Documentatie CastDoc

Fiodorov Cristian

December 2022

1 Introducere

Proiectul CastDoc este o aplicatie de tip client/server, comunicarea facanduse prin intermediul socket-urilor. Aplicatia are ca scop transformarea de documente dintr-un tip in altul. Clientul va fi responsabil de specificarea tipului documentului initial si final. Daca serverul este capabil sa realizeze transformarea, clientul va transmite documentul ce se doreste convertit si va primi de la server documentul in formatul lui nou.

2 Tehnologiile utilizate

2.1 TCP/IP

TCP/IP (Transmission Control Protocol/Internet Protocol) este cel mai utilizat protocol datorita disponibilitatii si flexibilitatii acestuia.
Modelul **TCP/IP** este un set de protocoale care asigură transferul pachetelor de date cu o rată de eroare foarte mică printr-o reţea neomogenă de calculatoare. TCP/IP este o suită de protocoale, dintre care cele mai importante sunt TCP şi IP. Modelul TCP/IP este structurat pe patru nivele (Nivelul Aplicatie, Nivelul Transport, Nivelul Internet si Nivelul Retea), fiecare avand scopul sau in aceasta constructie.

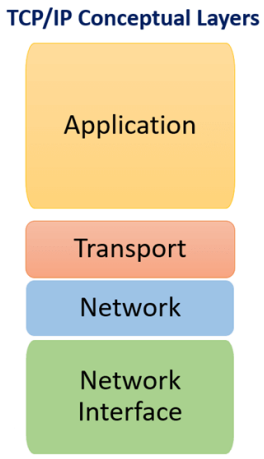

Figure 1: TCP/IP layers

TCP/IP foloseste modelul de comunicare **client/server** care este un model centralizat, spre deosebire de **P2P** (Peer-to-peer) care este un model distribuit. Prin urmare, in modelul client/server, unei masini (clientului) ii este furnizat un serviciu de catre alt calculator (server) prin intermediul internetului.
Am ales protocolul **TCP/IP**, deoarece are multe avantaje care o sa ne ajute in implimentarea proiectului propus (CastDoc). Spre exemplu, foloseste modelul client/server (ceea ce este crucial, deoarece proiectul nostru este bazat pe acest tip de model de comunicare), permite comunicarea intre masini care folosesc diferite sisteme de operare, si este un model standart folosit pe scara larga.

2.2 TCP

In cadrul acestui proiect, am utilizat protocolul de transport **TCP** (Transmission Control Protocol). Am utilizat **TCP** in defavoarea **UDP** (User Datagram Protocol), deoarece TCP este orientat conexiune, full-duplex si nu poate avea pierderi de informatii, ceea ce este foarte important pentru proiectul CastDocs din moment ce o sa trebuiasca sa transmite documente de la server catre client si vice versa. Daca am utiliza UDP, ar fi sanse sa pierdem parti din documente, deoarece UDP este nesigur si nu recurge la negocieri sau la confirmari pentru primirea datelor. De asemenea UDP nu garanteaza ordinea receptionarii datelor, deci ar fi sanse sa reconstruim documentul intr-o ordine gresita, chiar daca primim toate partile din el.

2.3 SHA-256

De asemenea, am utilizat tehnologia de hashing **SHA-256** (Secure Hash Algorithm 256-bit) pentru generarea hash-ului fisierelor care va fi verificat de catre server inaite sa realizeze transferul fisierului de la client la server, astfel nu vom avea duplicate de fisiere in server. Am utilizat versunea SHA-256, care utilizeaza 256 de biti pentru generarea hash-ului, in detrimentul celorlalte versiuni cu mai putini biti, deoarece cu cât este mai mare numărul de rezultate posibile ale indexării, cu atât sunt mai mici șansele ca două valori să creeze același rezultat. Nu am utilizat SHA-512, care genereaza un hash de 512 biti, deoarece am considerat 256 de biti indeajuns pentru scopul care dorim sa-l optinem.

2.4 Child process (proces copil)

Un proces este un program în execuție. Procesele sunt unitatea primitivă prin care sistemul de operare alocă resurse utilizatorilor. Orice proces are un spațiu de adrese și unul sau mai multe fire de execuție. Putem avea mai multe procese ce execută același program, dar oricare două procese sunt complet independente.
Un **proces copil** este un proces creat de un alt proces (procesul parinte), si este complet independent fata de procesul parinte sau de alte procese copii. Sunt doua proceduri pentru crearea unui proces copil: apelul de sistem **fork** (preferat in familia de sisteme de operare Unix) si **spawn** (preferat in versiunile moderne de Windows).

Am ales sa implimentez serverul concurent cu ajutorul proceselor copii, deoarece nu am nevoie de memorie partajata in server, ceea ce ne ofera threadurile. De asemenea, am avut nevoie de directoare separate de lucru pentru fiecare proces copil, ceea ce nu ne putea oferi firele de executie (deoarece ele impart acelasi director de lucru).

2.5 QT

Qt este un sistem inter-platformă de dezvoltare a programelor pentru calculator, care cuprinde o bibliotecă cu elemente de control, folosit atât pentru crearea programelor cu interfață grafică cât și pentru programe fără interfață grafică, cum sunt programele care rulează pe servere (backend).
Pentru acest proiect, am ales sa realizez interfata cu ajutorul framework-ului Qt, deoarece **Qt** este utilizat pe scara larga pentru dezvoltarea sowtware-ului care poate fi rulat pe diverse platforme sotware si hardware, fara modificari sau cu modificari reduse, avand in acelasi timp puterea si viteza aplicatiilor native.

3 Arhitectura aplicatiei

Aritectura aplicatiei reprezinta organizarea si structurarea unui sistem (aplicatii).
In imaginea de mai jos avem reprezentata arhitectura aplicatiei **CastDoc** sub forma unei diagrame. Din diagrama se observa cum comunica fiecare parte din aplicatie si cum interactioneaza utilizatorul cu aplicatia. Utilizatorul are acces la programul client prin intermediul unei interfate. Programul Client este responsabil doar pentru primirea si trimiterea de date catre server si client. In Programul Server are loc toata procesarea necesara pentru transformarea unui document dintr-un tip in altul si transmiterea noului document catre Client. Comunicarea intre server si client se efectueaza prin intermediul socket-urilor care utilizeaza TCP pe post de protocol de transport.

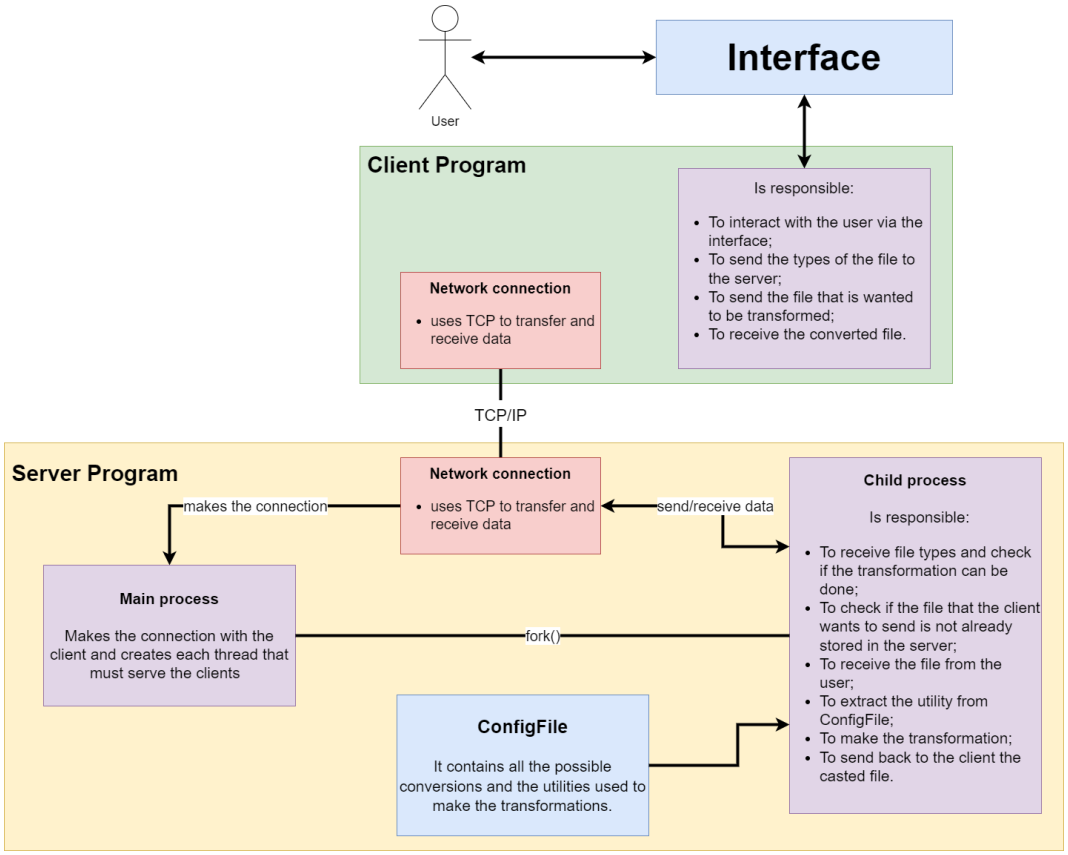

Figure 2: Diagrama aplicatiei

4 Detalii de implementare

4.1 Trimiterea/Receptionareaa fisierului

Trimiterea si receptionarea fisierului va fi realizata prin transmiterea/receptionarea de pachete cu dimensiunea de **4Kb**.

Pasii pentru implimentarea transmiterii si primirii de fisiere sunt:

1. **Expediator:** Transmite numele fisierului care urmeaza sa fie trimis.
2. **Destinatar:** Verifica daca exista un fisier cu acel nume. Daca nu exista - va crea un fisier care va avea numele primit, dar daca exista va adauga in fata numelui 'new_' si apoi va crea fisierul.
3. **Expediator:** Deschide fisierul.
4. **Expediator:** Copie intr-un buffer cate 4096 de bytes de informatie care mai tarziu vor fi trimisi prin socket catre destinatar.
5. **Destinatar:** Adauga in fisierul nou format informatia care o primeste prin socket de la expediator.
6. Se tot repeta pasii 4 si 5 pana cand este citit tot fisirul de catre expediator.

Mai jos se gaseste codul pentru functia de primire si pentru cea de transmitere de fisiere.

```
int send_file(int sd, const char* file_name, const char*
    receive_name)
{
    unsigned char buff[BuffSize];
    int fd;
    int bytes = 0;

    if((fd = open(file_name, O_RDONLY)) == -1){
        fprintf(stderr, "Cannot open %s!!", file_name);
        return -1;
    }

    bytes = strlen(receive_name) + 1;
    if(write(sd, &bytes, 4) == -1){
        fprintf(stderr, "Error at writing in socket!!");
        return -1;
    }

    if(write(sd, receive_name, bytes) == -1){
        fprintf(stderr, "Error at writing in socket!!");
        return -1;
    }

    memset(buff, 0, BuffSize);
    while(1) {
        if ((bytes = read(fd, buff, BuffSize)) == -1)
            return -1;
```

```
27
28          if(write(sd, &bytes, sizeof(bytes)) == -1)
29              return -1;
30
31          if(bytes == 0)
32              break;
33
34          if(write(sd, buff, bytes) == -1)
35              return -1;
36      }
37
38      close(fd);
39
40      return 0;
41  }
```

```
1   int receive_file(int sd, char* file_name)
2   {
3       int fd;
4       int bytes = 0;
5
6       if((read(sd, &bytes, 4)) == -1){
7           fprintf(stderr, "Error at reading from socket!!");
8           return -1;
9       }
10
11      memset(file_name, 0, 255);
12      if((read(sd, file_name, bytes)) == -1){
13          fprintf(stderr, "Error at reading from socket!!");
14          return -1;
15      }
16
17
18      file_name[bytes] = '\0';
19
20      while(access(file_name, F_OK) != -1){
21          char tmp[255] = "new_";
22          strcat(tmp, file_name);
23          strcpy(file_name, tmp);
24      }
25
26      printf("%s\n", file_name);
27
28      if(creat(file_name, 0666) == -1){
29          fprintf(stderr, "Error at the creation of %s!!", file_name
        );
30          return -1;
31      }
32
33      if((fd = open(file_name, O_WRONLY)) == -1){
34          fprintf(stderr, "Cannot open %s!!\n", file_name);
35          char mm[1000];
36          getcwd(mm, sizeof(mm));
37          fprintf(stderr, "current dir: %s\n", mm);
38          return -1;
39      }
40
41      unsigned char buff[BuffSize];
```

```
42      memset(buff, 0, BuffSize);
43
44      while(1) {
45          if(read(sd, &bytes, sizeof(bytes)) == -1)
46              return -1;
47
48          if(bytes == 0)
49              return 0;
50
51          int left_to_read = bytes;
52          int read_bytes = 0;
53          while(1) {
54              if((read_bytes = read(sd, buff + read_bytes,
    left_to_read)) == -1)
55                  return -1;
56
57              if((left_to_read -= read_bytes) == 0)
58                  break;
59          }
60
61          if(write(fd, buff, bytes) == -1)
62              return -1;
63      }
64
65      close(fd);
66
67      return 0;
68  }
```

4.2 Extragerea utilitarului

In momentul cand serverul receptioneaza tipul fisierului si tipul in care se doreste sa fie transformat, se verifica daca este posibila aceasta conversie. Daca este posibila se extrage utilitarul din **ConfigFile**.
Fisierul ConfigFile arata in felul urmator:

```
pdf-ps          pdf2ps
ps-pdf          ps2pdf
txt-man         text2man
```

In partea stanga avem tipurile de data si in partea dreapta avem utilitarul folosit pentru transformare.

```
int get_utility(const char* type1, const char* type2, char*
    utility)
{
  char types[100];
  sprintf(types, "%s-%s", type1, type2);

  int fd;
  if((fd = open("ConfigFile", O_RDONLY)) == -1){
    fprintf(stderr, "Nu sa putut deschide fisierul ConfigFile!!");
        return -1;
  }

  int bytes = 0;
  char line[1000] = "";

  while(1){
    memset(line, 0, 1000);

    bytes = read_line(fd, line);

    if(bytes <= 0){
      close(fd);
      return -1;
    }

    line[bytes] = '\0';

    if(strstr(line, types) != NULL){
      break;
    }
  }

  close(fd);


  char* u = strtok(line, " ");
  u = strtok(NULL, " ");

  strcpy(utility, u);

  return 0;
}
```

4.3 Transformarea fisierului

Dupa ce am obtinut utilitarul, construim comanda pentru transformarea fisierului si o executam cu ajutorul functiei system. Functia de mai jos reprezinta o versiune simplificata a functiei cast folosite.

```
int cast(const char* utility, const char* file_input, const char*
    file_output){
  char com[1000] = "";
  sprintf(com, "%s '%s' '%s'", utility, file_input, file_output);
  system(com);

  char hash[500];
  get_hash(file_output, hash);

  rename_file(file_output, hash);
  strcpy(hash_output, hash);

  return 0;
}
```

4.4 Functionalitatea de caching

Functionalitatea de caching va fi implimentata prin verificarea numelui si hash-ului fisierului. Pentru extragerea hash-ului vom folosi functia de mai jos.

```
int get_hash(const char* file_name, char* buff){
    std::random_device random;
    std::mt19937 mt(random());
    std::uniform_int_distribution<> number{ 0, 25 };

    char file_verify_cache[50];
    for(int i = 0; i < 49; ++i)
        file_verify_cache[i] = 'a' + number(mt);
    file_verify_cache[49] = '\0';

    while(access(file_name, F_OK) == -1)
        sleep(0);

    char command[1000] = "";
    sprintf(command, "sha256sum %s > %s", file_name,
    file_verify_cache);
    system(command);

    int fd1;
    if((fd1 = open(file_verify_cache, O_RDONLY)) == -1){
        fprintf(stderr, "Cannot open verify_cache!!");
        return -1;
    }
    memset(buff, 0, 1000);
    int bytes = read_line(fd1, buff);
    buff[bytes] = '\0';
    close(fd1);

    sprintf(command, "rm %s", file_verify_cache);
    system(command);
    return 0;
}
```

4.5 Transformari multiple pentru obtinerea fisierului dorit

Functionalitatea de a transforma dintr-un tip de fisier in altul, fara a avea un utilitar pentru a realiza transformarea directa, a fost implimentata prin realizarea unui graf $G = (V, A)$, unde V este multimea tuturor tipurilor suportate si A este o multime de arce unde $\forall t_1, t_2$, $(t_1, t_2) \in A \iff$ exista un utilitar care sa realizeze transformarea de la t_1 la t_2.
Am obtinut tipurile intermediare prin aplicarea unui algoritm, bazat pe BFS, de gasirea unui drum de la un nod la altul.

```
int get_cast_path(const char* type1, const char* type2, std::
    vector<std::string>& cast_path)
{
  std::vector<std::string> visited;
  std::queue<std::string> queue;
  queue.push(type1);

  std::unordered_map<std::string, std::string> parent;
  parent[type1] = "root";

  visited.push_back(type1);

  int found = -1;

  while (!queue.empty()) {
    std::string current_type = queue.front();
    queue.pop();

    if (current_type == type2){
      found = 0;
      break;
    }
    else {
      auto to_types = types_map[current_type];
      for(int i = 0; i < to_types.size(); ++i) {
        if (std::find_if(visited.begin(), visited.end(), [&](const
    std::string& str) { return str == to_types[i]; } ) == visited.
   end()){
          queue.push(to_types[i]);
          parent[to_types[i]] = current_type;
          visited.push_back(to_types[i]);
        }
      }
    }
  }

  if(found == -1)
    return -1;

  std::string current_type = type2;

  while (current_type != "root") {
    cast_path.push_back(current_type);
    current_type = parent[current_type];
  }

  return 0;
}
```

Dupa obtinerea drumului cu ajutorul functiei de mai sus, am obtinut si utilitarele, care trebuiesc folosit, prin iterarea tabloului in care este stocat drumul si prin apelarea functiei get_utility. Prin transformari repetate, obtinem fisierul de tipul dorit.
Exemplu: Daca dorim sa transformam un fisier de tipul **txt** intr-unul de tipul **gif**, functia de mai sus (get_cast_path) va returna urmatorul drum: txt → docx → pdf → jpg → gif.

5 Concluzii

Implimentarea proiectului dat poate fi imbunatatita si pot fi adaugate diverse caracteristici noi. Ceea ce tine de tehnologiile folosite, am putea folosi **SHA-512** in loc de **SHA-256** pentru crearea hash-ului folosit pentru operatia de caching a fisierelor in server. In momentul cand serverul va avea in cache multe fisiere, va exista posibilitatea, chiar si daca ar fi foarte mica, ca hash-urile a doua fisiere sa coincida. Ca sa scadem aceasta posibilitate, sa folosim **SHA-512** ar fi cea mai buna solutie.
De asemenea, am putea folosi alt framework in loc de **Qt** pentru crearea interfetei clientului, cum ar fi **GTK+**. **GTK+**, sau **The GIMP Toolkit**, este o bibliotecă care cuprinde elemente de control și un sistem de dezvoltare a interfețelor grafice. **GTK+** ne ofera posibilitatea sa cream interfate care arata mai modern de cat interfetele facute in **Qt**.

Bibliografie

[1] Proiecte Retele de calculatoare UAIC 2022-2023
`https://profs.info.uaic.ro/~computernetworks/ProiecteNet2022.php`

[2] Pagina Curs Retele de calculatoare UAIC 2022-2023
`https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php#s2`

[3] TCP/IP
`https://www.ibm.com/docs/en/zos/2.2.0?topic=internets-tcpip-tcp-udp-ip-protocols`

[4] TCP
`https://profs.info.uaic.ro/~computernetworks/files/4rc_NivelulTransport_Ro.pdf`

[5] SHA-256
`https://support.google.com/google-ads/answer/9004655?hl=ro`

[6] Proces copil (Child process) - Wikipedia
`https://en.wikipedia.org/wiki/Child_process`

[7] Qt - Wikipedia
`https://ro.wikipedia.org/wiki/Qt`