# SCD 2024-2025 - Tema 1 - Aplicație Client/Server pentru autorizare OAuth folosind RPC

Responsabili: George-Mircea Grosu și Silvia-Elena Nistor

Titular curs: Florin Pop

## 1 Obiectivele temei

În cadrul temei veți dezvolta o aplicație de tip client/server folosind conceptul de Remote Procedure Call (RPC) pentru simularea unui sistem de autorizare de tip OAuth (Vezi fig. [1]).

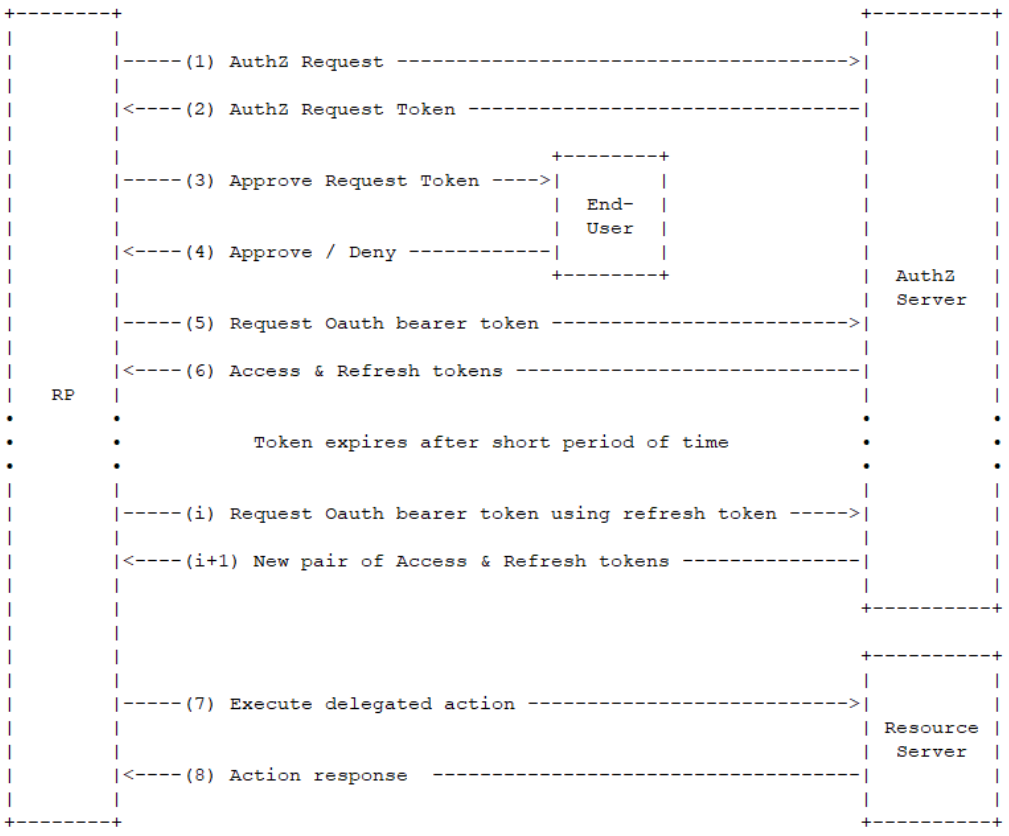


Figura 1: Prezentare generală a arhitecturii propuse.

Obiectivele principale ale acestei teme de laborator sunt:

- realizarea unei aplicații de tip client/server bazate pe RPC cu cereri/răspunsuri multiple prin care se pot autoriza operații de tip RIMDX (Read, Insert, Modify, Delete, Execute) pe un sistem de fișiere;
- utilizarea corectă a modelului de autorizare prin Oauth;
- scrierea de specificații pentru o aplicație simplă de autorizare și administrare a resurselor;
- utilizarea de structuri de date pentru aplicații ce folosesc protocolul RPC.

Cunoştinţele necesare rezolvării acestei teme de laborator sunt următoarele:

- programare în C/C++, structuri, funcții, lucrul cu fișiere binare/text;
- noțiuni privind folosirea utilitarului RPCGEN și a compilatorului gcc.

# 2 Descrierea soluției cerute

Pentru această temă ne propunem implementarea unei soluții de tip client/server pentru autorizarea accesului la resursele localizate pe un server de fișiere. Aplicația client/server va fi implementată peste protocolul RPC.

În soluția propusă, clientul reprezintă o aplicație terță, folosită de utilizatorul final, pentru gestionarea fișierelor de pe server. Astfel, clientul este responsabil pentru autorizarea operațiilor delegate de utilizator.

Server-ul are trei funcții: autorizarea clienților, gestionarea resurselor și componenta de aprobare a jetoanelor de autorizare din partea utilizatorului final. În acest sens, server-ul oferă clienților jetoane de acces (tokens) care autorizează operațiile efectuate asupra fișierelor de pe server.

**Exemplu**

Un student intenționează să folosească pentru redactarea lucrării de licență, un utilitar online pentru editarea documentelor în LaTeX. Studentul are încărcate toate resursele necesare în mediul de stocare Cloud pus la dispoziție de facultate. Pentru a nu își expune credențialele către utilitarul online, studentul alege să ofere permisiuni de gestionare a fișierelor din Cloud-ul facultății prin intermediul standardului Oauth. Astfel, utilitarul primește un jeton de acces pe care îl atașează operațiilor de editare și salvare a documentelor LaTeX direct în mediul de stocare Cloud al facultății.

## 2.1 Server

### 2.1.1 Server autorizare

Serverul gestionează în memorie o bază de date în care sunt stocate:

- identificatorii utilizatorilor – încărcați la pornirea server-ului dintr-un fișier primit ca parametru;

```
1 5                  // numarul de utilizatori cunoscuti de server
2 oD0prOgBqAsXBW8    // identificatorul unui utilizator
3 OVotQBYz418Ozkz
4 DQygMQ9F65QzYs7
5 bm5N4D4X2n66nK3
6 Xg6YuaDVv7i1kEk
```

Listing 1: Fișier intrare cu identificatorii utilizatorilor.

- seturile de permisiuni ale utilizatorilor;
- informațiile despre jetoanele de acces autorizate pentru aplicațiile terțe.

Baza de date este populată și administrată pe parcursul execuției server-ului și a interacțiunii cu utilizatorii finali.

Această componentă a server-ului expune un set minimal de proceduri pentru implementarea modelului de autorizare Oauth:

- **Request Authorization**
  1. Primește de la client ID-ul asociat utilizatorului.
  2. Verifică existența utilizatorului în baza de date.
     - Dacă există, întoarce un jeton pentru autorizarea cererii de acces la resurse.
     - Altfel, întoarce eroarea USER_NOT_FOUND.
- **Request Access Token**
  1. Primește de la client: ID-ul utilizatorului și jetonul pentru autorizarea cererii de acces (aprobat de utilizator).
  2. Verifică dacă jetonul este valid, și:
     (a) asociază în baza de date utilizatorul cu noul jeton de acces la resurse (întotdeauna va exista un singur jeton de acces activ pentru un utilizator);
     (b) întoarce un jeton de acces la resurse, un jeton pentru regenerarea celui de acces și o perioadă de valabilitate a acestora.
  3. Jetonul este considerat invalid dacă nu a fost aprobat de utilizator → întoarce eroarea REQUEST_DENIED.

#### 2.1.2 Server resurse

Serverul validează operațiile primite de la o aplicație terță, delegate de utilizator, în vederea gestionării resurselor de pe server. Resursele disponibile pe server vor fi încărcate dintr-un fișier text.

```
1 6                 // numarul tipurilor de resurse cunoscute de server
2 Files             // numele resursei
3 UserData
4 SystemData
5 SystemSettings
6 Applications
7 Preferences
```

Listing 2: Fișier intrare cu resursele disponibile pe server.

Pentru această componentă este necesară implementarea unei proceduri **Validate Delegated Action**.

1. Primește de la client: tipul operației, resursa accesată și jetonul de acces.
2. Verifică dacă jetonul de acces este valid, caz în care:
   (a) verifică dacă jetonul de acces este asociat id-ului de utilizator, iar dacă nu este, întoarce eroarea PERMISSION_DENIED.
   (b) verifică valabilitatea jetonului și dacă a expirat, atunci întoarce eroarea TOKEN_EXPIRED.
   (c) verifică dacă resursa există, iar în caz contrar întoarce eroarea RESOURCE_NOT_FOUND;
   (d) verifică dacă operația este permisă prin jetonul de acces, iar în caz contrar întoarce eroarea OPERATION_NOT_PERMITTED;
   (e) întoarce valoarea PERMISSION_GRANTED;

### 2.1.3 Utilizatorul final

Pentru respectarea modelului client-server, se consideră că interacțiunea cu utilizatorul este realizată printr-o procedură expusă tot de server, **Approve Request Token**.

1. Primește jetonul pentru autorizarea cererii de acces și îi atașează setul de permisiuni pentru fiecare resursă disponibilă pe server.
2. Apoi marchează jetonul ca fiind semnat.
3. Dacă utilizatorul nu aprobă cererea de accesare a datelor, atunci întoarce jetonul neschimbat.

Răspunsurile utilizatorilor vor fi încărcate dintr-un fișier CSV la pornirea server-ului. Acestea vor fi servite clienților în ordine FIFO.

```
1 <resursa>,<permisiune resursa>,<resursa>,<permisiune resursa>,...
2
3 Files,RM,Applications,R,User Data,R
4 Files,RIMD,Applications,RX,User Data,RI
5 Files,RIMD,Applications,RIMDX,System Data,R,Preferences,RM
6 *,-    // utilizatorul nu aproba permisiunile
7        // pe nicio resursa
8 Files,RM,Applications,R,UserData,RIM
```

Listing 3: Fișier intrare pentru aprobările utilizatorilor.

## 2.2 Client

Clientul citește la pornire un fișier CSV și execută în ordine FIFO toate operațiile descrise. În timpul execuției, acesta trebuie să se asigure că respectă și utilizează modelul Oauth pentru autorizarea operațiunilor.

În fișier vor exista două tipuri de operații: pentru generarea jetonului de acces și pentru operarea fișierelor de pe server.

```
1  <ID utilizator>,<actiune>,<resursa>
2  <ID utilizator>,REQUEST,<reimprospatare automata>
3
4  oD0prOgBqAsXBW8,REQUEST,0
5  oD0prOgBqAsXBW8,MODIFY,Files
6  OVotQBYz418Ozkz,REQUEST,1
7  OVotQBYz418Ozkz,EXECUTE,Applications
8  OVotQBYz418Ozkz,DELETE,Files
9  oD0prOgBqAsXBW8,INSERT,UserData
10 OVotQBYz418Ozkz,READ,SystemSettings
```

Listing 4: Fișier intrare pentru client.

### 2.2.1 Model de execuție a programului Client

Un exemplu complet de rulare, pentru o valabilitate de 2 operații a jetoanelor, este arătat în Listing-urile 5-10.

```
1 2
2 Client 1
3 Client 2
```

Listing 5: Exemplu fișier de intrare clienți.

```
1 6
2 Files
3 UserData
4 SystemData
5 SystemSettings
6 Applications
7 Preferences
```

Listing 6: Exemplu fișier de intrare resurse.

```
1 Files,RM,Applications,R,UserData,R                          // Client 1
2 Files,RIMD,Applications,RX,UserData,RI                      // Client 2
3                                                             // 1st request
4 Files,RIMD,UserData,RIMD,SystemData,R,Preferences,R         // Client 2
5                                                             // 2nd request
6 *,-                                                         // Client 1
7                                                             // 2nd request
8 Files,RM,Applications,R,UserData,RIM                        // Client 1
9                                                             // 3rd request
```

Listing 7: Exemplu fișier de intrare aprobări.

```
1                                   // Raspuns primit
2 Client 1,REQUEST,1                // <req. code> -> <acc. token>,
3                                                    <ref. token>
4 Client 1,MODIFY,Files             // PERMISSION_GRANTED
5 Client 2,EXECUTE,Applications     // PERMISSION_DENIED
6 Client 2,REQUEST,0                // <req. code> -> <acc. token>
7 Client 2,EXECUTE,Applications     // PERMISSION_GRANTED
8 Client 2,DELETE,Files             // PERMISSION_GRANTED
9 Client 1,READ,Applications        // PERMISSION_GRANTED
10 Client 2,READ,SystemSettings     // TOKEN_EXPIRED
11 Client 2,REQUEST,0               // <req. code> -> <acc. token>
12 Client 1,INSERT,UserData         // OPERATION_NOT_PERMITTED
13 Client 1,READ,Files              // PERMISSION_GRANTED
14 Client 1,REQUEST,0               // REQUEST_DENIED
15 Client 1,REQUEST,0               // <req. code> -> <acc. token>
16 Client 1,INSERT,UserData         // PERMISSION_GRANTED
17 Client 2,EXECUTE,Malware         // RESOURCE_NOT_FOUND
18 Client 3,REQUEST,0               // USER_NOT_FOUND
```

Listing 8: Exemplu fișier de intrare operații.

```
1 1 leintC -> e1Cntl i,nCl t1ie
2 PERMISSION_GRANTED
3 PERMISSION_DENIED
4 <req. code> -> <acc. token>
5 PERMISSION_GRANTED
6 PERMISSION_GRANTED
7 PERMISSION_GRANTED
8 TOKEN_EXPIRED
9 <req. code> -> <acc. token>
10 OPERATION_NOT_PERMITTED
11 PERMISSION_GRANTED
12 REQUEST_DENIED
13 <req. code> -> <acc. token>
14 PERMISSION_GRANTED
15 RESOURCE_NOT_FOUND
16 USER_NOT_FOUND
```

Listing 9: Model date ieșire client.

```
1 BEGIN Client 1 AUTHZ
2   RequestToken = 1 leintC
3   AccessToken = e1Cntl i
4   RefreshToken = nCl t1ie
5 PERMIT (MODIFY, Files, e1Cntl i, 1)  // (actiune, resursa, jeton acces,
6                                      // operatii ramase)
7 DENY (EXECUTE, Applications, , 0)
8 BEGIN Client 2 AUTHZ
9   RequestToken = l n2ieCt
10  AccessToken = ielC 2tn
11 PERMIT (EXECUTE, Applications, ielC 2tn, 1)
12 PERMIT (DELETE, Files, ielC 2tn, 0)
13 PERMIT (READ, Applications, e1Cntl i, 0)
14 DENY (READ, SystemSettings, , 0)
15 BEGIN Client 2 AUTHZ
16  RequestToken = tieC2n l
17  AccessToken = Ct2i eln
18 BEGIN Client 1 AUTHZ REFRESH
19  AccessToken =  ne1tlCi     // foloseste jetonul pentru
20                             // reinnoire
21  RefreshToken =  enltiC1
22 DENY (INSERT, UserData, ne1tlCi, 1)
23 PERMIT (READ, Files, ne1tlCi, 0)
24 BEGIN Client 1 AUTHZ
25  RequestToken = nt leiC1
26 BEGIN Client 1 AUTHZ
27  RequestToken = iCn 1tle
28  AccessToken = lCt1ien
29  RefreshToken = nl1eit C
30 PERMIT (INSERT, UserData, lCt1ien , 1)
31 DENY (EXECUTE, Malware, Ct2i eln, 1)
32 BEGIN Client 3 AUTHZ
```

Listing 10: Model date ieșire server.

```
./server <fisier clienti> <fisier resurse> <fisier aprobari> \
<valabilitate jetoane>

./client <adresa server> <fisier operatii>
```

Listing 11: Model comenzi rulare client și server.

# 3 Constrângeri pentru implementare

- ID-ul utilizatorilor este un *șir de caractere, alfa-numeric, cu lungime fixă de 15 caractere.*
- Numele resurselor va fi format din caractere alfa-numerice și nu va conține spații.
- În structura jetonului de acces nu trebuie să existe ID-ul utilizatorului.
- Semnarea jetonului pentru autorizarea cererii de acces este simbolică și nu există restricții privind modul de implementare.
- Valabilitatea unui jeton de acces este măsurată în operații autorizate cu acesta. Valoarea este primită ca parametru la rularea server-ului.
- La generarea jetonului de acces folosind jetonul de reînnoire nu se mai cere aprobarea utilizatorului. Reînnoirea jetonului de acces tebuie să fie inițiată de client.
- Pentru generarea jetoanelor se folosește funcția *generate_access_token* din fișierul *token.h* și se respectă următoarea dependență:
  - f(ID utilizator) = jeton pentru autorizarea cererii de acces
  - f(jeton pentru autorizarea cererii de acces) = jeton de acces
  - f(jeton de acces) = jeton de refresh
- Dacă clientul a optat pentru reînnoirea automată a acestuia, jetonul de acces va fi reînnoit înainte de a executa operația (operație care ar fi eșuat din cauza expirării jetonului).
- Dacă clientul nu a optat pentru reînnoire automată, atunci jetonul de reînnoire nu va fi generat.
- Arhiva temei trebuie să conțină: sursele, Makefile-ul, testele și scriptul check.sh folosit.
- Tema se va implementa în C/C++, respectând coding style-ul prezentat la Programare în anul I.

# 4 Cerințele pentru realizarea aplicației

Pentru realizarea aplicației Oauth RPC trebuie să efectuați următoarele sarcini:

- 10 puncte - definirea interfeței care expune tipurile de date și metodele aplicației;
- 10 puncte - descrierea specificațiilor de funcționare (prin comentarii în interfață) pentru toate tipurile de date și funcțiile definite în interfață;
- 45 de puncte - implementarea aplicației server
  - 5 puncte - gestionarea bazei de date cu informațiile utilizatorilor și ale clienților;
  - 15 puncte - procesul de autorizare a clienților (fără reînnoire);
  - 15 puncte - procesul de reînnoire a jetoanelor de acces;
  - 5 puncte - procesul de verificare a operațiilor executate de clienți;
  - 5 puncte - procesul utilizatorului final de aprobare a cererii de autorizare.
- 35 de puncte - implementarea aplicației client
  - 15 puncte - procesul de autorizare (obținerea jetoanelor de acces, fără reînnoire);
  - 15 puncte - procesul de reînnoire a jetoanelor de acces;
  - 5 puncte - execută operațiile descrise în fișier.

# 5 Flux de execuție

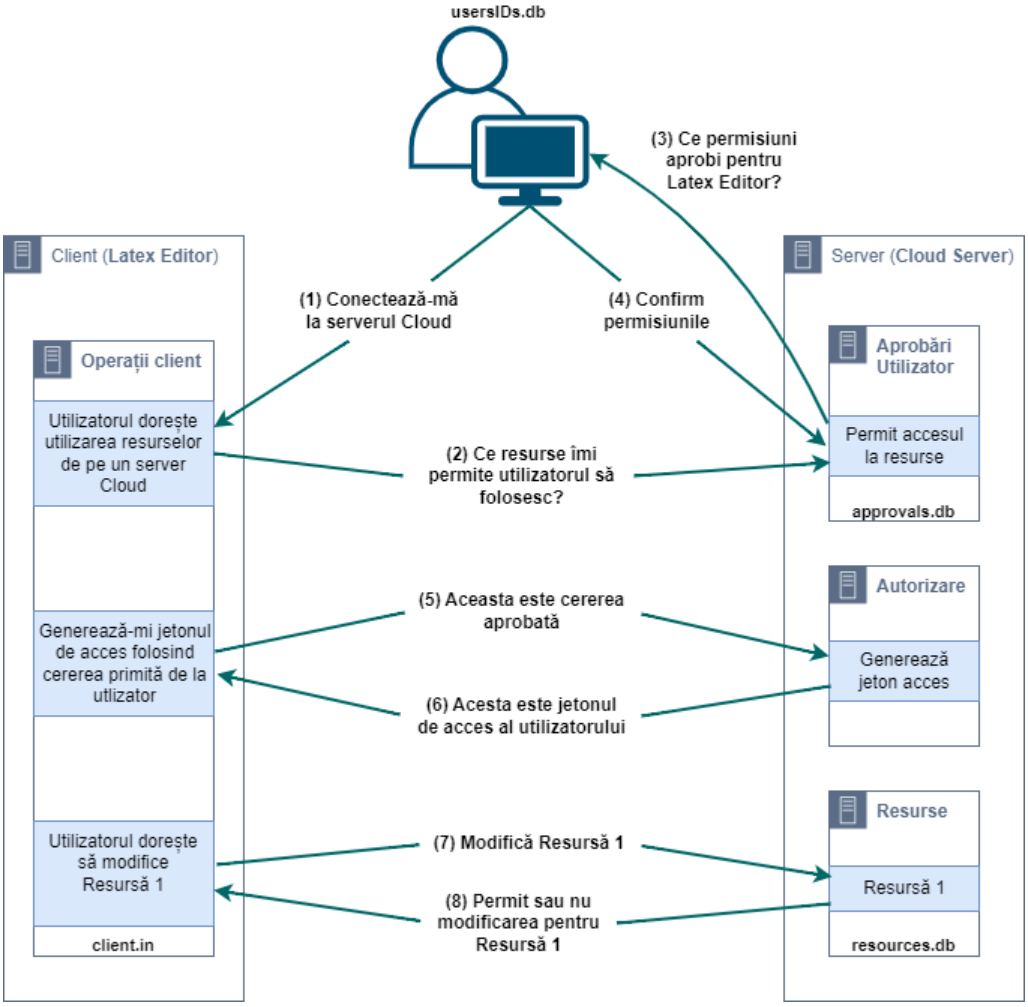


Figura 2: Prezentare vizuală a fluxului de execuție.

# 6 Observații

- Pot fi definite proceduri suplimentare cu condiția ca acestea să fie însoțite de un comentariu care să descrie utilitate/necesitatea acestora.
- Implementați modelul de autorizare Oauth descris în enunț (ordinea cererilor/răspunsurilor).
- Compilatorul *rpcgen* suprascrie fișierele.
- Scriptul *check.sh* și testele puse la dispoziție:
  - fișierul *check.sh* și directorul *tests* trebuie să se afle la aceeași cale ca executabilele;
  - parametrii de la începutul scriptului pot fi modificați în funcție de implementarea voastră;
  - singura modificare permisă asupra fișierelor de test este adăugarea unui newline la finalul fișierelor de ieșire (expected_output) dacă considerați că vă ajută;
  - reprezintă un utilitar care vă ușurează testarea implementării;
  - nu garantează obținerea punctajului maxim.
- Menționați în README dacă ați modificat stub-urile client/server.
  - Justificați necesitatea modificărilor efectuate.
  - Nu vom rula rpcgen dacă în arhivă găsim toate fișierele necesare compilării executabilelor.
- Includeți în arhivă un Makefile care să conțină cel puțin regula de *build* și *clean*.
- Vă încurajăm să folosiți un sistem pentru controlul versiunii pe parcursul rezolvării temei, păstrând privat nivelul de acces la soluție.
- Notare:
  - Cele 20 de puncte pentru interfață și descrierea specificațiilor de funcționare, nu vor fi acordate dacă:
    - Interfața nu este completă, adică nu conține toate funcțiile și structurile minim necesare pentru rezolvarea temei.
    - Comentariile nu denotă înțelegerea cerinței și a algoritmului implementat.
    - Nu s-a încercat implementarea niciuneia din componentele server sau client.
  - Încărcarea unei arhive incomplete sau a unei teme care nu compilează atrage după sine pierderea integrală a punctajului.
  - Corectarea se face folosind distribuția de linux Ubuntu 22.04.5 LTS.