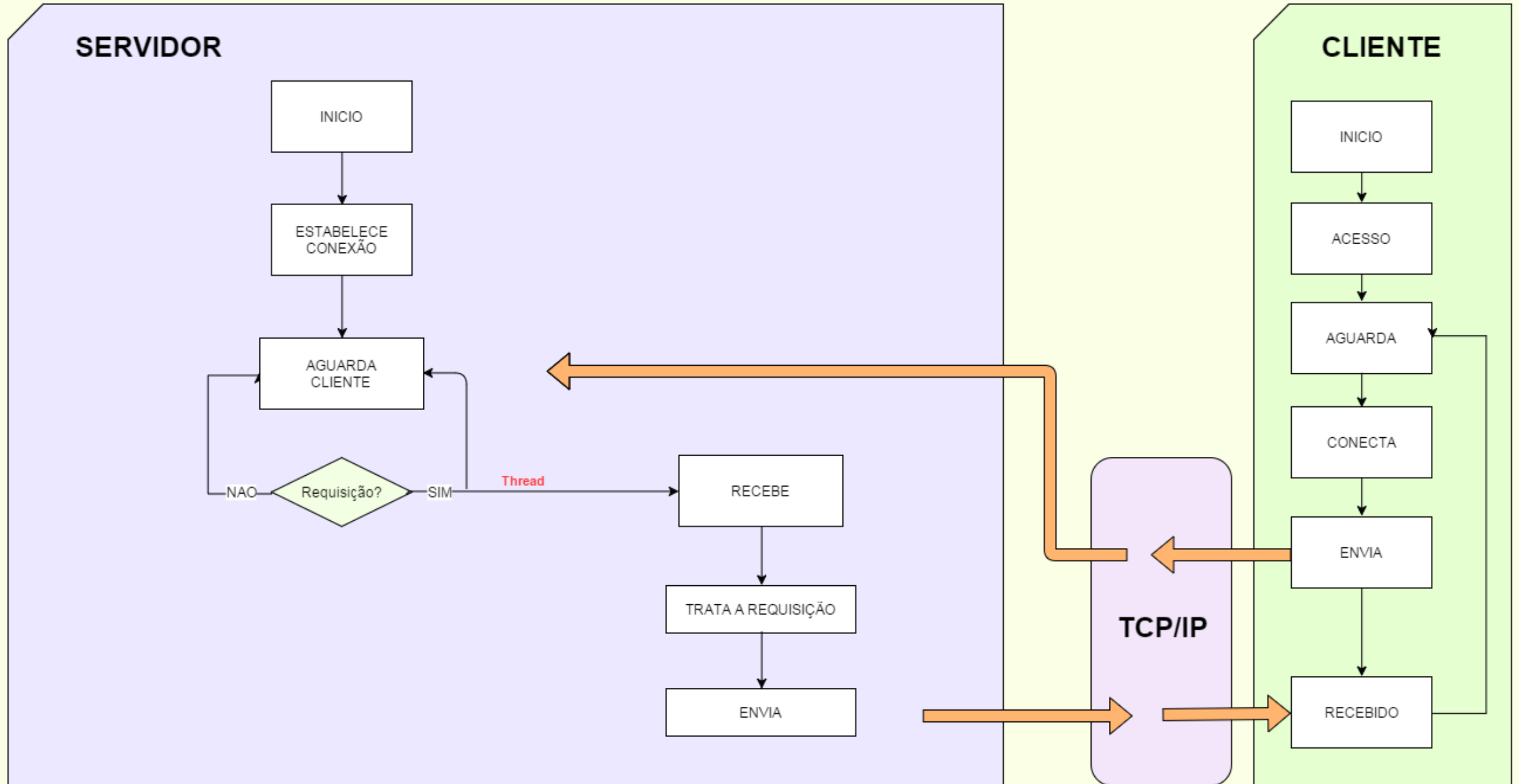


SGBD em C/C++ DDL/DML

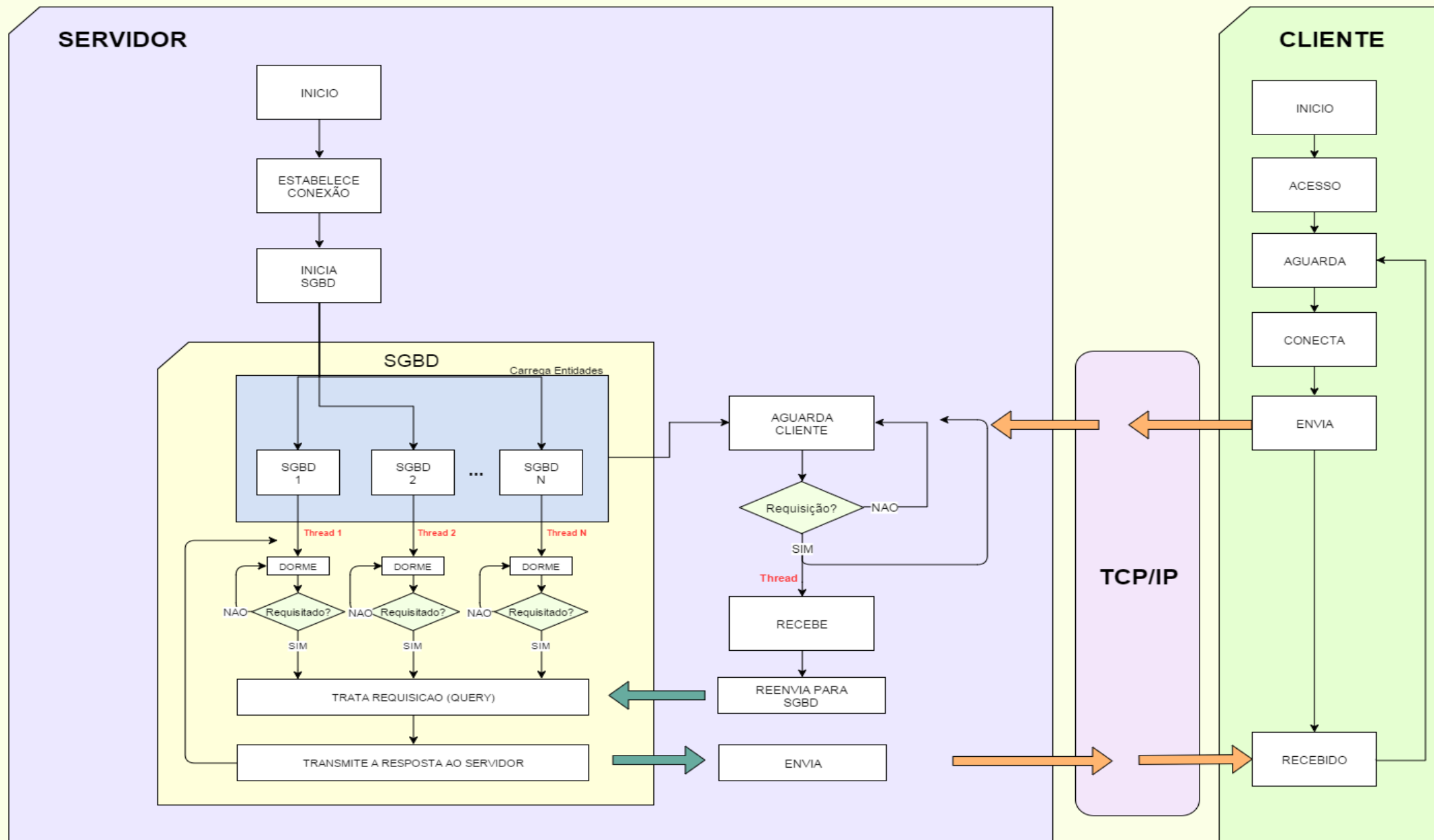
Max Jeison Prass

- Roda em um Servidor;
- São um conjunto de programas que rodam no servidor e que fornecem acesso à dados;
- Gerenciamento de arquivos:
 - Segurança, atomicidade, integridade dos dados, etc;
- Atender grande parte das requisições do Servidor (*Querys*);
- Fornece uma Interface ao usuário para acesso à dados (API's);
- Fornece uma linguagem que permite criação/alteração/consulta dos dados:
 - DDL – Data Definition Language: create, drop, altere,
 - DML – Data Manipulation Language: select, insert, update, delete;
 - DCL – Data Control Language: grant, revoke, connect, select, execute, etc.

Servidor de Mensagens



Servidor de Banco de Dados



Máquina de estados

Descrição

- O SERVIDOR é iniciado, estabelece a conexão e inicializa o SGBD;
- O SGBD, ao ser iniciado, realiza a leitura de todas as tabelas do banco:
 - Para cada Entidade é criada uma Thread SGBD. Todas elas iniciam-se dormindo;
 - O servidor mantém uma lista de Entidades e uma HANDLE em cada Entidade, para garantir a exclusão mútua do acesso aos arquivos;
- O Servidor então vai para o estado de AGUARDAR CLIENTE, para aguardar uma requisição;
- Quando a requisição chega o servidor repassa a informação para o SGBD, iniciando uma thread, e retorna para o estado AGUARDAR CLIENTE;
- A Thread SGBD identifica a Entidade que será acionada e o “acorda”, esta Entidade irá tratar a requisição do usuário e retornar ao cliente através do Servidor;
- Após a Entidade entregar a mensagem ele passa a dormir novamente.

```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
            //return 1; //fecha conexão
        }
        else
        {
            SERVERTHREAD *t = new SERVERTHREAD(server);
        }
    }

    server.Stop(NULL, "closed.");
    system("pause");

    return 1;
}
```

- Instancia a classe *SERVER*
- Inicializa comunicação socket
- Matém um loop de execução:
 - Aguarda cliente;
 - Abre uma *thread server*.

```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para at
         */
        if (!server.WaitClient())
        {
```


- Reinicia *ClienteSocket*
- Inicializa o SGDB
- Estabelece conexão com cliente
- Atende solicitações dos clientes

1.1 Iniciando a primeira classe "SERVER"

```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
```



```
Server::Server()
{
    ClientSocket = INVALID_SOCKET;
    ListenSocket = INVALID_SOCKET;
    struct addrinfo *Result = NULL;
}
```



```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
```

```
bool Server::Start()
{
    this->Connect();
    this->SGDB = new SGDB();

    return true;
}
```

```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
```

```
bool Server::Start()
{
    this->Connect();
    this->SGDB = new SGDB();

    return true;
}
```

```
class SGDB
{
    private:
        Entity *ENTITYI;
        Entity *ENTITYF;

    public:
        SGDB();


        bool LoadEntities();
        bool RequisitionQuery(Bcp **bcp);
        bool Query(string query);

        Entity* EntityFindByTable (string table);
};
```

- Matém uma fila de entidades ativas, mas dormindo.
- Realiza o tratamento das *queries* quando acordadas.

1.5 Carregando o banco de dados

```
bool Server::Start()  
{  
    this->Connect();  
    this->SGDB = new SGDB();  
    return true;  
}
```



```
SGDB::SGDB()  
{  
    // Carrega todas as entidades do banco de dados  
    this->LoadEntities();  
}
```

- O construtor da classe carrega todas as Entidades.

```
bool SGDB::LoadEntities()
{
    DIR *dir;
    struct dirent *lsdir;

    dir = opendir(FOLDERTABLES);

    std::regex regexfindfile("(\\.+)([.])\\.+");

    /* LEITURA DE TODOS OS ARQUIVOS (TABELAS) DO BANCO DE DADOS */
    while ( ( lsdir = readdir(dir) ) != NULL )
    {
        if (std::regex_search(lsdir->d_name, regexfindfile))
        {
            Entity *entity = new Entity(lsdir->d_name);

            ENTITY_LOAD(&this->ENTITYI, &this->ENTITYF, entity);

            SGBDTHREAD *sgbdthread = new SGBDTHREAD(entity);
        }
    }

    return true;
}
```

```
class Entity
{
private:
    string table;
    HANDLE mutex;
public:
    Entity();

    void setTable(string table);
    string getTable();

    void setMutex(HANDLE handle);
    HANDLE getMutex();

    Entity *right;
    Entity *left;
};
```

```
bool SGDB::LoadEntities()
{
    DIR *dir;
    struct dirent *lmdir;

    dir = opendir(FOLDERTABLES);

    std::regex regexfindfile("(\\.+)([.])(\\.+");

    /* LEITURA DE TODOS OS ARQUIVOS (TABELAS) DO BANCO DE DADOS */
    while ( ( lmdir = readdir(dir) ) != NULL )
    {
        if (std::regex_search(lmdir->d_name, regexfindfile))
        {
            Entity *entity = new Entity(lmdir->d_name);

            ENTITY_LOAD(&this->ENTITYI, &this->ENTITYF, entity);

            SGBDTHREAD *sgbdthread = new SGBDTHREAD(entity);
        }
    }

    return true;
}
```

- NOME DA TABELA


```
Entity::Entity(string table)
{
    // Salva nome da entidade;
    this->table = table;

    // Cria SEMAFORO para controle de execucao. A primeira execucao aguarda o metodo "Start"
    this->mutex = CreateSemaphore(NULL, 0, 1, (LPCWSTR)(table.c_str()));

    // Cria SEMAFORO para controle de alteracao da Query. Deixa a primeira alteracao livre.
    string mutexquery(table + "Q");
    this->mutexQuery = CreateSemaphore(NULL, 1, 1, (LPCWSTR)(mutexquery.c_str()));
}
```

- São dois semáforos:
 - 1º para controle de execução o 2º para segurança dos dados e fila.

```
bool SGDB::LoadEntities()
{
    DIR *dir;
    struct dirent *lsdir;

    dir = opendir(FOLDERTABLES);

    std::regex regexfindfile("(\\.+)([.])\\.+");

    /* LEITURA DE TODOS OS ARQUIVOS (TABELAS) DO BANCO DE DADOS */
    while ( ( lsdir = readdir(dir) ) != NULL )
    {
        if (std::regex_search(lsdir->d_name, regexfindfile))
        {
            Entity *entity = new Entity();
            entity->setTable(lsdir->d_name);

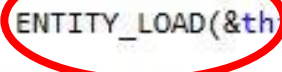
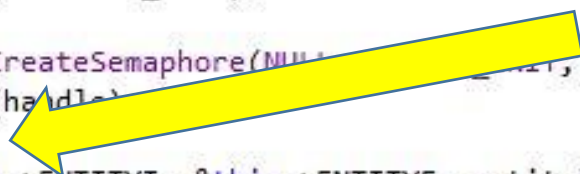
            HANDLE handle = CreateSemaphore(NULL, 1, THREAD_COUNT, (LPCWSTR)(lsdir->d_name));
            entity->setMutex(handle);

            ENTITY_LOAD(&this->ENTITYI, &this->ENTITYF, entity);

            SGBDTHREAD *sgbdthread = new SGBDTHREAD(entity);
        }
    }

    return true;
}
```

SALVANDO NA LISTA




```
bool SGDB::LoadEntities()
{
    DIR *dir;
    struct dirent *lmdir;

    dir = opendir(FOLDERTABLES);

    std::regex regexfindfile("(\\.+)([.])\\.+");

    /* LEITURA DE TODOS OS ARQUIVOS (TABELAS) DO BANCO DE DADOS */
    while ( ( lmdir = readdir(dir) ) != NULL )
    {
        if (std::regex_search(lmdir->d_name, regexfindfile))
        {
            Entity *entity = new Entity();
            entity->setTable(lmdir->d_name);

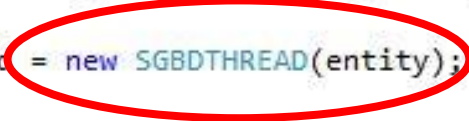
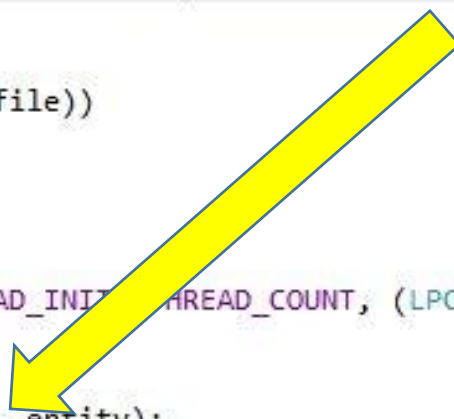
            HANDLE handle = CreateSemaphore(NULL, THREAD_INITIAL_COUNT, THREAD_COUNT, (LPCWSTR)(lmdir->d_name));
            entity->setMutex(handle);

            ENTITY_LOAD(&this->ENTITYI, &this->ENTITYF, entity);

            SGBDTHREAD *sgbdthread = new SGBDTHREAD(entity);
        }
    }

    return true;
}
```

AGORA A
ENTIDADE É
UMA THREAD



1.11

A Thread entra em loop de Execução e dorme na primeira iteração. Só executando após ser acordada pelo servidor.

```
class SGBDTHREAD: public Thread
{
    Entity *entity;

public:
    SGBDTHREAD(Entity *entity)
    {
        this->entity = entity;
        Thread::CreateNewThread(this);
    }

    void Run(void*)
    {
        FILE *file = NULL;
        string folder = FOLDERTABLES;
        folder += (*this->entity).getTable();
        file = fopen(folder.c_str(), "r");

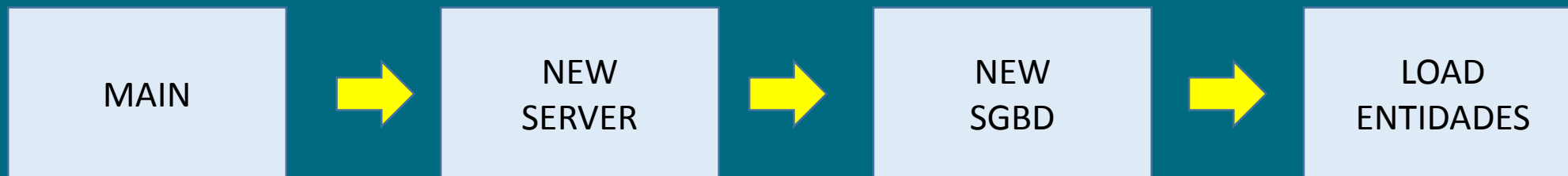
        while (true)
        {
            WaitForSingleObject((*this->entity).getMutex(), INFINITE);

            //(...) CODIGO DAS DML - Data Manipulation Language.

        }
    }
};
```

O QUE VIMOS ATÉ AQUI?

- Inicialização da classe SERVER pelo sistema;
- Inicialização do SGBD;
- Carrega todas as Entidades (Executando em paralelo)



```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
            //return 1; //fecha conexão
        }
        else
        {
            SERVERTHREAD *t = new SERVERTHREAD(server);
        }
    }
}
```

- Mas como atender as requisições?

```
int main()
{
    /*
     * Inicia servidor
     */
    Server server;

    /*
     * Inicia servidor
     */
    if (!server.Start())
    {
        return 1;
    }

    while (true)
    {
        /*
         * Aguarda entrada de usuario e abre a thread para atende-la.
         */
        if (!server.WaitClient())
        {
            //return 1; //fecha conexão
        }
        else
        {
            SERVERTHREAD *t = new SERVERTHREAD(server);
        }
    }
}
```

- Pela *thread Server!*



```
class SERVERTHREAD: public Thread
{
    Server server;

public:
    SERVERTHREAD(Server server)
    {
        this->server = server;
        Thread::CreateNewThread(this);
    }

    void Run(void*)
    {
        if (!this->server.Recv())
        {
            return;
        }
        else
        {
        }
    }
};
```

- Criada a cada requisição;
- Invoca o método *recv* do servidor.


```
bool Server::Recv()
{
    if (DEBUG)
    {
        printf("Wait for a message... ");
    }

    char recvbuf[BUFLen];
    int iResult;
    char* user = (char*)malloc(sizeof(recvbuf));
    char* msg = (char*)malloc(sizeof(recvbuf));
    char* action = (char*)malloc(sizeof(recvbuf));

    iResult = recv(ClientSocket, recvbuf, BUFLen, 0);

    if (iResult > 0)
    {
        // (...) TRECHO DE CODIGO RECEIVED

        REQUISITIONTHREAD *t = new REQUISITIONTHREAD(*this, dado);
    }
    else if (iResult == 0)
    {

```

- Recebe a mensagem via *socket*
- Cria uma REQUISITIONTHREAD

```
REQUISITIONTHREAD::REQUISITIONTHREAD(Server se
{
    this->server = server;
    this->BCPAtual = bcp;
    Thread::CreateNewThread(this);
}

REQUISITIONTHREAD::~~REQUISITIONTHREAD()
{
    free(BCPAtual);
}

void REQUISITIONTHREAD::Run(void*)
{
    this->server.Run(BCPAtual);
}
```

- Invoca o método do servidor para rodar;
- *E executa!*
- Usada para não des-sincronizar a comunicação estabelecida entre os N clientes.
- (Cliente socket)


```
int main()
{
    /*
     * Inicia servidor
     */
    Server *server = new Server();

    while (true)
    {
        /*
         * Aguarda entrada de usuário e abre a thread para atendê-la.
         */
        if (!server->WaitClient())
        {
            //return 1; //fecha conexão
        }
        else
        {
            server->Requisition();
        }
    }

    server->Stop(NULL, "closed.");
    system("pause");

    return 1;
}
```

- E volta a aguardar novo cliente!

```
// Recebe as mensagens (BCP) e repassa ao SGBD.  
bool Server::Run(Bcp *bcp)  
{  
    this->SGBD->RequisitionQuery(&bcp);  
  
    // Imprime o log da requisicao.  
    this->LOG(bcp);  
  
    // Envia acknowledgment  
    this->Request(bcp);  
  
    return true;  
}
```

- O servidor repassa a requisição para o SGBD.

```
bool SGDB::RequisitionQuery(Bcp **bcp)
{
    int action = GETACTIONCODE((*bcp)->action);

    // TESTANDO - Acordar SGBD de uma tabela especifica
    Entity *entity = this->EntityFindByTable("pfisica");
    if (entity != NULL)
    {
        ReleaseSemaphore(entity->getMutex(), 1, NULL);
    }

    return false;
}
```

- A Entidade é acordada e passa a executar a *Thread* correspondente.

3.2 TRATANDO REQUISIÇÕES

```
class SGBDTHREAD: public Thread
{
    Entity *entity;

public:
    SGBDTHREAD(Entity *entity)
    {
        this->entity = entity;
        Thread::CreateNewThread(this);
    }

    void Run(void*)
    {
        FILE *file = NULL;
        string folder = FOLDERTABLES;
        folder += (*this->entity).getTable();
        file = fopen(folder.c_str(), "r");

        while (true)
        {
            WaitForSingleObject((*this->entity).getMutex(), INFINITE);

            //(...) CODIGO DAS DML - Data Manipulation Language.

        }
    }
};
```

```
bool SGBD::RequisitionQuery(Bcp **bcp)
{
    int action = GETACTIONCODE((*bcp)->action);

    // TESTANDO - Acordar SGBD de uma tabela especifica
    Entity *entity = this->EntityFindByTable("pfisica");
    if (entity != NULL)
    {
        ReleaseSemaphore(entity->getMutex(), 1, NULL);
    }

    return false;
}
```

1

3

2

1. ACORDA
2. EXECUTA
3. DORME

Os modelos definem como os dados serão armazenados no banco de dados. Os mais conhecidos são:

- **Herárquico:** Um registro tem apenas um pai.
- **Em Rede:** Um registro pode estar associado a várias entidades, os dados contém apontadores a outros dados.
- **Relacional:** Não existem apontadores. Permitem a construção de *querys* que não foram antecipadas por quem projetou a base de dados. O mais utilizado: Mysql, Oracle, Acces, entre outros.
- **Orientado a Objetos:** As tabelas são definidas como objetos, e só podem ser acessados pelos seus métodos, pré-definidos. Contém referência a outros objetos.

Existem outras...

Arquivo de Dados

ENTIDADE
Atributo 1
Atributo 2
Atributo 3
...
Atributo N

Arquivo de Elos

Nome	Dep.	Nome	Diretor
Pedro	Vendas	Vendas	Mario



Implícito

Nome	Dep.	Nome	Diretor
Pedro	• →	Vendas	Mario

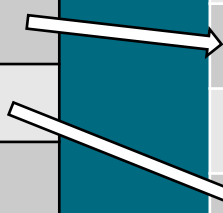


Explícito

Arquivo de Inversões

ENG	S1
ENG	S2

Nome	Categoria
Pedro	ENG
Rui	ENG



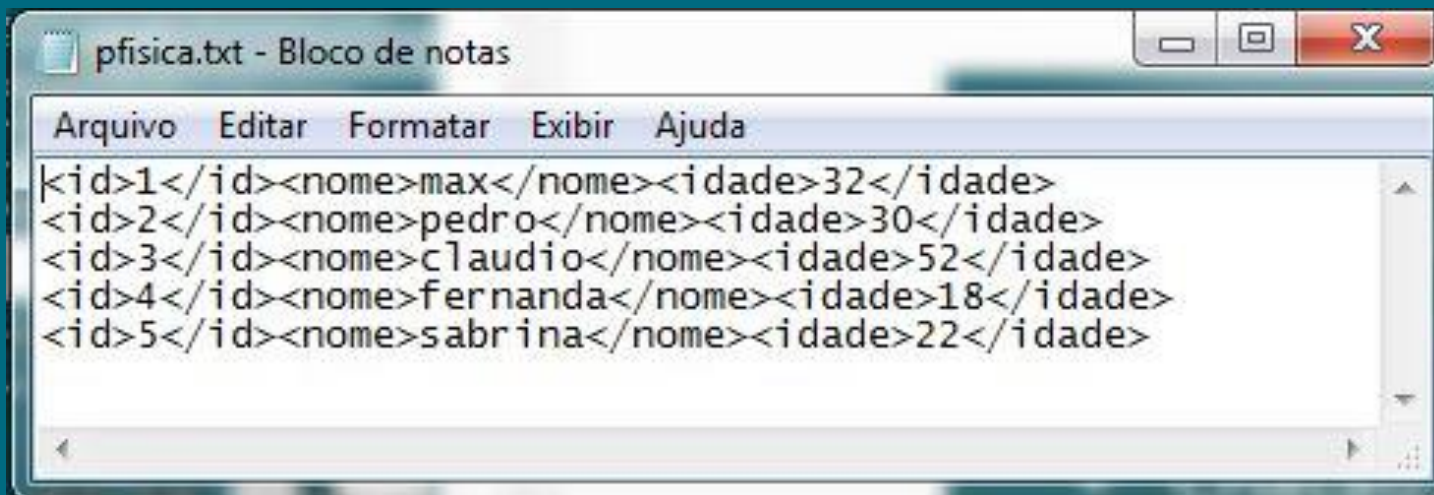
Inúmeros fatores devem ser levados em conta:

- Analisar as atividades dos equipamentos de armazenamento e canais de tráfego de informação;
- Frequência de acesso em cada arquivo;
- Frequência de uso e tempo gasto por operação: *insert, select, etc*;
- Espaço utilizado por arquivo;
- Número de registros por classe de inversão;
- Entre outros...

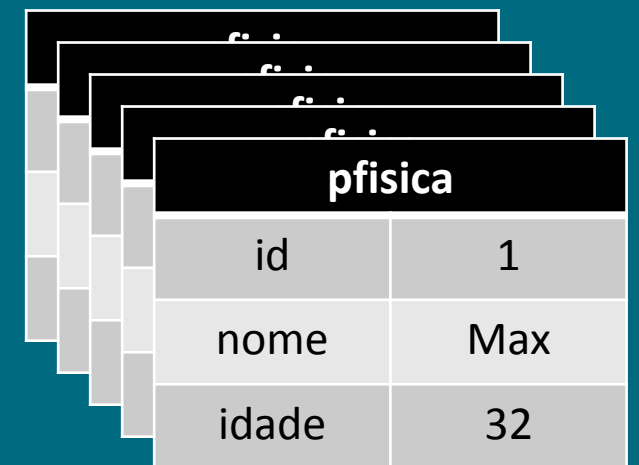
Queda de desempenho -> Reorganização do SGBD -> Custo elevado

- Aconselhável um SISTEMA DE MEDIÇÃO da eficiência do SGBD logo no início.

- Criação de uma estrutura básica (apenas para demonstração).
- As tabelas do banco são salvo na tabela 'Entities' do projeto:
- Exemplo:



```
<id>1</id><nome>max</nome><idade>32</idade>
<id>2</id><nome>pedro</nome><idade>30</idade>
<id>3</id><nome>claudio</nome><idade>52</idade>
<id>4</id><nome>fernanda</nome><idade>18</idade>
<id>5</id><nome>sabrina</nome><idade>22</idade>
```



pfisica	
id	1
nome	Max
idade	32

- Um sistema gerenciador de banco de dados necessita de um servidor *multithread* para que seja capaz de atender múltiplas requisições ao mesmo tempo (CONCLUÍDO);
- Agora, você só tem pegar este projeto e atribuir as suas próprias manipulação dos dados, e já poderá oferecer um serviço de banco de dados;
 - São as manipulações dos dados DDL e DML;
- Fique à vontade, use-o para as suas aulas, seus trabalhos ou até mesmo tentar ajudar a tornar isto real.

Obrigado!

PERGUNTAS?

Max J. Prass