

void getDaftarState(string f, string TabInput[]
//I.S menerima masukan string f dan array of string TabInput
//F.S mengolah string f dan memparsing f kedalam TabInput
{
    int i=0,j,k=0;
   while(f[i] != NEWLINE ){
   int j=0;
   while((f[i] != BLANK )&&(f[i] != NEWLINE)){
   TabInput[k][j] = f[i];
   i++;
   j++;
   }if(f[i] != NEWLINE ){
   i++;
   }
   k++;
  }
}

void salinDelta(string f, int k)
// I.S menrima string yang mengambarkan transisi
// F.s Menghasilkan tabel transisi
{
    char word[100];
    int i=0,j=0;
    memset(word,'\0',sizeof(word));
```

```
   while(f[i]!=BLANK)
   {
     word[j]=f[i];
     i++;
     j++;
   }
   strcpy(TabTrans[k].Qawal,word);
   i++;
   j=0;
   memset(word,'\0',sizeof(word));
   while(f[i]!=BLANK)
   {
    word[j]=f[i];
     i++;
     j++;
   }
   strcpy(TabTrans[k].Transition,word);
   i++;
   j=0;
   memset(word,'\0',sizeof(word));
   while(f[i]!=NEWLINE && f[i] != NULLREF)
   {
    word[j]=f[i];
    i++;
    j++;
   }
   strcpy(TabTrans[k].Qakhir,word);
}
```

3. Program menerima masukan string dari user.

Pada bagian ini program menerima masukan string yang berupa lintasan state yang diinginkan oleh user.

```
void bacaInputUser()
//I.S membaca masukan berupa array of character
//F.S memasukan ke dalam memory internal
{
    printf("Masukkan input string dibatasi '>':\n");
    strcpy(TabLintasan[0],initialState);
    gets(input);
    i=0;
    while(input[i] != NULLREF){
        j=0;
        while((input[i] != NEXT)&&(input[i] != NULLREF)){
            TabLintasan[lengthInput][j] = input[i];
            j++;
            i++;
        }
```

```
        lengthInput++;
        if(input[i] == NEXT){
            i++;
        }
    }
}
```

4. Implementasi Pencetakan Lintasan pada Program

Pada bagian ini program akan menunjukan lintasan state yang valid untuk ditempuh dan menunjukan masukan diterima atau tidak.

```
void printLintasan(int panjangLintasan)
// menampilkan lintasan yang ditempuh oleh DFA
{
    printf("\nLintasan yang ditempuh DFA:\n");
    printf("start---> ");
    for(int i=0;i<=panjangLintasan;i++){
        printf("%s",TabLintasan[i]);
        if(i != panjangLintasan){
            printf(" -> ");
        }
    }
}
```

5. Validasi DFA

Program untuk menentukan apakah DFA diterima atau tidak sesuai konfigurasi yang dibaca pada file eksternal. Diterima syaratnya pasangan current state dan input terdapat pada file eksternal, apabila tidak maka masuk dalam state rejected. Selain itu juga perlu dilakukan pengecekan apakah current state merupakan final state pada DFA.

```
void cekDFA()
// melakukan pengecekan apkah valid input user terhadap tabel
transition
{
     for(j=0;j<lengthInput-1;j++){
              accepted = false;
              for(i=1;i<=row && !accepted;i++){
                  if(strcmp(TabLintasan[j],TabTrans[i].Qawal) == 0
      &&  strcmp(TabLintasan[j+1],TabTrans[i].Transition) == 0){
                      lengthOfLintasan++;

       strcpy(TabLintasan[lengthOfLintasan],TabTrans[i].Qakhir);
                      accepted = true;
                  }
              }
              if(!accepted){
```

```
                    lengthOfLintasan++;
                    strcpy(TabLintasan[lengthOfLintasan],"rejected");
                    break;
                }
    }
}

boolean isFinalState(int i, int j, int x)
//menghasilkan true jika tablintasan ke j didalam tabel Final
State.
{
      boolean yesFinal=false;
          for(i=0;i<=x && !yesFinal;i++){
              if(strcmp(TabLintasan[j],TabFinalState[i]) == 0){
                  yesFinal=true;
              }
          }
    return yesFinal;
}
```

6. Program Utama

```
int main(){
          printf("Masukkan nama file yang akan dibaca: ");
          gets(namafile);
          bacaFileEksternal(namafile);
          bacaInputUser();
          cekDFA();
          printLintasan(lengthOfLintasan);
          if(accepted && isFinalState(i,j,lengthOfFinalState)){
         printf("\nMasukan DITERIMA\n");
          } else {
          printf("\nMasukan tidak DITERIMA\n");
          }
   return 0;
}
```

7. Interaksi I/O Program

```
Masukkan nama file yang akan dibaca: dfa.txt
Masukkan input string dibatasi '>':
socket>bind>listen>accept>return>read>wait>return>close

Lintasan yang ditempuh DFA:
start---> closed -> init -> bound -> listening -> connecting ->
connected -> waiting -> waiting -> connected -> closed
Masukan DITERIMA
```

```
Masukkan nama file yang akan dibaca: dfa.txt
Masukkan input string dibatasi '>':
socket>bind>listen>abort>listen>write

Lintasan yang ditempuh DFA:
start---> closed -> init -> bound -> listening -> bound ->
listening -> rejected
Masukan tidak DITERIMA
```

```
Masukkan nama file yang akan dibaca: dfa.txt
Masukkan input string dibatasi '>':
socket>connect>return>write>return>unblock>read>write>read>write>cl
ose

Lintasan yang ditempuh DFA:
start---> closed -> init -> connecting -> connected -> waiting ->
connected -> connecting[non-blocking] -> connecting[non-blocking] -
> connecting[non-blocking] -> connecting[non-blocking] ->
connecting[non-blocking] -> closed
Masukan DITERIMA
```

```
Masukkan nama file yang akan dibaca: dfa.txt
Masukkan input string dibatasi '>':
socket>listen

Lintasan yang ditempuh DFA:
start---> closed -> init -> rejected
Masukan tidak DITERIMA
```