Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Приложение для генерации кода загружаемых модулей ядра, реализующих пару клиент-сервер, работающих по протоколу ONC RPC, на основе файла шаблона (RPCL)

Студент **ИУ7-71Б** _____ **Лагутин Д. В.**

Руководитель _____ **Рязанова Н. Ю.**

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
«__» _____ 20__ г.

З А Д А Н И Е
на выполнение курсовой работы

по дисциплине

Операционные системы

Студент группы **ИУ7-71Б**

Лагутин Даниил Валерьевич

Тема курсовой работы

Приложение для генерации кода загружаемых модулей ядра, реализующих пару клиент-сервер, работающих по протоколу ONC RPC, на основе файла шаблона (RPCL)

График выполнения работы: 25% к 6 нед., 50% к 9 нед., 75% к 12 нед., 100% к 15 нед.

Техническое задание

Разработать приложение для генерации кода пары загружаемых модулей ядра, реализующих пару клиент-сервер, работающих по протоколу ONC RPC. Обеспечить поддержку спецификации описанной на языке RPC Language. Обеспечить возможность раздельной загрузки модулей, а также конфигурации основных параметров: ip адрес и номер порта сервера, номер используемой версии.

Оформление курсовой работы:

Расчетно-пояснительная записка на **25-40** листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т. п.)

Дата выдачи задания «26» сентября 2024 г.

Руководитель курсовой работы _____ **Н. Ю. Рязанова**
(Подпись, дата) (И.О.Фамилия)

Студент _____ **Д. В. Лагутин**
(Подпись, дата) (И.О.Фамилия)

Содержание

Введение

ONC RPC (Open Network Computing Remote Procedure Call) — реализация системы удаленного вызова процедур, разработанная как часть сетевой файловой системы NFS [1]. Из-за широкого распространения [2], для технология была разработана утилита генерации кода клиента и сервера на основе языка RPCL (Remote Procedure Call Language) [2] — `rpcgen`. Однако, данная программа предназначена исключительно для пространства пользователя. Целью данной работы является разработка приложения для генерации кода пары загружаемых модулей ядра Linux, реализующих модель клиент-сервер и позволяющие использовать технологию ONC RPC согласно спецификации на языке RPCL в ядре операционной системы, подобно `rpcgen`.

Задачи:

1) провести анализ протокола RPC;

2) рассмотреть структуры и функции ядра, реализующие выбранную технологию;

3) определить требуемый функционал приложения и разработать алгоритм работы как приложения-генератора, так и соответствующих загружаемых модулей;

4) проанализировать результаты работы разработанного приложения.

1 Аналитическая часть

1.1 Постановка задачи

В соответствии с заданием на курсовую работу необходимо разработать приложение, генерирующее код загружаемых модулей ядра клиента и сервера, взаимодействующих по протоколу ONC RPC, согласно описанной спецификации.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ составных частей протокола, способов представления и передачи информации;

2) рассмотреть структуры и функции ядра, реализующие клиентскую и серверную стороны будущего приложения, определить порядок их инициализации и вызова;

3) разработать алгоритм загружаемых модулей ядра;

4) разработать алгоритм разбора файла спецификации и генерации соответствующих модулей на его основании;

5) проанализировать результаты работы разработанного приложения.

1.2 Анализ протокола ONC RPC

Программа реализующая протокол ONC RPC называется сервисом и предоставляет доступ к одной или нескольким программам, которые, в свою очередь, реализуют одну или несколько удаленных процедур. Процедуры, их параметры и результаты документированы в спецификации протокола конкретной программы. Сервис поддерживает версионирование программ для поддержания совместимости с изменяющимися протоколами [2].

1.2.1 Анализ модели взаимодействия

Протокол ONC RPC основан на модели удаленного вызова процедур, которая аналогична модели локального вызова процедур, за исключением того, что поток управления проходит через два процесса: клиента и сервера [2]. Клиент посылает сообщение серверу и блокируется до получения ответа. Сообщение вызова содержит параметры процедуры, а ответа — результаты работы процедуры. Когда ответ получен, результаты процедуры извлекаются, и процесс на клиентской стороне возобновляется.

Ожидая поступления сообщений, процесс сервера засыпает, а при получении — извлекает параметры процедуры, вызывает обработчик для вычисления результата и отправляет ответ.

Таким образом с каждый момент времени активен только один процесс, однако протокол не накладывает никаких ограничений на реализуемую модель, и возможны другие варианты организации параллельных вычислений [2].

Протокол RPC может быть реализован для различных транспортных протоколов, при этом реализация протокола не регламентирует передачу сообщений от одного процесса к другому и включает в себя только спецификацию и способ интерпретации сообщений. Привязки конкретного клиента к конкретному сервису и транспортными параметрами также не является частью протокола RPC [2]. Для этого используются специальные приложения `portmapper`, в частности `rpcbind`, для связи информации о доступных сервисах и способах доступа к ним [3].

1.2.2 Анализ структурных частей протокола

Как уже было описано ранее сервис состоит из программ, которые содержат некоторое количество процедур, потенциально различных версий. Для их описания используется спецификация на языке RPC Language, который является расширением стандарта XDR для кодирования данных [2][4]. Грамматика расширения языка представлена на листинге 1.1 в форме Бэкуса-Наура.

Листинг 1.1 – Структура RPCL

```
program-def:
    "program" identifier "{"
        version-def
        version-def *
    "}" "=" constant ";"
version-def:
    "version" identifier "{"
        procedure-def
        procedure-def *
    "}" "=" constant ";"
procedure-def:
    proc-return identifier "("
        proc-firstarg "," type-specifier )*
    ")" "=" constant ";"
proc-return: "void" | type-specifier
proc-firstarg: "void" | type-specifier
```

Ограничения [2]:

— ключевые слова `program` и `version` не могут быть использованы как идентификаторы;

— идентификаторы программы и связанные с ними номера не могут повторяться в контексте спецификации;

— идентификаторы и связанные с ними номера версии не могут повторяться в контексте программы;

— идентификаторы и связанные с ними номера процедуры не могут повторяться в контексте версии;

— в качестве номеров могут быть использованы только беззнаковые целые числа.

1.2.3 Анализ типов данных стандарта XDR

Стандарт XDR в свою очередь предоставляет не только язык для описания типов данных, но и формат для их представления. Структура описания XDR представлена на листинге 1.2.

Листинг 1.2 – Структура XDR

```
declaration:
    type-specifier identifier
    | type-specifier identifier "[" value "]"
    | type-specifier identifier "<" [ value ] ">"
    | "opaque" identifier "[" value "]"
    | "opaque" identifier "<" [ value ] ">"
    | "string" identifier "<" [ value ] ">"
    | type-specifier "*" identifier
    | "void"
value:
    constant | identifier
type-specifier:
    [ "unsigned" ] "int" | [ "unsigned" ] "hyper"
    | "float" | "double" | "quadruple"
    | "bool"
    | enum-type-spec | struct-type-spec | union-type-spec
    | identifier
enum-type-spec:
    "enum" enum-body
enum-body:
    "{" ( identifier "=" value ) ( "," identifier "=" value )* "}"
struct-type-spec:
    "struct" struct-body
struct-body:
    "{" ( declaration ";" ) ( declaration ";" )* "}"
union-type-spec:
    "union" union-body
union-body:
    "switch" "(" declaration ")" "{"
    ( "case" value ":" declaration ";" )
    ( "case" value ":" declaration ";" )*
    [ "default" ":" declaration ";" ]
    "}"
constant-def:
    "const" identifier "=" constant ";"
type-def:
    "typedef" declaration ";"
    | "enum" identifier enum-body ";"
    | "struct" identifier struct-body ";"
    | "union" identifier union-body ";"
definition:
    type-def | constant-def
specification:
    definition *
```

Нотация копирует синтаксис языка C, с добавлением некоторых особенностей [4]:

— нотация `"<" [ value ] ">"` означает массив переменной длины с возможным ограничением максимального количества элементов в `value` элементов;

— аналог объявления указателя из языка C, означает необязательное поле и кодируется как массив переменной длины с ограничением в 1 элемент;

— данные с типом `opaque` передаются без изменений (как массив байт);

— строка `string` нуль терминирована и передается как массив байт;

— `bool` — беззнаковое целое;

— вещественные числа передаются как есть, так как документированы стандартом IEEE 754.

Для представления данных XDR использует порядок байт от старшего к младшему и объединяет данные в блоки по 4 байта, поэтому длина любого сообщения должна быть кратна 4. Элементы массивов, структур кодируются в порядке описания (появления). В случае передачи массивов переменной длины, их размер записывается перед телом как беззнаковое 4 байтовое целое. Дискриминант объединения также записывается перед телом, аналогично как 4 байтовое целое [4].

1.3 Анализ поддержки в ядре Linux

В ядре реализация RPC представлена в составе модуля `sunrpc`, который может быть скомпилирован как часть ядра или загружаемый модуль.

Перед использованием, модуль должен быть загружен при помощи команды `modprobe sunrpc`, после чего разработчику становится доступна кодовая база для написания клиента и сервера приложения. В дальнейшем, структуры и функции будут рассматриваться для версии ядра 6.12.7 [5].

1.3.1 Анализ инициализации и порядка работы сервера

Сервер описывается структурой `svc_serv` (листинг 1.3) и называется сервисом, который в свою очередь является демоном, отвечающим за прием и обработку сообщений, а так же поддержание потоков обработчиков.

Листинг 1.3 – Структура сервиса

```
struct svc_serv {
    /* Массив программ и его размер */
    struct svc_program *    sv_programs;
    unsigned int            sv_nprogs;
    /* Число потоков обработчиков */
    unsigned int        sv_nrthreads;
    /* Максимальное количество соединений.
     * Равно числу потоков, если указан 0 */
    unsigned int        sv_maxconn;
    struct svc_stat *   sv_stats;
    spinlock_t          sv_lock;
    /* Размеры буфера и сообщения */
    unsigned int        sv_max_payload;
    unsigned int        sv_max_mesg;
    unsigned int        sv_xdrsize;
    /* Списки сокетов */
    struct list_head    sv_permsocks;
    struct list_head    sv_tempsocks;
    int                 sv_tmpcnt;
    struct timer_list   sv_temptimer;
    char *          sv_name;
    /* Пул потоков */
    unsigned int        sv_nrpools;
    bool                sv_is_pooled;
    struct svc_pool *   sv_pools;
    /* Функция потока обработчика */
    int         (*sv_threadfn)(void *data);
#if defined(CONFIG_SUNRPC_BACKCHANNEL)
    struct lwq      sv_cb_list;
    bool            sv_bc_enabled;
#endif /* CONFIG_SUNRPC_BACKCHANNEL */
};
```

Сервис может быть создан в однопоточном или многопоточном варианте с использованием функций `svc_create` и `svc_create_pooled` (листинг 1.4), которые приводят последующему вызову `__svc_create`. Функция создает и инициализирует структуру `svc_serv`, из примечательных моментов, макси-

мальный размер сообщения sv_max_payload имеет значение по умолчанию, равное одной странице (при использовании значения 0 для параметра bufsize).

Листинг 1.4 – Создание сервиса

```
struct svc_serv *svc_create(struct svc_program *prog,
                            unsigned int bufsize,
                            int (*threadfn)(void *data));
struct svc_serv *svc_create_pooled(struct svc_program *prog,
                                   unsigned int nprog,
                                   struct svc_stat *stats,
                                   unsigned int bufsize,
                                   int (*threadfn)(void *data));
```

Программа, ее версии и процедуры описываются следующими структурами, представленными на листингах 1.5–1.7. Соответствующие массивы версий и процедур могут быть определены как статически, так и динамически.

Листинг 1.5 – Структура программы

```
struct svc_program {
    /* Номер программы */
    u32          pg_prog;
    /* Минимальная и максимальная обслуживаемые версии */
    unsigned int         pg_lovers;
    unsigned int         pg_hivers;
    /* Массива версий и его размер */
    unsigned int         pg_nvers;
    const struct svc_version **pg_vers;
    /* Название программы */
    char *           pg_name;
    /* Название класса прграмм, разделяющих аутентификацию,
     * используется общий кеш */
    char *           pg_class;
    /* Пользовательский обработчик аутентификации */
    enum svc_auth_status    (*pg_authenticate)(struct svc_rqst *rqstp);
    /* Функция инициализации контекста потока */
    __be32           (*pg_init_request)(struct svc_rqst *,
                           const struct svc_program *,
                           struct svc_process_info *);
    /* Функция регистрации сервиса (portmapper) */
    int          (*pg_rpcbind_set)(struct net *net,
                           const struct svc_program *,
                           u32 version, int family,
                           unsigned short proto,
                           unsigned short port);
};
```

Листинг 1.6 – Структура версии

```
struct svc_version {
    /* Номер версии */
    u32          vs_vers;
    /* Массив процедур и их количество */
    const struct svc_procedure *vs_proc;
    u32                         vs_nproc;
    unsigned long __percpu  *vs_count;
    /* Размер буфера сообщения */
    u32          vs_xdrsize;
    /* Используется ли portmapper */
    bool             vs_hidden;
    /* Игнорируется ли ошибка регистрации */
    bool             vs_rpcb_optnl;
    bool             vs_need_cong_ctrl;
    /* Функция диспетчеризации */
    int          (*vs_dispatch)(struct svc_rqst *rqstp);
};
```

Листинг 1.7 – Структура процедуры

```
struct svc_procedure {
    /* Реализация функции обработчика процедуры */
    __be32           (*pc_func)(struct svc_rqst *);
    /* Кодирование и декодирование */
    bool             (*pc_decode)(struct svc_rqst *rqstp,
                          struct xdr_stream *xdr); // аргументы
    bool             (*pc_encode)(struct svc_rqst *rqstp,
                          struct xdr_stream *xdr); // результат
    /* Очистка использованных значений */
    void             (*pc_release)(struct svc_rqst *);
    /* Размер аргумента и количество байт, которые должны быть очищены */
    unsigned int         pc_argsize;
    unsigned int         pc_argzero;
    /* Размер результата и максимальный размер сообщения */
    unsigned int         pc_ressize;
    unsigned int         pc_xdrressize;
    unsigned int         pc_cachetype;
    const char *         pc_name;
};
```

Инициализация сервиса не приводит к его регистрации, запуску или созданию транспортной инфраструктуры. Типичная последовательность запуска представлена на листинге 1.8.

Листинг 1.8 – Последовательность регистрации и запуска сервиса

```
1  // 1
2  int svc_bind(struct svc_serv *serv, struct net *net);
3  // 2
4  int svc_xprt_create(struct svc_serv *serv, const char *xprt_name,
5                      struct net *net, const int family,
6                      const unsigned short port, int flags,
7                      const struct cred *cred);
8  // 3
9  int svc_set_num_threads(struct svc_serv *serv, struct svc_pool *pool,
10                         int nrservs);
```

Вызов `svc_bind`, вопреки названию, не регистрирует сервис у portmapper, но удаляет существующие записи других сервисов, если они использую номера вновь созданной программы.

Регистрация же производится при создании нового транспорта в функции `csv_xprt_create`. Зарегистрированные протоколы содержатся в списке `svc_xprt_class_list` и описываются структурой `svc_xprt_class` (листинги 1.9–1.10). В настоящий момент возможно использовать только TCP или UDP. Вызов `svc_xprt_create` осуществляет поиск транспорта по его названию, переданному параметром `xprt_name`, после чего вызывается метод `xpo_create`, значение которого совпадает для обоих протоколов — `svc_create_socket`. Функция создает сокет (`__sock_create`), связывает его с переданным адресом (`kernel_bind`) и вызывает `setup_socket`, где и происходит регистрация нового сервиса вызовом `svc_register`, который вызывает метод `pg_rpcbind_set` текущей программы для каждой ее версии. При использовании rpcbind, в ядре определена стандартная реализация метода — `svc_generic_rpcbind_set`.

Листинг 1.9 – Структура транспортного протокола (часть 1)

```
1  struct svc_xprt_class {
2      const char                *xcl_name;
3      struct module             *xcl_owner;
4      const struct svc_xprt_ops *xcl_ops;
5      struct list_head           xcl_list;
6      u32                        xcl_max_payload;
7      int                        xcl_ident;
8  };
9  struct svc_xprt_ops {
10     struct svc_xprt *(*xpo_create)(struct svc_serv *,
11                                    struct net *net,
12                                    struct sockaddr *, int,
13                                    int);
```

Листинг 1.10 – Структура транспортного протокола (часть 2)

```
    struct svc_xprt *(*xpo_accept)(struct svc_xprt *);
    int              (*xpo_has_wspace)(struct svc_xprt *);
    int              (*xpo_recvfrom)(struct svc_rqst *);
    int              (*xpo_sendto)(struct svc_rqst *);
    int              (*xpo_result_payload)(struct svc_rqst *,
                                           unsigned int,
                                           unsigned int);
    void             (*xpo_release_ctxt)(struct svc_xprt *xprt,
                                         void *ctxt);
    void             (*xpo_detach)(struct svc_xprt *);
    void             (*xpo_free)(struct svc_xprt *);
    void             (*xpo_kill_temp_xprt)(struct svc_xprt *);
    void             (*xpo_handshake)(struct svc_xprt *xprt);
};
```

После чего остается запустить обработчик функцией `svc_set_num_threads`. Вызов создает поток ядра и начинает выполнять функцию из поля `sv_threadfn` сервиса. Ядро не предоставляет готовых решений для обработчика, поэтому его необходимо определить самостоятельно. Минимальный пример реализации представлен на листинге 1.11.

Листинг 1.11 – Минимальная реализация функции потока

```
int threadfn(void *data)
{
    struct svc_rqst *rqstp = data;
    svc_thread_init_status(rqstp, 0);

    while (!svc_thread_should_stop(rqstp)) {
        svc_recv(rqstp);
    }

    svc_exit_thread(rqstp);

    return 0;
}
```

После инициализации в начале обработчика необходимо вызвать функцию `svc_thread_init_status`, передающую статус основному потоку, позволяя ему продолжить работу. В случае заявления ошибки, функция также останавливает и поток обработчика. Далее идет основной поток обработки запросов: `svc_thread_should_stop` — проверяет необходимо ли завершить поток обработчик; `svc_recv` — получает запрос и вызывает обработчик `svc_process`, если в настоящий момент нет доступных запросов, то поток

переключается в состояние idle.

Работа сервиса прекращается повторным вызовом функции `svc_set_num_threads` с параметром `nrservs` равным 0. Получив сигнал к завершению (установкой бита `SP_NEED_VICTIM`), поток должен прекратить работу и вызвать `svc_exit_thread` для очистки контекста потока и информирования основного потока о завершении (сброс бита `SP_VICTIM_REMAINS`).

Обработка запроса происходит внутри функции `svc_process`, алгоритм которой представлен на рисунках 1.1–1.3.

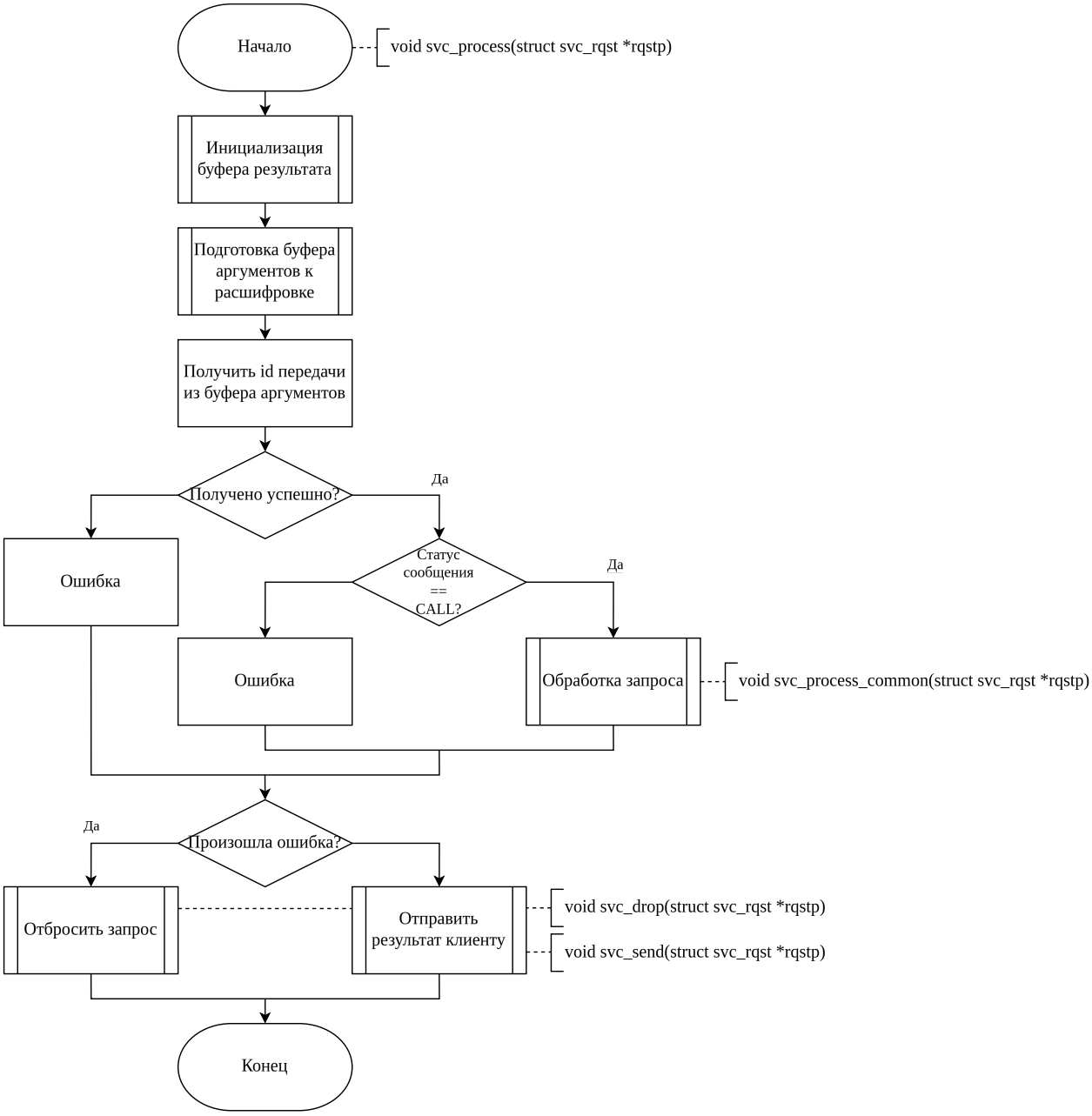

Рисунок 1.1 – Схема алгоритма функции `svc_process`

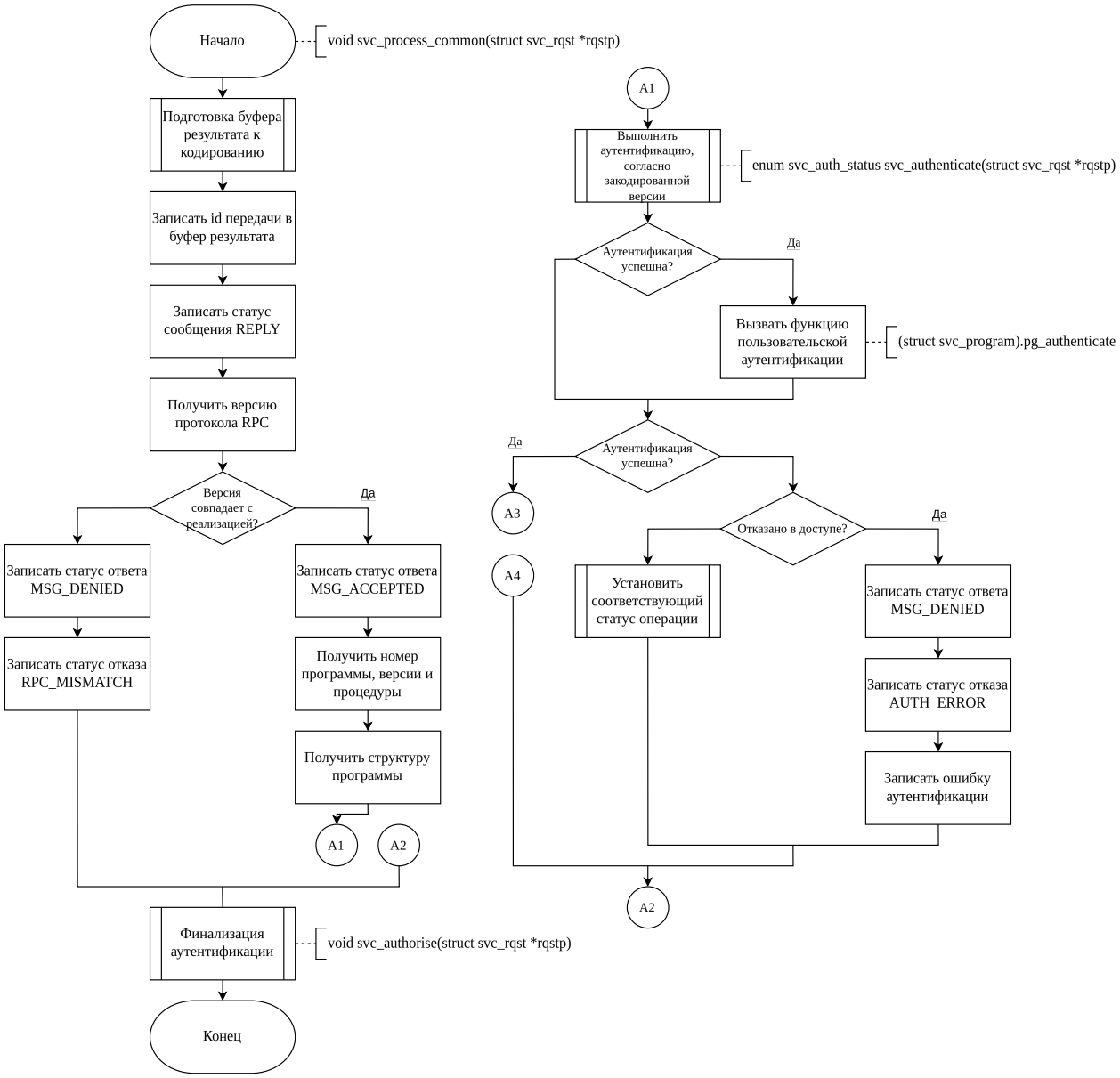

Рисунок 1.2 – Схема алгоритма функции `svc_process_common` (часть 1)

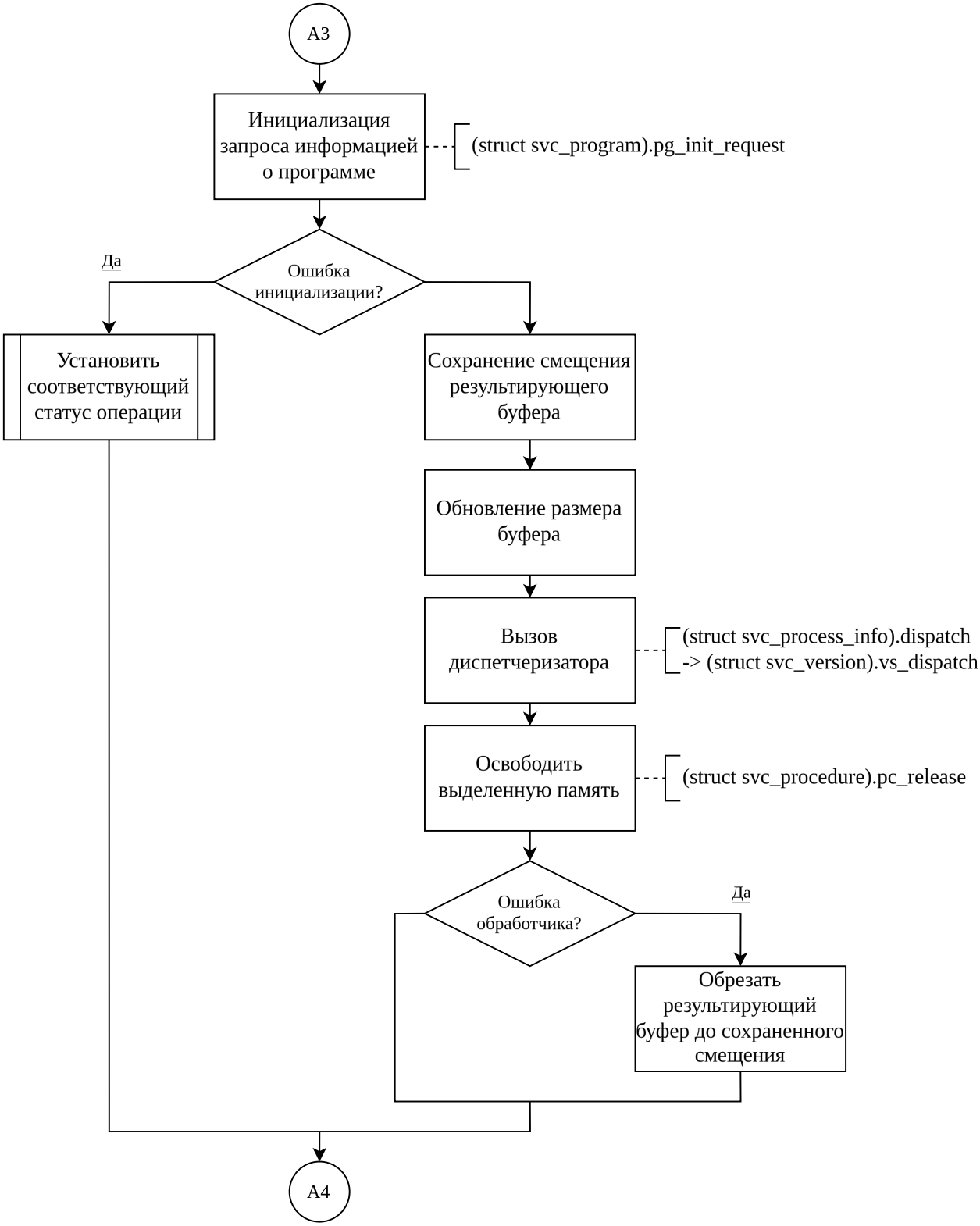

Рисунок 1.3 – Схема алгоритма функции `svc_process` (часть 2)

Шаблонная реализация метода программы pg_init_request представлена в функции svc_generic_init_request и выполняет следующие задачи:

— проверка существования запрашиваемой версии и процедуры;

— получение структуры svc_procedure (поле rq_procinfo);

— инициализация полей аргумента и результата (rq_argp и rq_resp);

— получение функции диспетчеризации (vs_dispatch).

Для функции диспетчеризации не предусмотрено готовое решение, однако из алгоритма обработки запроса видно, что остается преобразовать полученные аргументы (pc_decode), выполнить функцию обработчик запрашиваемой процедуры (pc_func) и закодировать результат (pc_encode). Минимальный пример такой функции представлен в листинге 1.12.

Листинг 1.12 – Минимальная реализация функции диспетчеризации

```
1  int dispatch(struct svc_rqst *rqstp)
2  {
3      const struct svc_procedure *proc = rqstp->rq_procinfo;
4      __be32 *statp = rqstp->rq_accept_statp;
5      if (!proc->pc_decode(rqstp, &rqstp->rq_arg_stream)) {
6          *statp = rpc_garbage_args;
7          return 0;
8      }
9      *statp = proc->pc_func(rqstp);
10     if (!proc->pc_encode(rqstp, &rqstp->rq_res_stream)) {
11         *statp = rpc_system_err;
12         return 0;
13     }
14     return 1;
15 }
```

1.3.2 Анализ порядка выполнения запроса

Аналогичные структуры для описания программы определены и для клиента (листинги 1.13–1.15).

Листинг 1.13 – Структура программы

```
struct rpc_program {
    /* Название программы */
    const char *                name;
    /* Номер программы */
    u32                         number;
    /* Массив версий и их количество */
    const struct rpc_version ** version;
    unsigned int                nrvers;
    struct rpc_stat *           stats;
    const char *                pipe_dir_name;
};
```

Листинг 1.14 – Структура версии

```
struct rpc_version {
    /* Номер версии */
    u32                        number;
    /* Массив процедур и их количество */
    const struct rpc_procinfo *procs;
    unsigned int               nrprocs;
    /* Количество вызовов */
    unsigned int              *counts;
};
```

Листинг 1.15 – Структура процедуры

```
struct rpc_procinfo {
    /* Номер процедуры */
    u32           p_proc;
    /* Кодирование аргументов */
    kxdreproc_t   p_encode;
    /* Декодирование результатов */
    kxdrdproc_t   p_decode;
    /* Размер */
    unsigned int p_arglen; /* аргумент */
    unsigned int p_replen; /* результат */
    /* Ожидаемое время задержки */
    unsigned int p_timer;
    u32           p_statidx;
    /* Название процедуры */
    const char  *p_name;
};
```

Для вызова процедуры необходимо вызвать одну из следующих функций оберток, представленных в листинге 1.16. Или же инициализировать и запустить задачу, описываемую структурой `rpc_task`, вручную (листинг 1.17).

Листинг 1.16 – Прототип функций вызова удаленных процедур

```
// Содержимое вызова
struct rpc_message {
    /* Информация о процедуре */
    const struct rpc_procinfo *rpc_proc;
    /* Аргументы */
    void *                      rpc_argp;
    /* Результат */
    void *                      rpc_resp;
    const struct cred *         rpc_cred;
};

// Функции обратного вызова
struct rpc_call_ops {
    void (*rpc_call_prepare)(struct rpc_task *, void *);
    void (*rpc_call_done)(struct rpc_task *, void *);
    void (*rpc_count_stats)(struct rpc_task *, void *);
    void (*rpc_release)(void *);
};

int     rpc_call_async(struct rpc_clnt *clnt,
                       const struct rpc_message *msg,
                       int flags,
                       const struct rpc_call_ops *tk_ops,
                       void *calldata);
int     rpc_call_sync(struct rpc_clnt *clnt,
                      const struct rpc_message *msg, int flags);
```

Листинг 1.17 – Структура `rpc_task`

```
struct rpc_task_setup {
    struct rpc_task           *task;
    struct rpc_clnt           *rpc_client;
    struct rpc_xprt           *rpc_xprt;
    struct rpc_cred           *rpc_op_cred;
    const struct rpc_message  *rpc_message;
    const struct rpc_call_ops *callback_ops;
    void                      *callback_data;
    struct workqueue_struct   *workqueue;
    unsigned short             flags;
    signed char                priority;
};

struct rpc_task *rpc_run_task(const struct rpc_task_setup *);
```

Вызов процедуры описан конечной машиной состояний. Возможные состояния задачи и переходы между ними представлены в таблицах 1.1 и 1.2.

Таблица 1.1 – Состояния задачи

Номер	Название	Назначение
1	call_start	Начальное состояние
2	call_reserve	Выделение слота транспортного протокола
3	call_reserveresult	Проверка выделенного слота
4	call_retry_reserve	Повторная попытка выделения слота
5	call_refresh	Обновление удостоверяющей информации
6	call_refreshresult	Проверка удостоверяющей информации
7	call_allocate	Выделение буфера
8	call_encode	Кодирование аргументов
9	call_bind	Получение порта программы
10	call_connect	Соединение с сервером
11	call_bind_status	Обработка релультатов rpcbind
12	call_connect_status	Обработка результата соединения
13	call_transmit	Отправка запроса
14	call_transmit_status	Проверка статуса передачи
15	call_status	Проверка статуса запроса
16	call_decode	Декодирование результата
17	rpc_exit_task	Фиктивное состояние завершения и разрушения запроса

Таблица 1.2 – Переходы состояний задачи

Инициатор	Цель	Причина (при наличии)
1	2	Иначе
	17	Клиент выключен
2	3	-
3	4	Не удалось выделить слот (EAGAIN | ENOMEM)
	5	Слот выделен
	17	Иная ошибка
4	3	-
5	6	-
6	5	ENOMEM
		Просроченное удостоверение
	7	Актуальное удостоверение
	17	Превышено число повторений
7	7	RPC_IS_ASYNC и не получен фатальный сигнал
	8	Буфер выделен
	17	Иная ошибка

Продолжение таблицы 1.2

8	9	Порт сервиса не получен
	10	Соединение не установлено
	13	Кодирование успешно и соединениеустановлено
	17	Не удалось закодировать заголовок
		Иная ошибка
9	10	Порт уже получен
	11	Иначе
	14	Запрос уже отправлен
10	12	Иначе
	13	Соединение уже установлено
	14	Запрос уже отправлен
11	9	Некритичная ошибка (ENOMEM | EACCESS | ENOBUFS | EAGAIN | ETIMEDOUT)
		Сетевая ошибка (если сброшен флаг SOFTCONN)
	10	Порт получен
	14	Запрос уже отправлен
	17	Иная ошибка
12	9	Некритичная ошибка (ENOBUFS | EAGAIN| ETIMEDOUT)
		Сетевая ошибка (если сброшен флаг SOFTCONN)
	13	Соединение установлено
	14	Запрос уже отправлен
	17	Иная ошибка
13	14	-
14	8	EBADMSG
	9	ENOMEM | ENOBUFS | EBADSLT | EAGAIN
	13	Сетевая ошибка (если сброшен флаг SOFTCONN)
	15	Запрос передан
	17	Иная ошибка
15	8	Сетевая ошибка (если сброшен флаг SOFTCONN)
		EAGAIN | ENFILE | ENOBUFS | ENOMEM
	16	Запрос выполнен успешно
	17	Иная ошибка

Продолжение таблицы 1.2

16	2	EKEYREJECTED
	8	EAGAIN
	17	Результат декодирован успешно
		Иная ошибка

Вне зависимости от выбранного подхода предварительно необходимо создать объект клиента — структуру `rpc_clnt`, вызовом функции `rpc_create` (листинг 1.18).

Листинг 1.18 – Создание клиента

```
struct rpc_create_args {
    struct net               *net;
    int                       protocol;
    struct sockaddr          *address;
    size_t                    addrsize;
    const struct rpc_program *program;
    u32                       version;
    rpc_authflavor_t          authflavor;
    const struct cred        *cred;
    unsigned long             flags;

    struct sockaddr          *saddress;
    const struct rpc_timeout *timeout;
    const char               *servername;
    const char               *nodename;
    struct rpc_stat          *stats;
    u32                       prognumber;
    u32                       nconnect;
    char                     *client_name;
    struct svc_xprt          *bc_xprt;
    unsigned int              max_connect;
    struct xprtsec_parms      xprtsec;
    unsigned long             connect_timeout;
    unsigned long             reconnect_timeout;
};

struct rpc_clnt *rpc_create(struct rpc_create_args *args);
```

Поведение клиента может быть модифицировано с использованием следующих флагов:

— `RPC_CLNT_CREATE_HARDRTRY` — моментальное повторение запроса при возникновении ошибки;

— `RPC_CLNT_CREATE_AUTOBIND` — порт сервиса получается перед каж-

дым повтором запроса;

— RPC_CLNT_CREATE_NONPRIVPORT — не использовать зарезервированные порты (выбирается случайно);

— RPC_CLNT_CREATE_NOPING — не проверять соединение перед созданием;

— RPC_CLNT_CREATE_DISCRTRY — закрывать соединение перед повтором запроса;

— RPC_CLNT_CREATE_QUIET — не выводить отладочные сообщения;

— RPC_CLNT_CREATE_INFINITE_SLOTS — использовать максимальное количество слотов;

— RPC_CLNT_CREATE_NO_IDLE_TIMEOUT — не позволять простаивающие соединения;

— RPC_CLNT_CREATE_NO_RETRANS_TIMEOUT — бесконечное ожидание ответа;

— RPC_CLNT_CREATE_SOFTERR — таймаут интерпретируется как ошибка;

— RPC_CLNT_CREATE_REUSEPORT — использовать тот же порт при повторе запроса;

— RPC_CLNT_CREATE_CONNECTED — установить соединение при создании клиента.

1.3.3 Анализ преобразования аргументов и результата

Для кодирования и декодирования сообщения определена структура и операции над ней для поточного преобразования буфера (листинги 1.19–1.20).

Листинг 1.19 – Структура буфера (часть 1)

```
1 /* Структура буфера для принятия и передачи сообщения */
2 struct xdr_buf {
3     struct kvec     head[1],    /* RPC header + non-page data */
4                     tail[1];    /* Appended after page data */
5     struct bio_vec *bvec;
```

Листинг 1.20 – Структура буфера (часть 2)

```
    struct page ** pages;     /* Array of pages */
    unsigned int   page_base, /* Start of page data */
                   page_len,  /* Length of page data */
                   flags;     /* Flags for data disposition */
#define XDRBUF_READ           0x01 /* target of file read */
#define XDRBUF_WRITE          0x02 /* source of file write */
#define XDRBUF_SPARSE_PAGES   0x04 /* Page array is sparse */
    unsigned int   buflen,    /* Total length of storage buffer */
                   len;       /* Length of XDR encoded message */
};

/* Обертка для потоковой обработки */
struct xdr_stream {
    __be32 *p;                /* start of available buffer */
    struct xdr_buf *buf;      /* XDR buffer to read/write */
    __be32 *end;              /* end of available buffer space */
    struct kvec *iov;         /* pointer to the current kvec */
    struct kvec scratch;      /* Scratch buffer */
    struct page **page_ptr;   /* pointer to the current page */
    void *page_kaddr;         /* kmapped address of the current page */
    unsigned int nwords;      /* Remaining decode buffer length */
    struct rpc_rqst *rqst;    /* For debugging */
};
```

Для записи данных необходимо получить текущее положение указателя внутри буфера при помощи функции `xdr_reserve_space`. Функция отслеживает смещение и проверяет достаточно ли места для размещения `nbytes` байт информации.

Далее разработчик может работать с полученным указателем по своему усмотрению, при этом для типов описанных в спецификации xdr, существуют готовые функции для кодирования (листинги 1.21–1.22) и их обертки, позволяющие опустить вызов функции `xdr_reserve_space`.

Листинг 1.21 – Функции для кодирования (часть 1)

```
__be32 * xdr_reserve_space(struct xdr_stream *xdr, size_t nbytes);

__be32 *xdr_encode_hyper(__be32 *p, __u64 val);
__be32 *xdr_encode_bool(__be32 *p, u32 n);
int xdr_encode_array2(const struct xdr_buf *buf, unsigned int base,
                      struct xdr_array2_desc *desc);
__be32 *xdr_encode_opaque_fixed(__be32 *p, const void *ptr,
                                unsigned int len);
__be32 *xdr_encode_opaque(__be32 *p, const void *ptr,
                          unsigned int len);
```

Листинг 1.22 – Функции для кодирования (часть 2)

```
__be32 *xdr_encode_string(__be32 *p, const char *s);
__be32 *xdr_encode_array(__be32 *p, const void *s, unsigned int len);

int xdr_stream_encode_bool(struct xdr_stream *xdr, __u32 n);
ssize_t xdr_stream_encode_u32(struct xdr_stream *xdr, __u32 n);
ssize_t xdr_stream_encode_be32(struct xdr_stream *xdr, __be32 n);
ssize_t xdr_stream_encode_be32(struct xdr_stream *xdr, __be32 n);
ssize_t xdr_stream_encode_opaque_inline(struct xdr_stream *xdr,
                                        void **ptr, size_t len);
ssize_t xdr_stream_encode_opaque_fixed(struct xdr_stream *xdr,
                                       const void *ptr, size_t len);
ssize_t xdr_stream_encode_uint32_array(struct xdr_stream *xdr,
                                       const __u32 *array,
                                       size_t array_size);
```

Декодирование построено по той же модели — разработчик запрашивает буфер определенного размера и считывает из него данные, аналогично библиотека предоставляет функции обертки для базовых типов данных (листинги 1.23–1.24).

Листинг 1.23 – Функции для декодирования (часть 1)

```
__be32 *xdr_inline_decode(struct xdr_stream *xdr, size_t nbytes);

__be32 *xdr_decode_hyper(__be32 *p, __u64 *valp);
__be32 *xdr_decode_opaque_fixed(__be32 *p, void *ptr,
                                unsigned int len);
__be32 *xdr_decode_string_inplace(__be32 *p, char **sp,
                                  unsigned int *lenp,
                                  unsigned int maxlen);
int xdr_decode_array2(const struct xdr_buf *buf, unsigned int base,
                      struct xdr_array2_desc *desc);

ssize_t xdr_stream_decode_bool(struct xdr_stream *xdr, __u32 *ptr);
ssize_t xdr_stream_decode_u32(struct xdr_stream *xdr, __u32 *ptr);
ssize_t xdr_stream_decode_u64(struct xdr_stream *xdr, __u64 *ptr);
ssize_t xdr_stream_decode_opaque_fixed(struct xdr_stream *xdr,
                                       void *ptr, size_t len);
ssize_t xdr_stream_decode_opaque_inline(struct xdr_stream *xdr,
                                        void **ptr, size_t maxlen);
ssize_t xdr_stream_decode_uint32_array(struct xdr_stream *xdr,
                                       __u32 *array,
                                       size_t array_size)
ssize_t xdr_stream_decode_opaque(struct xdr_stream *xdr, void *ptr,
                                 size_t size);
```

Листинг 1.24 – Функции для декодирования (часть 2)

```
ssize_t xdr_stream_decode_opaque_dup(struct xdr_stream *xdr, void **p,
                                     size_t maxlen, gfp_t gfp_flags);
ssize_t xdr_stream_decode_string(struct xdr_stream *xdr, char *str,
                                 size_t size);
ssize_t xdr_stream_decode_string_dup(struct xdr_stream *xdr, char **s,
                                     size_t maxlen, gfp_t gfp_flags);
```

Выводы

В результате анализа протокола ONC RPC и его реализации в ядре операционной системы Linux, было установлено, что для генерации кода модулей ядра, взаимодействующих по модели клиент-сервер, необходимо:

— определить соответствующие структуры программ, версий и процедур для клиента (`rpc_program`, `rpc_version`, `rpc_procedure`) и сервера (`svc_program`, `svc_version`, `svc_procedure`). Описание сервиса состоит из инициализации структуры программы, которая должна содержать функции аутентификации, инициализации контекста потока и регистрации у `portmapper`, структуры версии, содержащей максимальный размер буфера XDR и функцию диспетчеризации, а также процедур, предоставляющих соответствующий обработчик и функции для кодирования результата и декодирования аргументов;

— запустить сервер, путем последовательного вызова функций:

1) `svc_create` для создания структуры сервиса;

2) `svc_bind` для удаления записи об используемых номерах программ, версий и процедур у `portmapper` в случае повторного появления;

3) `svc_xprt_create` для связи сервиса с транспортным протоколом и регистрации у `portmapper`;

4) `svc_set_num_threads` для запуска сервиса;

— произвести вызов, при помощи следующих функций:

1) `rpc_cred` для создания структуры клиента;

2) `rpc_call_sync` и `rpc_call_async` для вызова удаленной процедуры.

2 Конструкторская часть

2.1 Требования к программе

К разрабатываемому приложению выдвигаются следующие требования:

— поддержка и разбор спецификации на языке rpcl;

— возможность раздельной загрузки и использования модулей разных программ;

— возможность конфигурирования порта и адреса модуля сервера;

— возможность использования предоставляемых процедур из сторонних модулей.

2.2 Схема алгоритма программы

Алгоритм работы приложения может быть разбит на следующие этапы, представленные на рисунке 2.1.

Рисунок 2.1 – Декомпозиция алгоритма работы программы

2.2.1 Преобразование текста спецификации

Для обеспечения единообразия разбора определений описанного сервиса код спецификации необходимо преобразовать в набор заранее описанных лексических единиц — токенов. Согласно структуре языка, описанной в листингах 1.2 и 1.1, можно выделить следующие группы токенов:

— идентификатор;

— ключевое слово (`"const"`, `"case"`, `"switch"`, `"default"`, `"typedef"`, `"program"`, `"version"`, `"procedure"`);

— скобки (круглые `"()"`, фигурные `"{}"`, квадратные `"[]"`, треугольные `"<>"`);

— идентификатор типа (`"void"`, `"unsigned"`, `"integer"`, `"hyper"`, `"float"`, `"double"`, `"boolean"`, `"quadruple"`, `"string"`, `"opaque"`, `"pointer"`, `"enum"`, `"struct"`, `"union"`);

— разделитель (точка с запятой `";"`, двоеточие `":"`, запятая `","`);

— оператор присвоения `"="`;

— целочисленный литерал;

— комментарий.

На рисунках 2.2–2.3 представлен алгоритм преобразования текста спецификации в набор токенов. Подразумевается, что алгоритму доступен массив набора правил для сопоставления текущего символа. Принцип работы такого набора правил сводится к последовательному сравнению поставляемых символов с заранее заданным массивом или проверке его принадлежности определенному подмножеству символов алфавита. В качестве пропускаемых символов выступают все пробельные символы.

Был определен следующий приоритет групп токенов: разделители, скобки, литералы, ключевые слова, идентификаторы типа, операторы, идентификаторы, комментарии.

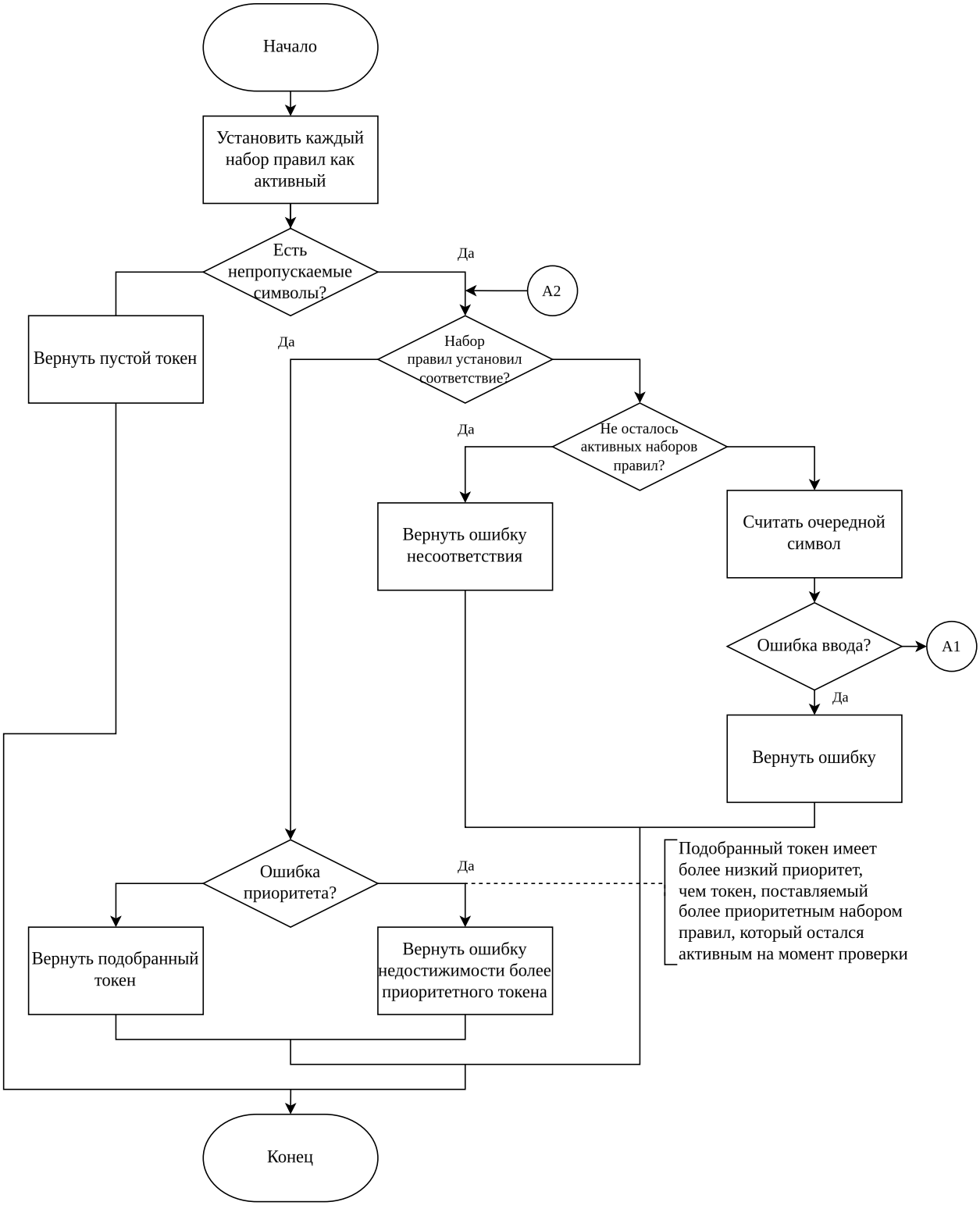

Рисунок 2.2 – Алгоритм лексического анализа текста спецификации (часть 1)

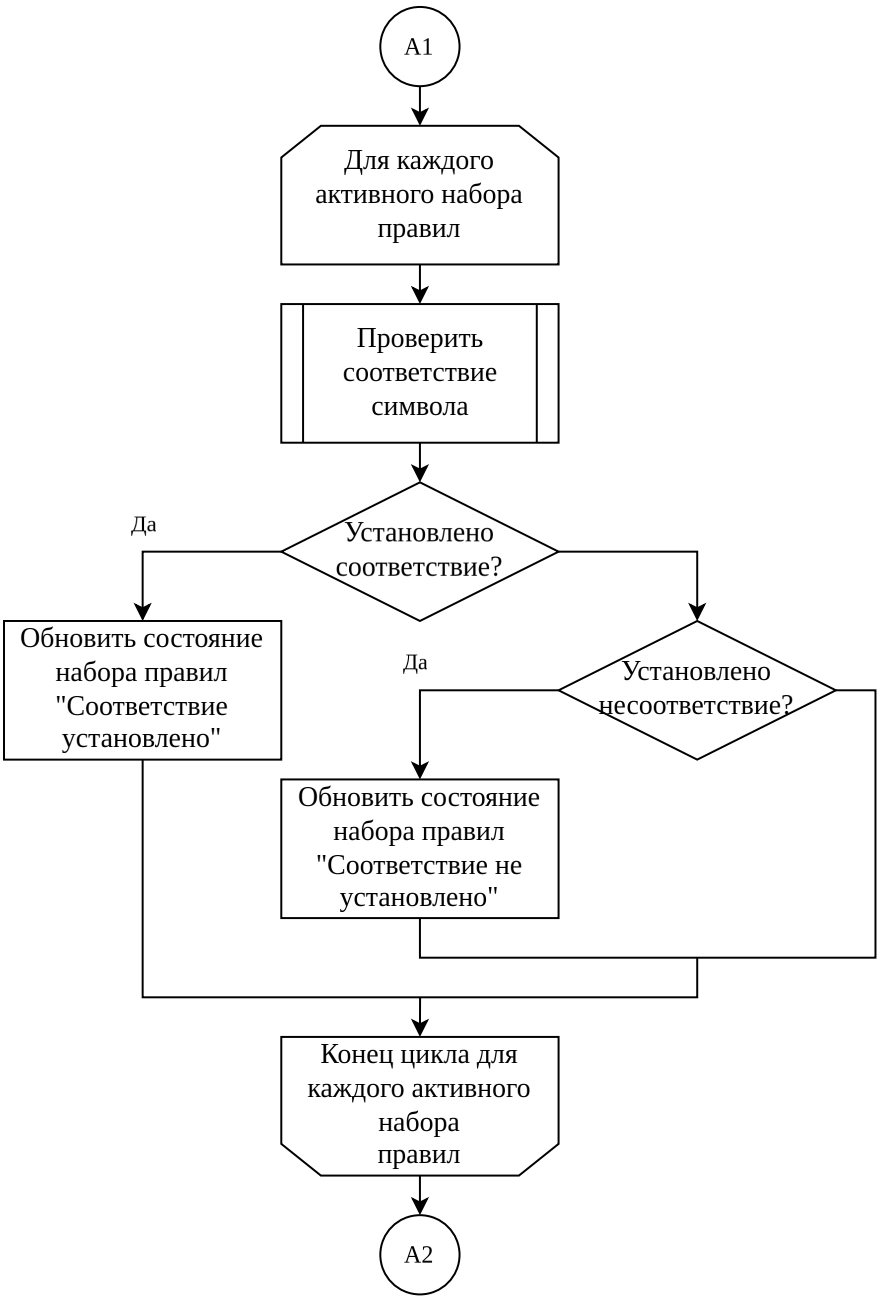

Рисунок 2.3 – Алгоритм лексического анализа текста спецификации (часть 2)

2.2.2 Разбор определений

В соответствии с листингами 1.1 и 1.2, спецификация может состоять из 3 видов определений:

1) определение константы;
2) определение типа;
3) определение программы.

Алгоритм преобразования потока токенов в поток определений представлен на рисунках 2.4–2.16

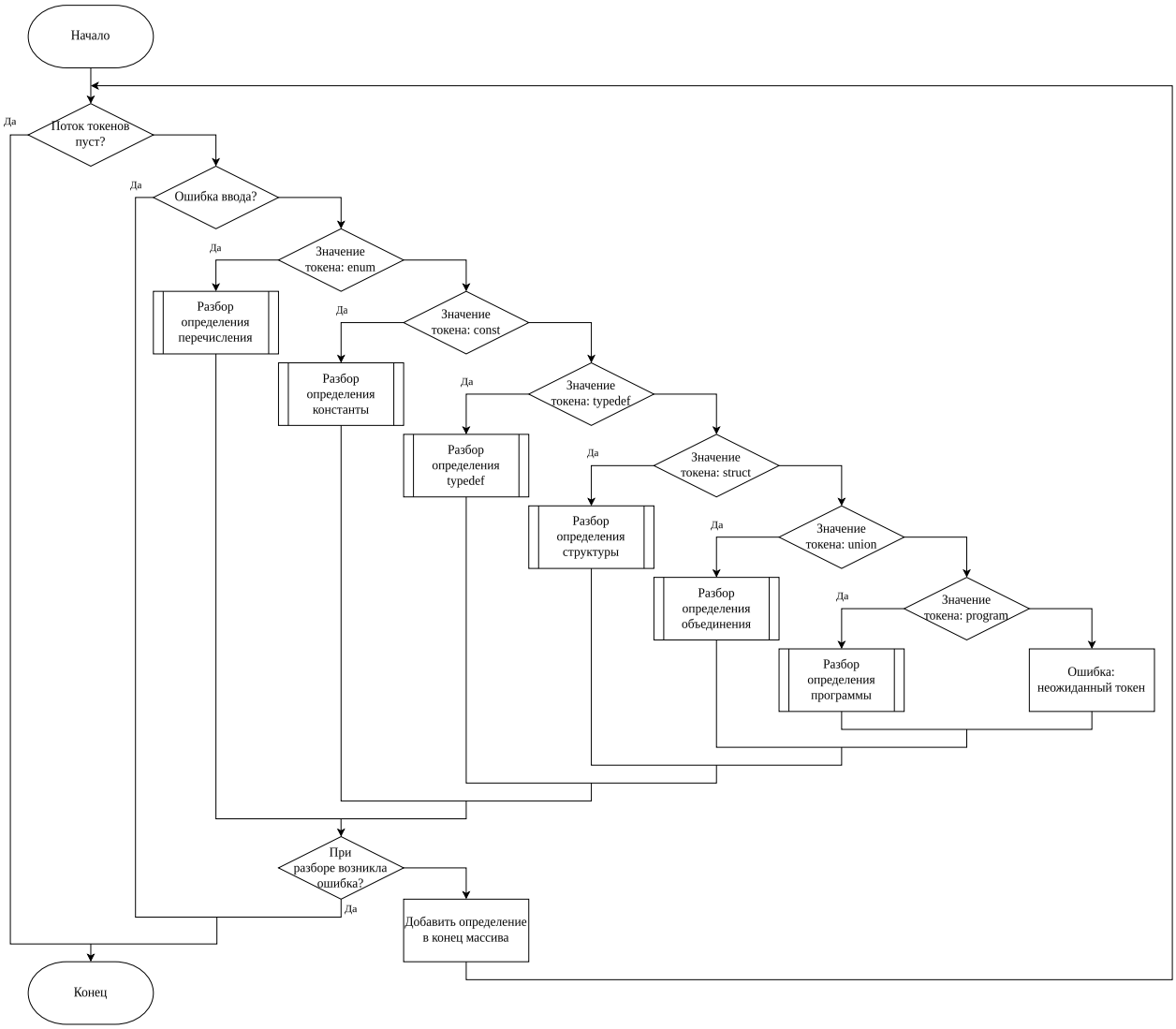

Рисунок 2.4 – Алгоритм разбора токенов

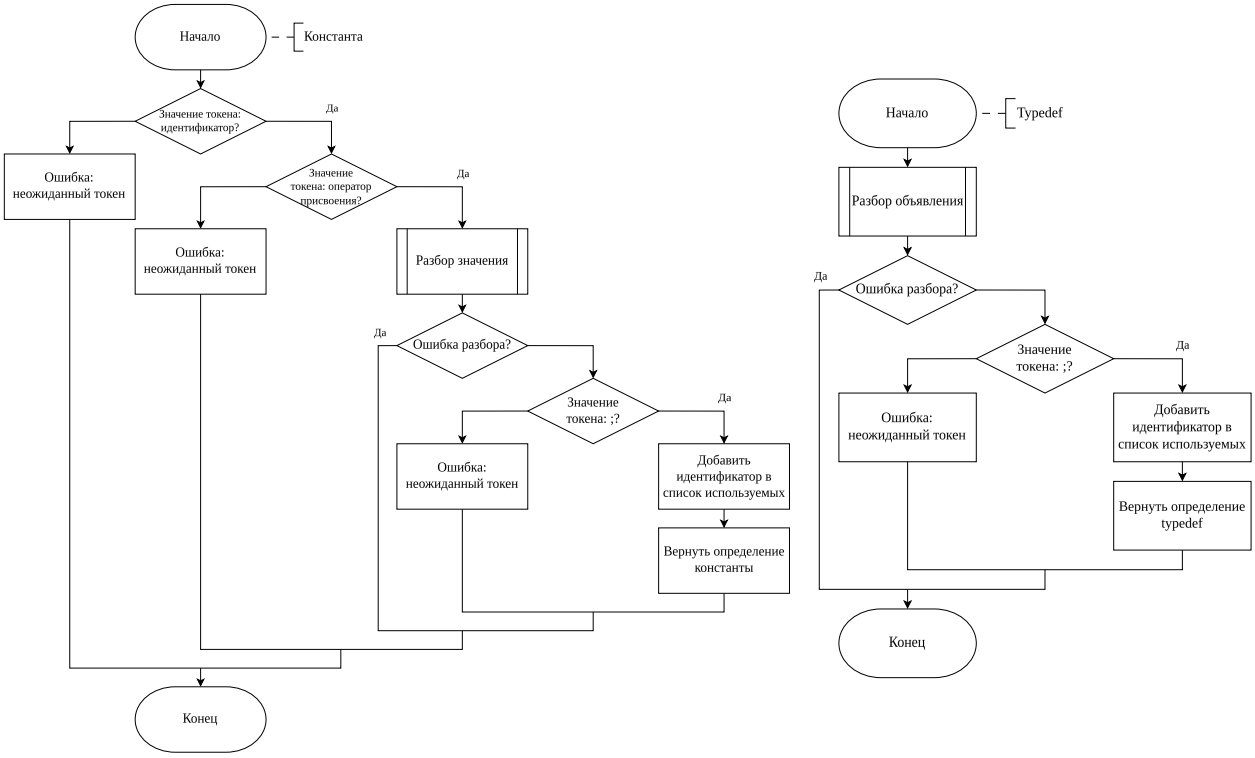

Рисунок 2.5 – Алгоритм разбора константы и `typedef`

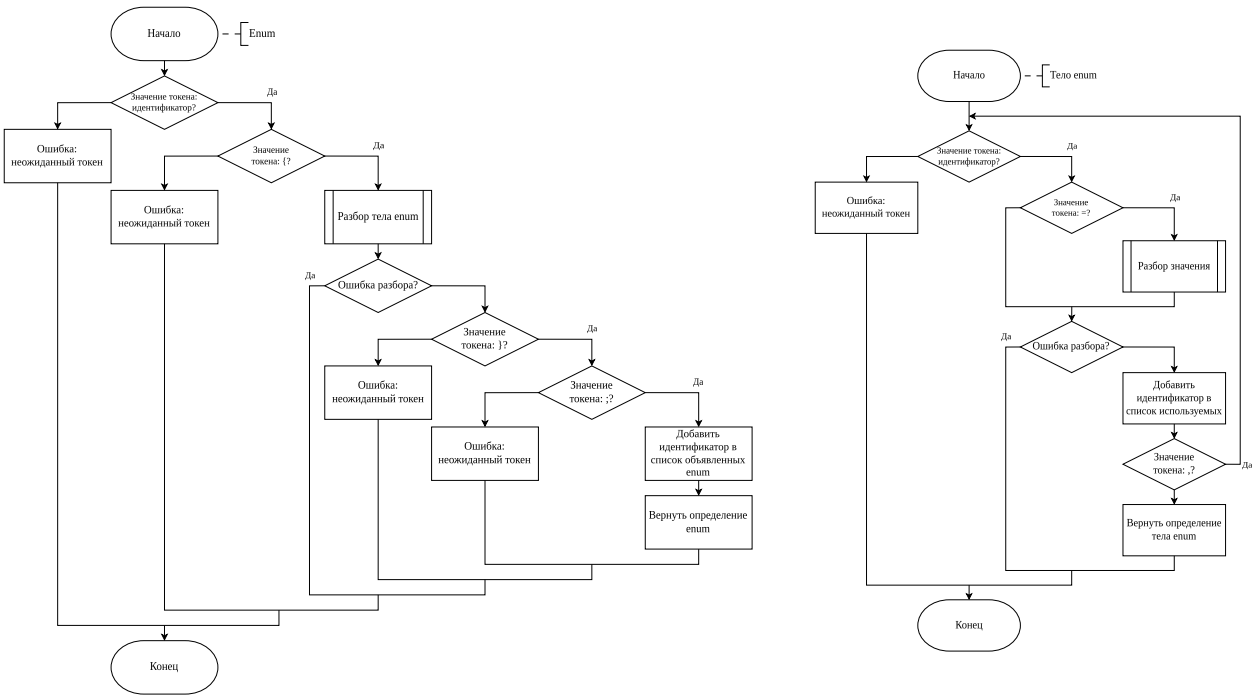

Рисунок 2.6 – Алгоритм разбора `enum`

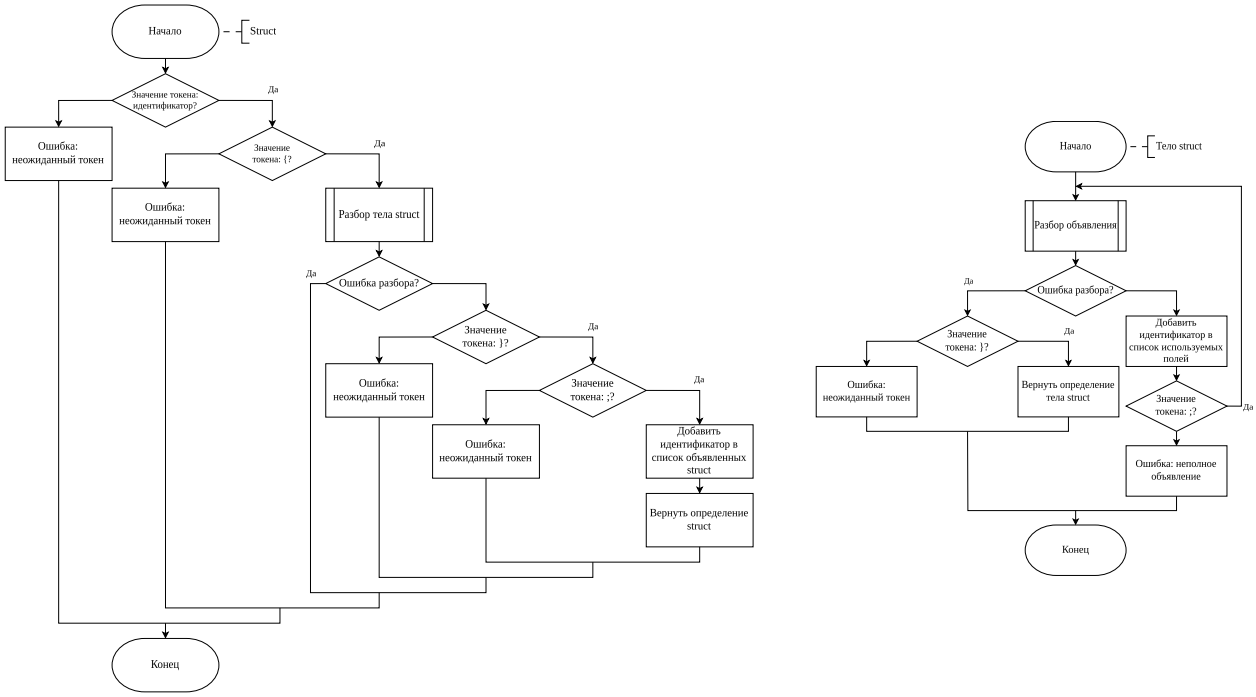

Рисунок 2.7 – Алгоритм разбора `struct`

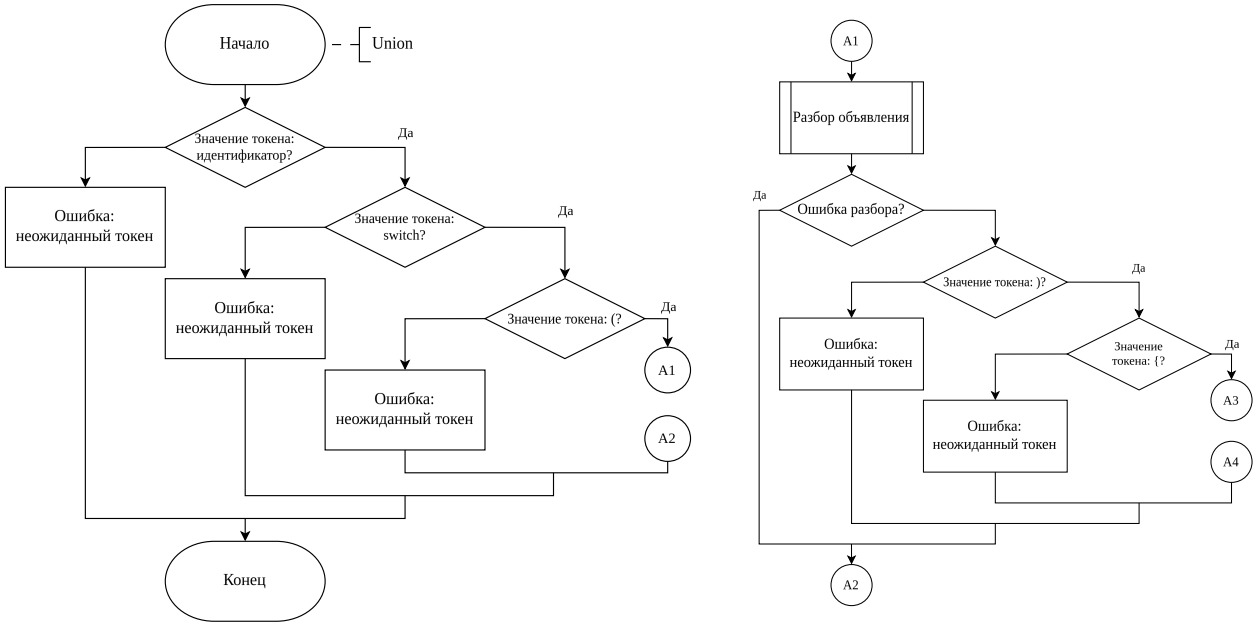

Рисунок 2.8 – Алгоритм разбора `union` (часть 1)

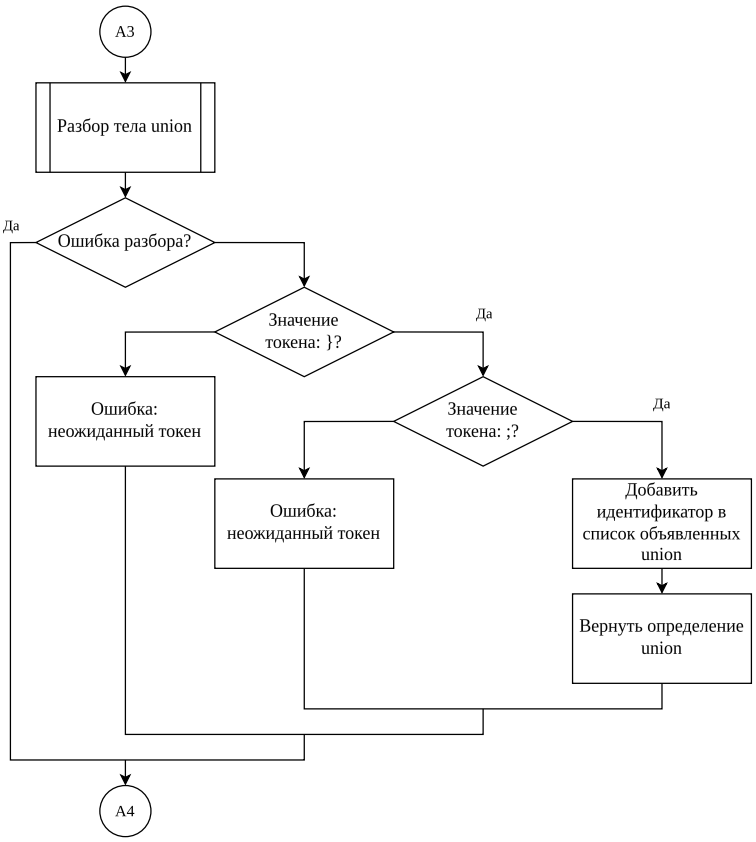

Рисунок 2.9 – Алгоритм разбора `union` (часть 2)

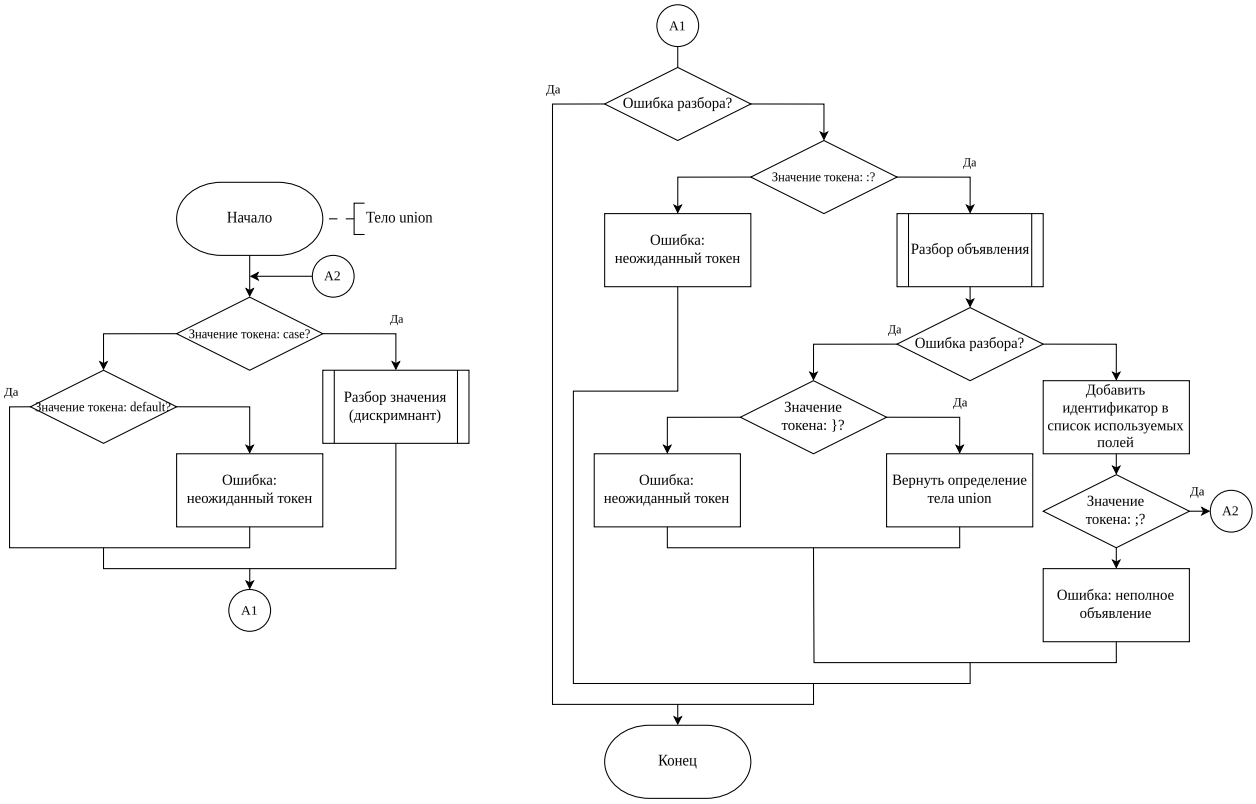

Рисунок 2.10 – Алгоритм разбора тела `union`

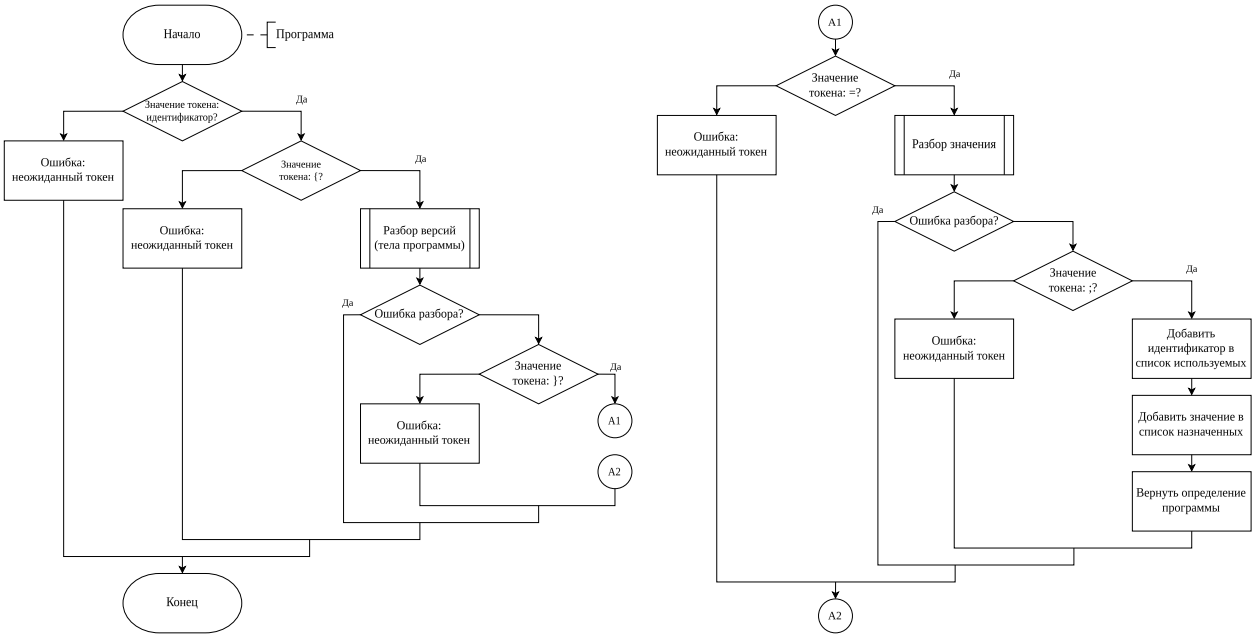

Рисунок 2.11 – Алгоритм разбора программы

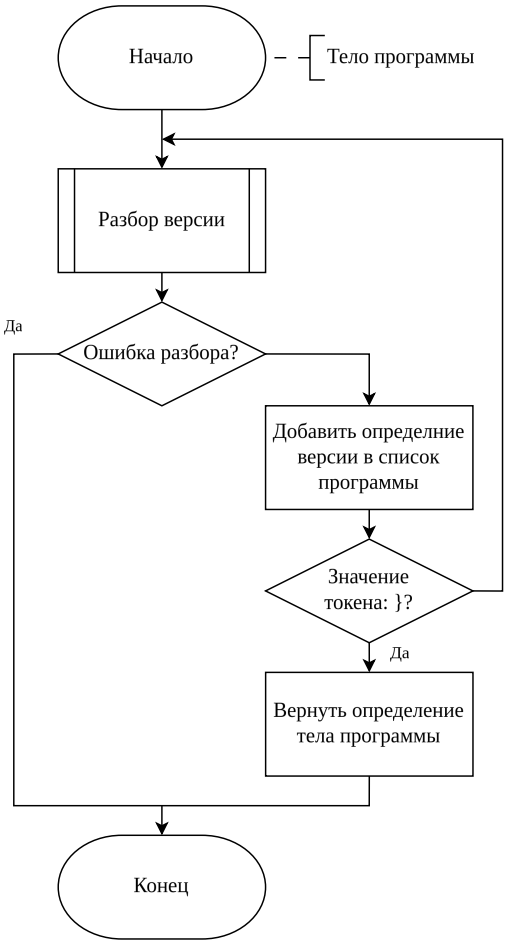

Рисунок 2.12 – Алгоритм разбора тела программы

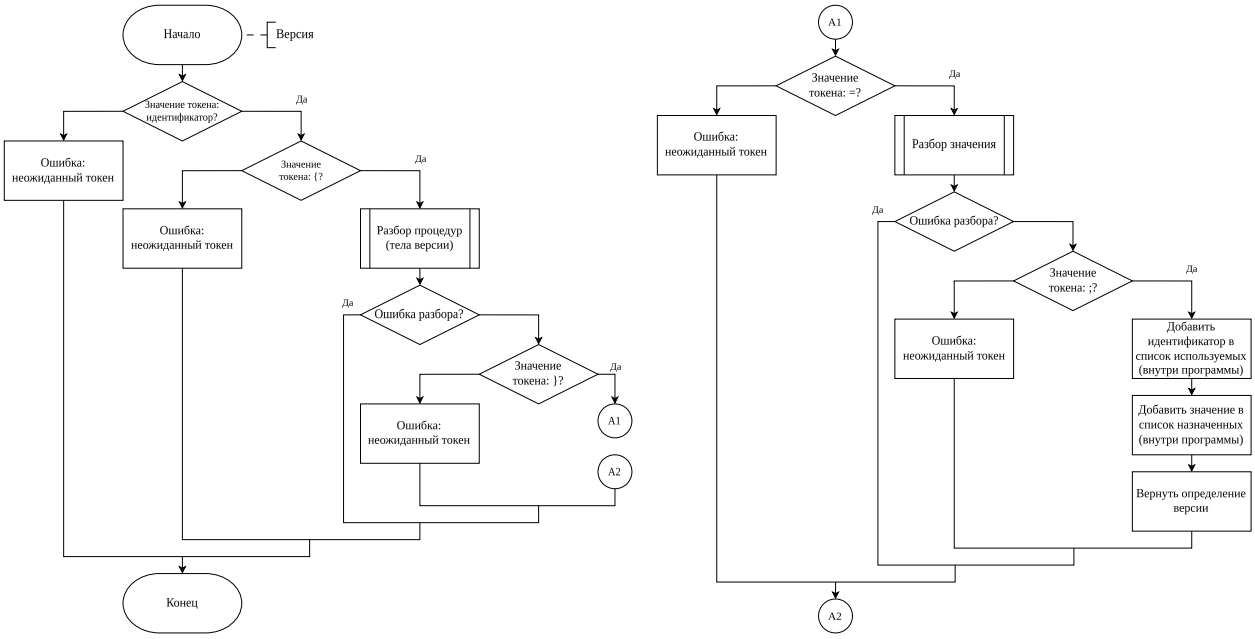

Рисунок 2.13 – Алгоритм разбора версии

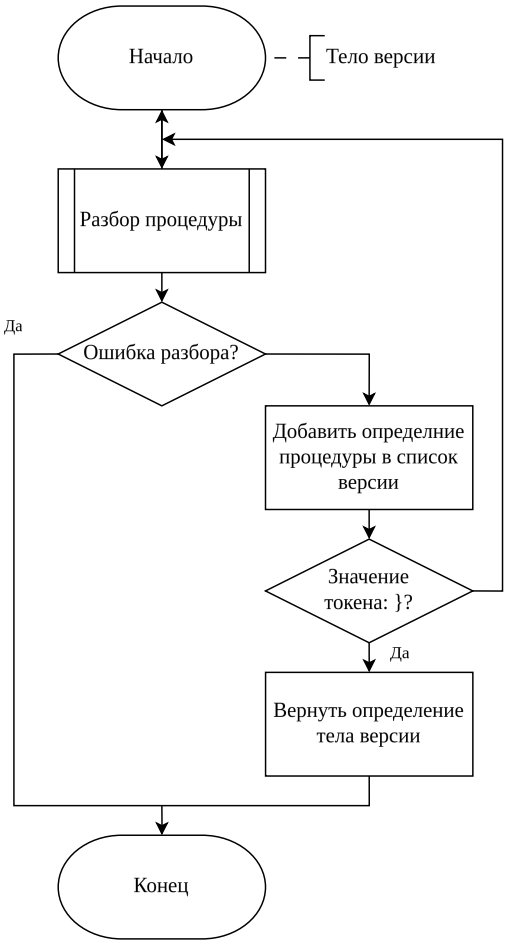

Рисунок 2.14 – Алгоритм разбора тела версии

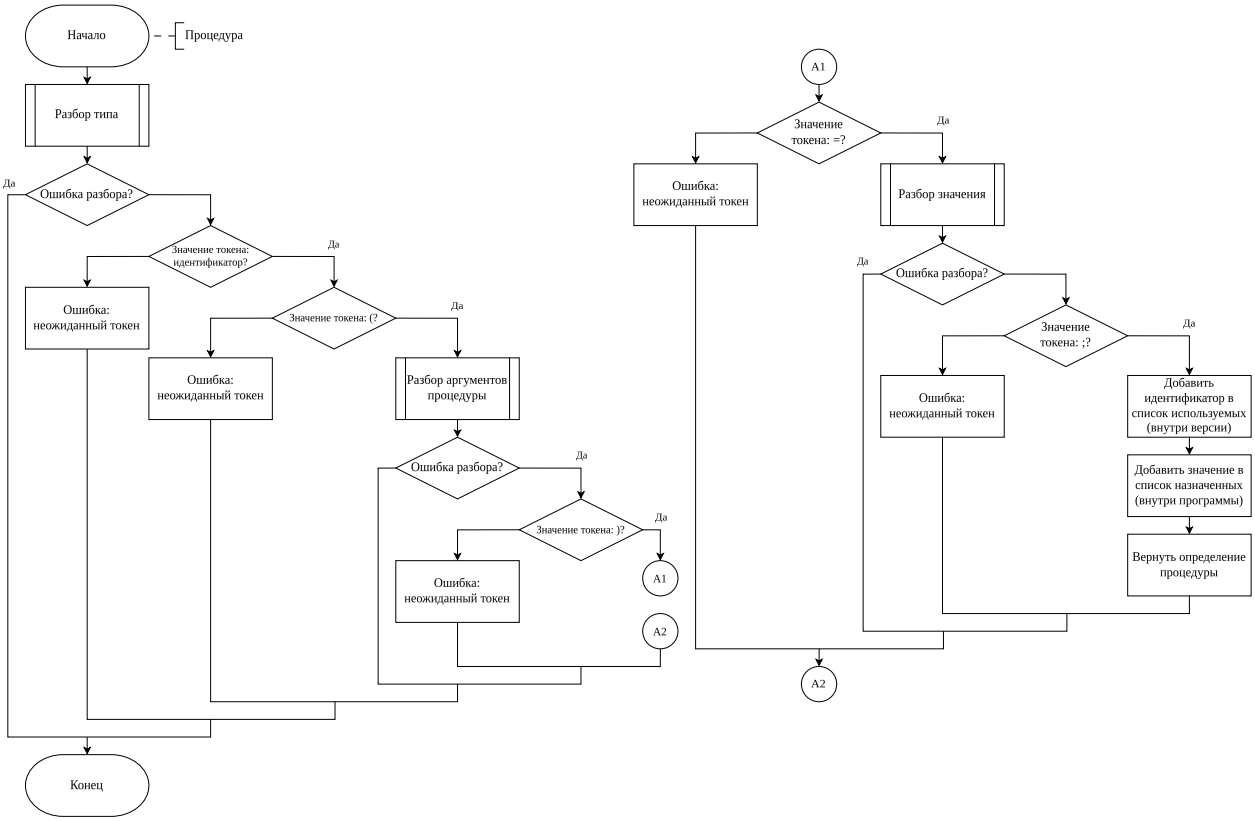

Рисунок 2.15 – Алгоритм разбора процедуры

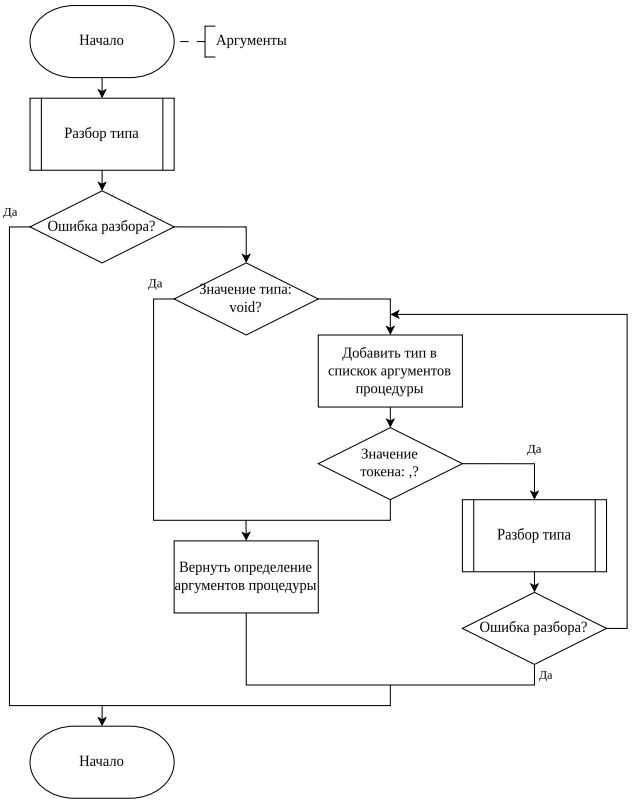

Рисунок 2.16 – Алгоритм разбора тела процедуры

2.2.3 Генерация кода модулей

Алгоритм генерации модулей зависит от макета, который будет разработан в разделе 3.2, и потому не будет приведен в качестве схемы. Общая идея заключается в создании отдельного модуля для каждой программы. Между собой модули будут разделять определения констант и типов.

3 Технологическая часть

3.1 Выбор языка и среды программирования

В качестве языка реализации приложения был выбран rust из-за развитой системы типов, позволяющей описывать и обрабатывать потоки ввода, токенов и определений преимущественно декларативно. Для модулей ядра был выбран язык C, так как это язык реализации операционной системы Linux. Использовался пакетный менеджер cargo и компилятор rustc.

3.2 Макет разрабатываемых модулей ядра

На листингах 3.1–3.8 представлен код макетов модулей ядра клиента и сервера.

Листинг 3.1 – Прототип сервера (часть 1)

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/bitops.h>
#include <linux/kthread.h>
#include <linux/freezer.h>
#include <linux/sunrpc/stats.h>
#include <linux/sunrpc/svcsock.h>
#include <linux/sunrpc/svc_xprt.h>
#include <linux/sunrpc/svc.h>
#include <linux/sunrpc/xdr.h>

MODULE_LICENSE("GPL");

static bool u64_arg(struct svc_rqst *rqstp, struct xdr_stream *xdr) {
    u64 *arg = rqstp->rq_argp;
    if (0 != xdr_stream_decode_u64(xdr, arg)) {
        return false;
    }
    return true;
}
```

Листинг 3.2 – Прототип сервера (часть 2)

```
static bool u64_res(struct svc_rqst *rqstp, struct xdr_stream *xdr)
{
    u64 *res = rqstp->rq_resp;
    if (0 > xdr_stream_encode_u64(xdr, *res)) {
        return false;
    }
    return true;
}

static __be32 increment(struct svc_rqst *request)
{
    u64 *in = request->rq_argp;
    u64 *out = request->rq_resp;
    *out = *in + 1;
    return rpc_success;
}

static enum svc_auth_status access_any(struct svc_rqst *rqstp)
{
    return SVC_OK;
}

static unsigned long v_count = 0;
static struct svc_stat stat;

static const struct svc_procedure procedures[] = {
    [1] = {
        .pc_func = increment,
        .pc_decode = u64_arg,
        .pc_encode = u64_res,
        .pc_argsize = sizeof(u64),
        .pc_argzero = sizeof(u64),
        .pc_ressize = sizeof(u64),
        .pc_xdrressize = sizeof(u64),
        .pc_name = "inc",
    },
};

static int dispatch(struct svc_rqst *rqstp)
{
    const struct svc_procedure *proc = rqstp->rq_procinfo;
    __be32 *statp = rqstp->rq_accept_statp;
    if (!proc->pc_decode(rqstp, &rqstp->rq_arg_stream))
    {
        *statp = rpc_garbage_args;
        return 0;
    }
```

Листинг 3.3 – Прототип сервера (часть 3)

```
    *statp = proc->pc_func(rqstp);
    if (test_bit(RQ_DROPME, &rqstp->rq_flags)) {
        *statp = rpc_success;
        return 0;
    }
    if (!proc->pc_encode(rqstp, &rqstp->rq_res_stream)) {
        *statp = rpc_system_err;
        return 0;
    }
    return 1;
}

static const struct svc_version version = {
    .vs_vers = 1,
    .vs_nproc = ARRAY_SIZE(procedures),
    .vs_proc = procedures,
    .vs_count = &v_count,
    .vs_dispatch = dispatch,
    .vs_xdrsize = sizeof(u64),
    .vs_hidden = false,
    .vs_rpcb_optnl = false,
    .vs_need_cong_ctrl = false
};

static const struct svc_version *versions[] = {
    [1] = &version,
};

static struct svc_program program = {
    .pg_prog = 1,
    .pg_lovers = 1,
    .pg_hivers = 1,
    .pg_nvers = ARRAY_SIZE(versions),
    .pg_vers = versions,
    .pg_name = "test",
    .pg_class = "test",
    .pg_authenticate = access_any,
    .pg_init_request = svc_generic_init_request,
    .pg_rpcbind_set  = svc_generic_rpcbind_set
};

struct svc_xprt *xprt;
struct svc_serv *server = NULL;
```

Листинг 3.4 – Прототип сервера (часть 4)

```
static int threadfn(void *data)
{
    struct svc_rqst *rqstp = data;
    svc_thread_init_status(rqstp, 0);
    set_freezable();
    while (!svc_thread_should_stop(rqstp)) {
        svc_recv(rqstp);
    }
    svc_exit_thread(rqstp);
    return 0;
}

static int __init init_md(void) {
    stat.program = &program;
    server = svc_create(&program, 0, threadfn);
    if (NULL == server) {
        printk("+ Error during svc_create\n");
        return 1;
    }
    int rc = svc_bind(server, &init_net);
    if (0 == rc && NULL == xprt) {
        rc = svc_xprt_create(server, "tcp", &init_net, AF_INET,
                             0, 0, get_current_cred());
        rc = rc < 0 ? rc : 0;
    }
    if (0 == rc) {
        rc = svc_set_num_threads(server, NULL, 1);
    }
    if (0 == rc) {
        printk("+ rpc module start\n");
    } else {
        printk("+ Error: %pe\n", ERR_PTR(rc));
    }
    return rc;
}

static void __exit exit_md(void) {
    if (server) {
        svc_xprt_destroy_all(server, &init_net);
        svc_set_num_threads(server, NULL, 0);
        svc_rpcb_cleanup(server, &init_net);
        svc_destroy(&server);
    }
    printk("+ rpc module end end\n");
}
module_init(init_md);
module_exit(exit_md);
```

Листинг 3.5 – Прототип клиента (часть 1)

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/bitops.h>
#include <linux/kthread.h>
#include <linux/freezer.h>
#include <linux/sunrpc/stats.h>
#include <linux/sunrpc/clnt.h>
#include <linux/sunrpc/xprt.h>
#include <linux/sunrpc/xdr.h>

MODULE_LICENSE("GPL");

static void voidarg(
    struct rpc_rqst *rqstp,
    struct xdr_stream *xdr,
    const void *data
) { }

static int voidres(
    struct rpc_rqst *rqstp,
    struct xdr_stream *xdr,
    void *data
)
{
    return 0;
}

static void u64_arg(
    struct rpc_rqst *rqstp,
    struct xdr_stream *xdr,
    const void *data
)
{
    const u64 *arg = data;
    xdr_stream_encode_u64(xdr, *arg);
}

static int u64_res(
    struct rpc_rqst *rqstp,
    struct xdr_stream *xdr,
    void *data
)
{
    int rc = xdr_stream_decode_u64(xdr, data);
    return rc;
}
```

Листинг 3.6 – Прототип клиента (часть 2)

```
struct rpc_procinfo progrs[] = {
    [1] = {
        .p_proc = 1,
        .p_encode = u64_arg,
        .p_decode = u64_res,
        .p_arglen = sizeof(u64),
        .p_replen = sizeof(u64),
        .p_statidx = 1,
        .p_name = "increment",
    },
};

unsigned int count = 0;

struct rpc_version version = {
    .number = 1,
    .nrprocs = ARRAY_SIZE(progrs),
    .procs = progrs,
    .counts = &count,
};

const struct rpc_version *versions[] = {
    [1] = &version,
};

struct rpc_stat stat = {};

struct rpc_program program = {
    .name = "test",
    .number = 1,
    .nrvers = ARRAY_SIZE(versions),
    .version = versions,
    .stats = &stat,
};

struct rpc_clnt *client = NULL;

static struct rpc_clnt *get_client(void) {
    if (NULL != client) {
        return client;
    }
    struct sockaddr_in sin = {
        .sin_family      = AF_INET,
        .sin_addr.s_addr = INADDR_LOOPBACK,
    };
```

Листинг 3.7 – Прототип клиента (часть 3)

```
    struct rpc_create_args args = {
        .net = &init_net,
        .protocol = XPRT_TRANSPORT_TCP,
        .address = (struct sockaddr *)&sin,
        .addrsize = sizeof(sin),
        .program = &program,
        .version = 1,
        .authflavor = RPC_AUTH_NULL,
        .cred = current_cred(),
        .flags = RPC_CLNT_CREATE_NOPING | RPC_CLNT_CREATE_REUSEPORT,
    };
    struct rpc_clnt *inner = rpc_create(&args);
    if (!IS_ERR(inner)) {
        client = inner;
    }
    return inner;
}

static int __init init_md(void) {
    int rc = 0;
    struct rpc_clnt *client;
    if (IS_ERR(client = get_client())) {
        rc = PTR_ERR(client);
        client = NULL;
    }
    if (0 == rc) {
        u64 num = 10;
        u64 res;
        struct rpc_message msg = {
            .rpc_proc = &progrs[1],
            .rpc_argp = &num,
            .rpc_resp = &res,
            .rpc_cred = get_current_cred(),
        };
        rc = rpc_call_sync(client, &msg, 0);
        printk("+ Got result %lld\n", res);
    }
    if (0 == rc) {
        printk("+ rpc client module start\n");
    } else {
        printk("+ Error: %pe\n", ERR_PTR(rc));
    }
    if (client) {
        rpc_shutdown_client(client);
    }
    return rc;
}
```

Листинг 3.8 – Прототип клиента (часть 4)

```
static void __exit exit_md(void) {
    printk("+ rpc module end\n");
}
module_init(init_md);
module_exit(exit_md);
```

3.3 Сведения о модулях разрабатываемого приложения

Таким образов, исходя из приведенного макета, была составлена следующая структура организации исходного кода модулей ядра, приведенная на листинге 3.9–3.10.

Листинг 3.9 – Организаци кода модулей ядра (часть 1)

```
.
+-- clients                      -- модули клиента
|   +-- <program_name>
|   |   +-- <version_name>       -- описание версии
|   |   |   +-- constants.h      -- номера процедур
|   |   |   +-- procedure_api.c  -- экспортируемые вызовы удаленных
|   |   |   +-- procedure_api.h     процедур
|   |   |   +-- procedures.h     -- определение функций кодирования и
|   |   |   |                       оберток аргументов
|   |   |   +-- procedure_xdr.c  -- объявление функций кодирования
|   |   |   +-- version.c        -- объявление версии
|   |   |   +-- version.h        -- определение версии
|   |   +-- authentication.c     -- кодирование пользовательской
|   |   +-- authentication.h        аутентификации
|   |   +-- constants.h          -- номера версий
|   |   +-- program.c            -- объявление программы и точка входа
|   |                               модуля
|   +-- client.c                 -- общая работа со структурой rpc_clnt
|   +-- client.h
+-- servers -- модули клиента
|   +-- <program_name>
|   |   +-- <version_name>       -- описание версии
|   |   |   +-- constants.h      -- номера процедур
|   |   |   +-- procedure_handlers.c -- обработчики процедур
|   |   |   +-- procedures.h     -- определение процедур
|   |   |   +-- procedure_xdr.c  -- объявление функция кодирования и
|   |   |   |                       освобождения
```

Листинг 3.10 – Организаци кода модулей ядра (часть 2)

```
|   |   |   +-- version.c        -- объявление версии
|   |   |   +-- version.h        -- определение версии
|   |   +-- authentication.c     -- обработчик пользовательской
|   |   +-- authentication.h        аутентификации
|   |   +-- constants.h          -- номера версий
|   |   +-- program.c            -- объявление программы и точка входа
|   |                               модуля
|   +-- common.c                 -- разделяемые функции threadfn и
|   +-- common.h                    dispatch
+-- constants.h                  -- опредление констант и enum
+-- types.h                      -- определение остальных типов
+-- Makefile
```

Приложение разбито на модули в соответствии с основным алгоритмом:

— `lexer` — содержит реализацию лексического анализатора и основные наборы правил;

— `rpc` — содержит описание составных частей токенов rpc;

— `rpc_lexer` — создает и настраивает анализатор для разбора спецификации;

— `rpc_parser` — преобразует поток токенов в поток определений;

— `rpc_generator` — преобразует поток определений в исходный код модулей ядра.

3.4 Реализация приложения

Код основных частей приложения представлен на рисунках 3.11–3.56.

Листинг 3.11 – Типаж лексического анализатора и его стандартная реализация (часть 1)

```
pub trait Lexer<T> {
    fn parse(
        self: &mut Self,
        input: impl std::io::Read,
    ) -> impl Iterator<Item=Result<T>>;
```

Листинг 3.12 – Типаж лексического анализатора и его стандартная реализация (часть 2)

```
    fn parse_str(
        self: &mut Self,
        string: &str,
    ) -> impl Iterator<Item=Result<T>> {
        self.parse(std::io::Cursor::new(string))
    }
}

pub trait MatchRule<T: Clone> {
    fn get(self: &mut Self) -> Box<dyn Matcher<T>>;
}

pub trait Matcher<T: Clone> {
    fn check(self: &mut Self, c: Char) -> State<T>;
    fn reset(self: &mut Self);
}

pub trait SkipRule {
    fn get(self: &mut Self) -> Box<dyn Skip>;
}

pub trait Skip {
    fn is_skipping(self: &mut Self, c: char) -> bool;
}

pub struct Lexer<T> {
    rules: Vec<Box<dyn MatchRule<T>>>,
    skip: Option<Box<dyn SkipRule>>,
}

impl<T: Clone> crate::Lexer<T> for Lexer<T> {
    fn parse(
        self: &mut Self, input: impl std::io::Read
    ) -> impl Iterator<Item = Result<T>> {
        TokenIterator::new(
            &mut self.rules, self.skip.as_mut(),
            CodePoints::from(input),
        )
    }
}

struct MatcherState<T: Clone> {
    matcher: Box<dyn Matcher<T>>,
    last: State<T>,
}
```

Листинг 3.13 – Типаж лексического анализатора и его стандартная реализация (часть 3)

```
impl<T: Clone> MatcherState<T> {
    fn new(matcher: Box<dyn Matcher<T>>) -> Self {
        Self {
            matcher,
            last: State::Matching,
        }
    }
}

impl<T: Clone> Matcher<T> for MatcherState<T> {
    fn check(self: &mut Self, c: Char) -> State<T> {
        self.last = self.matcher.check(c);
        self.last.clone()
    }

    fn reset(self: &mut Self) {
        self.last = State::Matching;
        self.matcher.reset();
    }
}

pub struct TokenIterator<I, T: Clone>
where
    I: Iterator<Item = std::io::Result<char>>,
{
    matchers: Vec<MatcherState<T>>,
    skip: Option<Box<dyn Skip>>,
    chars: I,
    prev: Option<Char>,
}

impl<I, T: Clone> TokenIterator<I, T>
where
    I: Iterator<Item = std::io::Result<char>>,
{
    fn new(
        matchers: &mut [Box<dyn MatchRule<T>>],
        skip: Option<&mut Box<dyn SkipRule>>,
        iter: I,
    ) -> Self {
        Self {
            matchers: matchers.iter_mut()
                .map(|m| MatcherState::new(m.get())).collect(),
            skip: skip.map(|r| r.get()),
            chars: iter,
```

Листинг 3.14 – Типаж лексического анализатора и его стандартная реализация (часть 4)

```
            prev: None,
        }
    }

    fn next_char(self: &mut Self) -> std::io::Result<Char> {
        if let Some(c) = self.prev.take() {
            Ok(c)
        } else {
            self.chars.next()
                .map(|r| r.map(Char::Char))
                .unwrap_or(Ok(Char::EOF))
        }
    }

    fn reset(self: &mut Self) {
        self.matchers.iter_mut().for_each(MatcherState::reset)
    }
}

enum MatchLock {
    None,
    Matching,
    Matched,
}

impl<I, T: Clone> Iterator for TokenIterator<I, T>
where
    I: Iterator<Item = std::io::Result<char>>,
{
    type Item = Result<T>;

    fn next(&mut self) -> Option<Self::Item> {
        let mut active = self.matchers.len();
        let mut matched = 0;
        let mut error: Option<Error> = None;
        let mut last = Char::EOF;
        let mut empty = false;
        self.reset();
        self.skip = self.skip.take().and_then(|mut skip| {
            let mut start = false;
            while error.is_none() && !start {
                match self.next_char() {
                    Err(err) => error = Some(Error::io(err)),
                    Ok(c) => {
                        last = c;
```

Листинг 3.15 – Типаж лексического анализатора и его стандартная реализация (часть 5)

```
                        start = match c {
                            Char::EOF => true,
                            Char::Char(c) => !skip.is_skipping(c),
                        };
                    },
                }
            }
            if error.is_none() {
                self.prev = Some(last);
            }
            Some(skip)
        });
        if error.is_none() {
            match self.next_char() {
                Err(err) => {
                    self.prev = None;
                    error = Some(Error::io(err));
                }
                Ok(c) => {
                    self.prev = Some(c);
                    if let Char::EOF = c {
                        empty = true;
                    }
                },
            }
            if empty {
                return None;
            }
        }
        while error.is_none() && 0 == matched && 0 != active {
            let mut matching = MatchLock::None;
            match self.next_char() {
                Err(err) => error = Some(Error::io(err)),
                Ok(c) => {
                    last = c;
                    self.matchers.iter_mut()
                        .filter(|m| match m.last {
                            State::Rejected => false,
                            _ => true,
                        })
                        .for_each(|m| match m.check(c) {
                            State::Rejected => active -= 1,
```

Листинг 3.16 – Типаж лексического анализатора и его стандартная реализация (часть 6)

```
                            State::Matched(_) => {
                                if let MatchLock::Matching = matching {
                                    error = Some(
                                        Error::broken_grammar_msg(
                                            "Unreacheable higher \
                                             order rule found"
                                        )
                                    );
                                } else {
                                    matching = MatchLock::Matched;
                                    active -= 1;
                                    matched += 1;
                                }
                            },
                            State::Matching => {
                                if let MatchLock::None = matching {
                                    matching = MatchLock::Matching;
                                }
                            }
                        });
                }
            }
        }
        if let Some(error) = error {
            Some(Err(error))
        } else if 0 == matched {
            Some(Err(match last {
                Char::EOF => Error::unexpected_eof(),
                _ => Error::unknown_token(),
            }))
        } else {
            self.prev = Some(last);
            Some(Ok(
                self.matchers.iter().find_map(|m| match &m.last {
                    State::Matched(v) => Some(v),
                    _ => None,
                }).expect("Counter isn't 0").clone()
            ))
        }
    }
}
```

Листинг 3.17 – Разбор заготовленной последовательности байт (часть 1)

```
pub struct CharSequenceMatcher<T, FG, FA>
where
    T: Clone,
    FG: FnMut() -> T,
    FA: FnMut(Char) -> bool,
{
    chars: Vec<char>,
    last: usize,
    resf: FG,
    allowed: FA,
    cooked: bool,
}

impl<T, FG, FA> Matcher<T> for CharSequenceMatcher<T, FG, FA>
where
    T: Clone,
    FG: FnMut() -> T,
    FA: FnMut(Char) -> bool,
{
    fn check(self: &mut Self, c: Char) -> State<T> {
        if self.cooked {
            return State::Rejected
        }
        let res = match self.chars.len().cmp(&self.last) {
            std::cmp::Ordering::Greater => match c {
                Char::EOF => State::Rejected,
                Char::Char(c) => {
                    let target = self.chars
                        .get(self.last)
                        .expect("Was checked");
                    self.last += 1;
                    if c == *target {
                        State::Matching
                    } else {
                        State::Rejected
                    }
                }
            },
            std::cmp::Ordering::Equal => {
                if !(self.allowed)(c) {
                    State::Matched((self.resf)())
                } else {
                    State::Rejected
                }
            },
            std::cmp::Ordering::Less => panic!("Index overflow"),
        };
```

Листинг 3.18 – Разбор заготовленной последовательности байт (часть 2)

```
        if let State::Rejected | State::Matched(_) = res {
            self.cooked = true;
        }
        res
    }

    fn reset(self: &mut Self) {
        self.last = 0;
        self.cooked = false;
    }
}
```

Листинг 3.19 – Разбор целых чисел (часть 1)

```
enum RadixState {
    None,
    Pending,
    Set(u8),
}

impl RadixState {
    fn get(self: &Self) -> u32 {
        match self {
            Self::None | Self::Pending => 10,
            Self::Set(r) => *r as u32,
        }
    }
}

pub struct IntegerMatcher<T, F>
where
    T: Clone,
    F: FnMut(i64) -> T
{
    radix: RadixState,
    cooked: bool,
    any: bool,
    number: i64,
    resf: F,
    sign: i64,
}

impl<T, F> Matcher<T> for IntegerMatcher<T, F>
where
    T: Clone,
    F: FnMut(i64) -> T
{
```

Листинг 3.20 – Разбор целых чисел (часть 2)

```
    fn check(self: &mut Self, c: Char) -> State<T> {
        if self.cooked {
            return State::Rejected
        }
        let r = self.radix.get();
        let res = match c {
            Char::EOF => {
                if self.any {
                    State::Matched(
                        (self.resf)(self.number * self.sign)
                    )
                } else {
                    State::Rejected
                }
            },
            Char::Char(c) => {
                if let RadixState::Pending = self.radix {
                    let mut lc = c.to_lowercase();
                    match (lc.next(), lc.next()) {
                        (Some('b'), None) => {
                            self.radix = RadixState::Set(2);
                            State::Matching
                        },
                        (Some('o'), None) => {
                            self.radix = RadixState::Set(8);
                            State::Matching
                        },
                        (Some('x'), None) => {
                            self.radix = RadixState::Set(16);
                            State::Matching
                        },
                        _ => {
                            if let Some(n) = c.to_digit(r) {
                                self.radix = RadixState::None;
                                self.number = n as i64;
                                State::Matching
                            } else {
                                State::Matched((self.resf)(0))
                            }
                        },
                    }
                } else if !self.any {
                    if '-' == c {
                        if 0 > self.sign {
                            State::Rejected
                        } else {
```

Листинг 3.21 – Разбор целых чисел (часть 3)

```
                            self.sign = -1;
                            State::Matching
                        }
                    } else if '0' == c {
                        self.any = true;
                        self.radix = RadixState::Pending;
                        State::Matching
                    } else if let Some(n) = c.to_digit(r) {
                        self.any = true;
                        self.number = n as i64;
                        State::Matching
                    } else {
                        State::Rejected
                    }
                } else {
                    if let Some(n) = c.to_digit(r) {
                        self.number =
                            self.number * (r as i64) + n as i64;
                        State::Matching
                    } else {
                        State::Matched(
                            (self.resf)(self.number * self.sign)
                        )
                    }
                }
            }
        };
        if let State::Rejected | State::Matched(_) = res {
            self.cooked = true;
        }
        res
    }

    fn reset(self: &mut Self) {
        self.radix = RadixState::None;
        self.cooked = false;
        self.any = false;
        self.number = 0;
        self.sign = 1;
    }
}
```

Листинг 3.22 – Разбор идентификатора (часть 1)

```
struct IdentifierMatcher {
    current: String,
    cooked: bool,
}

impl IdentifierMatcher {
    fn new() -> Self {
        Self {
            current: String::new(),
            cooked: false,
        }
    }
}

impl Matcher<token::Token> for IdentifierMatcher {
    fn check(self: &mut Self, c: Char) -> State<token::Token> {
        if self.cooked {
            return State::Rejected
        }
        let res = match c {
            Char::EOF => {
                if "" == self.current {
                    State::Rejected
                } else {
                    State::Matched(
                        token::Token::Identifier(self.current.clone())
                    )
                }
            },
            Char::Char(c) => {
                if "" == self.current {
                    if !char::is_alphanumeric(c) {
                        State::Rejected
                    } else {
                        let mut buf: [u8; 4] = [0; 4];
                        self.current += c.encode_utf8(&mut buf);
                        State::Matching
                    }
                } else if allowed_chars(Char::Char(c)) {
                    let mut buf: [u8; 4] = [0; 4];
                    self.current += c.encode_utf8(&mut buf);
                    State::Matching
                } else if "" != self.current {
                    State::Matched(
                        token::Token::Identifier(self.current.clone())
                    )
                } else {
```

Листинг 3.23 – Разбор идентификатора (часть 2)

```
                    State::Rejected
                }
            },
        };
        if let State::Rejected | State::Matched(_) = res {
            self.cooked = true
        }
        res
    }

    fn reset(self: &mut Self) {
        self.current.clear();
        self.cooked = false;
    }
}
```

Листинг 3.24 – Разбор определения (часть 1)

```
struct PickIterator<I: Iterator<Item=token::Token>> {
    picked: Option<token::Token>,
    iter: I,
}

impl<I: Iterator<Item=token::Token>> PickIterator<I> {
    fn new(iter: I) -> Self {
        Self {
            picked: None,
            iter,
        }
    }

    fn push_back(self: &mut Self, t: token::Token) {
        match self.picked {
            Some(_) => panic!("Supposed to pick only one token"),
            None => self.picked = Some(t),
        }
    }
}

impl<I: Iterator<Item=token::Token>> Iterator for PickIterator<I> {
    type Item = token::Token;

    fn next(&mut self) -> Option<Self::Item> {
        if let Some(t) = self.picked.take() {
            Some(t)
        } else {
            self.iter.next()
```

Листинг 3.25 – Разбор определения (часть 2)

```
        }
    }
}

struct PendingTypes {
    structs: Option<String>,
    unions: Option<String>,
}

struct DefinedTypes {
    typedefs: std::collections::HashSet<String>,
    enums: std::collections::HashSet<String>,
    structs: std::collections::HashSet<String>,
    unions: std::collections::HashSet<String>,
}

struct Handle<I: Iterator<Item=token::Token>> {
    tokens: PickIterator<I>,
    namespace: std::collections::HashSet<String>,
    values: std::collections::HashSet<String>,
    pending_types: PendingTypes,
    defined_types: DefinedTypes,
    assigned_numbers: std::collections::HashSet<rpc::Value>,
}

pub fn parse(
    tokens: impl Iterator<Item=token::Token>,
) -> Result<rpc::Module> {
    let mut module = rpc::new_module();
    let mut handle = Handle {
        tokens: PickIterator::new(tokens.filter(|t| match t {
            token::Token::Comment(_) => false,
            _ => true,
        })),
        namespace: std::collections::HashSet::new(),
        values: std::collections::HashSet::new(),
        pending_types: PendingTypes {
            structs: None,
            unions: None,
        },
        defined_types: DefinedTypes {
            typedefs: std::collections::HashSet::new(),
            enums: std::collections::HashSet::new(),
            structs: std::collections::HashSet::new(),
            unions: std::collections::HashSet::new(),
        },
```

Листинг 3.26 – Разбор определения (часть 3)

```
        assigned_numbers: std::collections::HashSet::new(),
    };
    let mut err = None;
    while let (None, Some(t)) = (&err, handle.tokens.next()) {
        handle.tokens.push_back(t);
        match parse_definition(&mut handle) {
            Ok(def) => module.definitions.push(def),
            Err(error) => err = Some(error),
        }
    }
    match err {
        Some(err) => Err(err),
        None => Ok(module),
    }
}

fn parse_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Definition> {
    match handle.tokens.next() {
        None =>
            Error::unexpected_eof("Definition expected".to_string()),
        Some(token::Token::Type(token::Type::Enum)) =>
            parse_enum_definition(handle)
                .map(|(id, en)| rpc::Definition::Enum(id, en)),
        Some(token::Token::Keyword(token::Keyword::Const)) =>
            parse_const_definition(handle)
                .map(|(id, v)| rpc::Definition::Const(id, v)),
        Some(token::Token::Keyword(token::Keyword::Typedef)) =>
            parse_typedef_definition(handle)
                .map(|(id, tp)| rpc::Definition::Typedef(id, tp)),
        Some(token::Token::Type(token::Type::Struct)) =>
            parse_struct_definition(handle)
                .map(|(id, st)| rpc::Definition::Struct(id, st)),
        Some(token::Token::Type(token::Type::Union)) =>
            parse_union_definition(handle)
                .map(|(id, un)| rpc::Definition::Union(id, un)),
        Some(token::Token::Keyword(token::Keyword::Program)) =>
            parse_program_definition(handle)
                .map(|(v, pr)| rpc::Definition::Program(v, pr)),
        Some(t) => Error::unknown_definition(t),
    }
}
```

Листинг 3.27 – Разбор значения

```
fn parse_value(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Value> {
    parse_value_condition(handle, |num| Ok(num))
}
fn parse_value_condition<F: FnOnce(i64) -> Result<i64>>(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
    cond: F,
) -> Result<rpc::Value> {
    match handle.tokens.next() {
        None => Error::unexpected_eof("Expected value".to_string()),
        Some(token::Token::Literal(token::Literal::Integer(num))) =>
            cond(num).map(|num| rpc::Value::Number(num)),
        Some(token::Token::Identifier(id)) => {
            match handle.values.get(&id) {
                Some(_) => Ok(rpc::Value::Identifier(id)),
                None => Error::undefined_value(id),
            }
        },
        Some(t) =>
            Error::unexpected_token("Expected value".to_string(), t),
    }
}
```

Листинг 3.28 – Разбор типа (часть 1)

```
fn parse_type_identifier(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Type> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Expected declaration type".to_string()
        ),
        Some(token::Token::Type(t)) => match t {
            token::Type::Void => Ok(rpc::Type::Void),
            token::Type::Unsigned => match handle.tokens.next() {
                None => Error::unexpected_eof(
                    "Expected integer type".to_string()
                ),
                Some(token::Token::Type(token::Type::Integer)) =>
                    Ok(rpc::Type::Unsigned(rpc::Integer::Integer)),
                Some(token::Token::Type(token::Type::Hyper)) =>
                    Ok(rpc::Type::Unsigned(rpc::Integer::Hyper)),
                Some(t) => Error::unexpected_token(
                    "Expected integer type".to_string(), t,
                ),
            },
```

Листинг 3.29 – Разбор типа (часть 2)

```
            token::Type::Integer =>
                Ok(rpc::Type::Integer(rpc::Integer::Integer)),
            token::Type::Hyper =>
                Ok(rpc::Type::Integer(rpc::Integer::Hyper)),
            token::Type::Float =>
                Ok(rpc::Type::Float(rpc::Float::Single)),
            token::Type::Double =>
                Ok(rpc::Type::Float(rpc::Float::Double)),
            token::Type::Quadruple =>
                Ok(rpc::Type::Float(rpc::Float::Quadruple)),
            token::Type::Boolean => Ok(rpc::Type::Boolean),
            token::Type::String => Ok(rpc::Type::String),
            token::Type::Opaque => Ok(rpc::Type::Opaque),
            token::Type::Enum => match handle.tokens.next() {
                None => Error::unexpected_eof(
                    "No identifier for enum".to_string()
                ),
                Some(token::Token::Identifier(id)) => {
                    match handle.defined_types.enums.get(&id) {
                        None => Error::undefined_type(format!{
                            "Unknown enum with identifier {id}"
                        }),
                        _ => Ok(
                            rpc::Type::Named(rpc::NamedType::Enum(id))
                        ),
                    }
                },
                Some(t) => Error::unexpected_token(
                    "No identifier for enum".to_string(),
                    t
                ),
            },
            token::Type::Struct => match handle.tokens.next() {
                None => Error::unexpected_eof(
                    "No identifier for struct".to_string()
                ),
                Some(token::Token::Identifier(id)) => {
                    match (
                        handle.defined_types.structs.get(&id),
                        &handle.pending_types.structs
                    ) {
                        (None, None) => Error::undefined_type(format!{
                            "Unknown struct with identifier {id}"
                        }),
                        (None, Some(pid)) if *pid != id =>
                            Error::undefined_type(format!{
                                "Unknown struct with identifier {id}"
```

Листинг 3.30 – Разбор типа (часть 3)

```
                        }),
                    _ => Ok(rpc::Type::Named(
                        rpc::NamedType::Struct(id)
                    ))
                }
            },
            Some(t) => Error::unexpected_token(
                "No identifier for struct".to_string(), t,
            ),
        },
        token::Type::Union => match handle.tokens.next() {
            None => Error::unexpected_eof(
                "No identifier for union".to_string()
            ),
            Some(token::Token::Identifier(id)) => {
                match (
                    handle.defined_types.unions.get(&id),
                    &handle.pending_types.unions,
                ) {
                    (None, None) => Error::undefined_type(format!{
                        "Unknown union with identifier {id}"
                    }),
                    (None, Some(pid)) if *pid != id =>
                        Error::undefined_type(format!{
                            "Unknown union with identifier {id}"
                        }),
                    _ => Ok(
                        rpc::Type::Named(rpc::NamedType::Union(id))
                    )
                }
            },
            Some(t) => Error::unexpected_token(
                "No identifier for union".to_string(), t,
            ),
        },
        token::Type::Pointer => Error::undefined_type(
            "No type for pointer".to_string()
        ),
    },
    Some(token::Token::Identifier(id)) => {
        match handle.defined_types.typedefs.get(&id) {
            None => Error::undefined_type(format!{
                "Unknown type identifier {id}"
            }),
            _ => Ok(rpc::Type::Named(rpc::NamedType::Typedef(id))),
        }
    },
```

Листинг 3.31 – Разбор типа (часть 4)

```
        Some(t) => Error::unexpected_token(
            "Expected declaration type".to_string(), t,
        ),
    }.and_then(|tp| Ok(match handle.tokens.next() {
        Some(token::Token::Type(token::Type::Pointer)) =>
            rpc::Type::Pointer(Box::new(tp)),
        None => tp,
        Some(t) => {
            handle.tokens.push_back(t);
            tp
        }
    })).and_then(|tp| match tp {
        rpc::Type::Named(nm) => match nm {
            rpc::NamedType::Struct(st) => {
                match &handle.pending_types.structs {
                    Some(rst) if st == *rst =>
                        Error::use_of_pending_type(format!(
                            "Can't use struct \"{st}\" that is being \
                             defined directly"
                        )),
                    _ => Ok(
                        rpc::Type::Named(rpc::NamedType::Struct(st))
                    ),
                }
            },
            rpc::NamedType::Union(un) => {
                match &handle.pending_types.unions {
                    Some(run) if un == *run =>
                        Error::use_of_pending_type(format!(
                            "Can't use union \"{un}\" that is being \
                             defined directly"
                        )),
                    _ => Ok(
                        rpc::Type::Named(rpc::NamedType::Union(un))
                    ),
                }
            },
            _ => Ok(rpc::Type::Named(nm)),
        },
        _ => Ok(tp),
    })
}

fn parse_array_type(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
    tp: rpc::Type,
) -> Result<rpc::Type> {
```

Листинг 3.32 – Разбор типа (часть 5)

```
    match handle.tokens.next() {
        Some(token::Token::Bracket(br)) => match br {
            token::Bracket::LeftTriangle => {
                match handle.tokens.next() {
                    None => Error::unexpected_eof(
                        "Expected variadic array closing bracket or \
                         size hint".to_string()
                    ),
                    Some(token::Token::Bracket(
                        token::Bracket::RightTriangle
                    )) =>
                        Ok(rpc::Type::VArray(Box::new(tp), None)),
                    Some(t) => {
                        handle.tokens.push_back(t);
                        parse_value_condition(handle, |num| {
                            if 0 >= num {
                                Error::non_positive_array_size(num)
                            } else {
                                Ok(num)
                            }
                        }).and_then(|v| match handle.tokens.next() {
                            None => Error::unexpected_eof(
                                "Expected variadic array closing \
                                 bracket".to_string()
                            ),
                            Some(token::Token::Bracket(
                                token::Bracket::RightTriangle
                            )) => Ok(rpc::Type::VArray(
                                Box::new(tp), Some(v),
                            )),
                            Some(t) => Error::unexpected_token(
                                "Expected variadic array closing \
                                 bracket".to_string(), t,
                            ),
                        })
                    },
                }
            },
            token::Bracket::LeftSquare => parse_value_condition(
                handle,
                |num| if 0 >= num {
                    Error::non_positive_array_size(num)
                } else {
                    Ok(num)
                }
            ).and_then(|v| match handle.tokens.next() {
                None => Error::unexpected_eof(
```

Листинг 3.33 – Разбор типа (часть 6)

```
                    "Expected array closing bracket".to_string()
                ),
                Some(token::Token::Bracket(
                    token::Bracket::RightSquare
                )) => Ok(rpc::Type::Array(Box::new(tp), v)),
                Some(t) => Error::unexpected_token(
                    "Expected array closing bracket".to_string(), t,
                ),
            }),
            _ => {
                handle.tokens.push_back(token::Token::Bracket(br));
                Ok(tp)
            }
        },
        None => Ok(tp),
        Some(t) => {
            handle.tokens.push_back(t);
            Ok(tp)
        }
    }
}

fn parse_type(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Type> {
    parse_type_identifier(handle)
        .and_then(|tp| parse_array_type(handle, tp))
}
```

Листинг 3.34 – Разбор объявления

```
fn parse_declaration(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Type)> {
    parse_type_identifier(handle)
        .and_then(|tp| match handle.tokens.next() {
            None => Error::unexpected_eof(
                "Expected declaration identifier".to_string()
            ),
            Some(token::Token::Identifier(name)) => Ok((name, tp)),
            Some(t) => Error::unexpected_token(
                "Expected declaration identifier".to_string(), t,
            ),
        }).and_then(|(name, tp)| {
            parse_array_type(handle, tp).map(|tp| (name, tp))
        })
}
```

Листинг 3.35 – Разбор enum (часть 1)

```
fn parse_enum_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Enum)> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "No enum identifier was provided".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.defined_types.enums.get(&id) {
                Some(_) => Error::type_redefined(format!{
                    "Enum with id \"{id}\" already exists"
                }),
                None => Ok(id),
            }
        },
        Some(t) => Error::unexpected_token(
            "Expected enum identifier".to_owned(), t,
        ),
    }.and_then(|id| match handle.tokens.next() {
        None => Error::unexpected_eof("No enum body".to_owned()),
        Some(token::Token::Bracket(token::Bracket::LeftCurly)) =>
            Ok(id),
        Some(t) => Error::unexpected_token(
            "Expected enum body \"{\"".to_owned(), t,
        ),
    }).and_then(|id| parse_enum_body(handle).map(|en| (id, en)))
    .and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Enum definition wasn't finished".to_owned(),
        ),
        Some(token::Token::Bracket(token::Bracket::RightCurly)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Enum definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Enum definition wasn't finished".to_owned(),
        ),
        Some(token::Token::Separator(token::Separator::Semicolon)) =>
            Ok(pass),
        Some(t) => Error::expression_not_closed(
            "Enum definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|(id, en)| {
        handle.defined_types.enums.insert(id.clone());
        Ok((id, en))
```

Листинг 3.36 – Разбор enum (часть 2)

```
    })
}

fn parse_enum_item(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, Option<rpc::Value>)> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Expected enum item identifier".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.namespace.get(&id) {
                Some(_) => Error::identifier_redefined(format!(
                    "Enum identifier \"{id}\" already exists"
                )),
                None => Ok(id),
            }
        },
        Some(t) => Error::expression_not_closed(
            "Expected enum item identifier".to_owned(), t,
        ),
    }.and_then(|id| match handle.tokens.next() {
        None => Ok((id, None)),
        Some(token::Token::Operator(token::Operator::Assign)) =>
            parse_value(handle).map(|v| (id, Some(v))), // [Value]
        Some(t) => {
            handle.tokens.push_back(t);
            Ok((id, None))
        }
    }).and_then(|(id, v)| {
        handle.namespace.insert(id.clone());
        handle.values.insert(id.clone());
        Ok((id, v))
    })
}

fn parse_enum_body(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Enum> {
    let mut en = rpc::new_enum();
    let mut error: Option<Error> = None;
    while match parse_enum_item(handle) {
        Ok(item) => {
            en.push(item);
            match handle.tokens.next() {
                None => false,
                Some(
```

Листинг 3.37 – Разбор enum (часть 3)

```
                    token::Token::Separator(token::Separator::Comma)
                ) => true,
                Some(t) => {
                    handle.tokens.push_back(t);
                    false
                }
            }
        },
        Err(err) => {
            error = Some(err);
            false
        },
    } {}
    match error {
        None => Ok(en),
        Some(err) => Err(err),
    }
}
```

Листинг 3.38 – Разбор константы (часть 1)

```
fn parse_const_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Value)> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Expected const identifier".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.namespace.get(&id) {
                Some(_) => Error::identifier_redefined(format!(
                    "Constant identifier \"{id}\" already exists"
                )),
                None => Ok(id),
            }
        }
        Some(t) => Error::expression_not_closed(
            "Expected const identifier".to_owned(), t,
        ),
    }.and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Expected assign sign".to_string()
        ),
        Some(
            token::Token::Operator(token::Operator::Assign),
        ) => Ok(pass),
```

Листинг 3.39 – Разбор константы (часть 2)

```
        Some(t) => Error::unexpected_token(
            "Expected assign sign".to_string(), t,
        ),
    }).and_then(|id| parse_value(handle).map(|v| (id, v)))
    .and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("Const definition wasn't \
                                       finished".to_owned()),
        Some(
            token::Token::Separator(token::Separator::Semicolon)
        ) => Ok(pass),
        Some(t) => Error::expression_not_closed(
            "Const definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|(id, v)| {
        handle.namespace.insert(id.clone());
        handle.values.insert(id.clone());
        Ok((id, v))
    })
}
```

Листинг 3.40 – Разбор typedef (часть 1)

```
fn parse_typedef_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Type)> {
    parse_declaration(handle)
        .and_then(|(id, tp)| match handle.namespace.get(&id) {
            Some(_) => Error::type_redefined(format!(
                "Type with identifier {id} already exists"
            )),
            None => Ok((id, tp)),
        }).and_then(|pass| match handle.tokens.next() {
            None => Error::unexpected_eof(
                "Const definition wasn't finished".to_owned()
            ),
            Some(
                token::Token::Separator(token::Separator::Semicolon)
            ) => Ok(pass),
            Some(t) => Error::expression_not_closed(
                "Const definition wasn't finished".to_owned(), t,
            ),
        }).and_then(|(id, tp)| {
            handle.namespace.insert(id.clone());
            handle.defined_types.typedefs.insert(id.clone());
            Ok((id, tp))
        })
}
```

Листинг 3.41 – Разбор структуры (часть 1)

```
fn parse_struct_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Struct)> {
    let out = match handle.tokens.next() {
        None => Error::unexpected_eof(
            "No struct identifier was provided".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.defined_types.structs.get(&id) {
                Some(_) => Error::type_redefined(format!{
                    "Struct with id \"{id}\" already exists"
                }),
                None => {
                    handle.pending_types.structs = Some(id.clone());
                    Ok(id)
                },
            }
        },
        Some(t) => Error::unexpected_token(
            "Expected struct identifier".to_owned(), t
        ),
    }.and_then(|id| match handle.tokens.next() {
        None => Error::unexpected_eof("No struct body".to_owned()),
        Some(
            token::Token::Bracket(token::Bracket::LeftCurly)
        ) => Ok(id),
        Some(t) => Error::unexpected_token(
            "Expected struct body \"{\"".to_owned(), t,
        ),
    }).and_then(|id| parse_struct_body(handle).map(|en| (id, en)))
    .and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Struct definition wasn't finished".to_owned()
        ),
        Some(
            token::Token::Bracket(token::Bracket::RightCurly)
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Struct definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Struct definition wasn't finished".to_owned(),
        ),
        Some(
            token::Token::Separator(token::Separator::Semicolon)
        ) => Ok(pass),
```

Листинг 3.42 – Разбор структуры (часть 2)

```
        Some(t) => Error::expression_not_closed(
            "Struct definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|(id, en)| {
        handle.defined_types.structs.insert(id.clone());
        Ok((id, en))
    });
    handle.pending_types.structs = None;
    out
}

fn parse_struct_body(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Struct> {
    let mut st = rpc::new_struct();
    let mut error: Option<Error> = None;
    while match parse_declaration(handle)
        .and_then(|(id, tp)| match st.get(&id) {
            Some(_) => Error::structure_field_redefined(format!(
                "Field with identifier \"{id}\" already exists"
            )),
            None => {
                st.insert(id, tp);
                match handle.tokens.next() {
                    None => Error::unexpected_eof(
                        "Structure field declaraion wasn't \
                         finished".to_string(),
                    ),
                    Some(
                        token::Token::Separator(
                            token::Separator::Semicolon
                        )
                    ) => Ok(()),
                    Some(t) => Error::unexpected_token(
                        "Structure field declaraion wasn't \
                         finished".to_string(),
                        t,
                    ),
                }.and_then(|_| match handle.tokens.next() {
                    None => Ok(false),
                    Some(
                        token::Token::Bracket(
                            token::Bracket::RightCurly
                        )
                    ) => {
                        handle.tokens.push_back(
                            token::Token::Bracket(
```

Листинг 3.43 – Разбор структуры (часть 3)

```
                                token::Bracket::RightCurly
                            )
                        );
                        Ok(false)
                    },
                    Some(t) => {
                        handle.tokens.push_back(t);
                        Ok(true)
                    },
                })
            },
        }) {
        Ok(next) => next,
        Err(err) => {
            error = Some(err);
            false
        },
    } {}
    match error {
        None => Ok(st),
        Some(err) => Err(err),
    }
}
```

Листинг 3.44 – Разбор объединения (часть 1)

```
fn parse_union_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(String, rpc::Union)> {
    let out = match handle.tokens.next() {
        None => Error::unexpected_eof(
            "No union identifier was provided".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.defined_types.unions.get(&id) {
                Some(_) => Error::type_redefined(format!{
                    "Union with id \"{id}\" already exists"
                }),
                None => {
                    handle.pending_types.unions = Some(id.clone());
                    Ok(id)
                },
            }
        },
        Some(t) => Error::unexpected_token(
            "Expected union identifier".to_owned(), t,
        ),
```

Листинг 3.45 – Разбор объединения (часть 2)

```
    }.and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Keyword \"case\" expected".to_owned(),
        ),
        Some(
            token::Token::Keyword(token::Keyword::Switch),
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Keyword \"case\" expected".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("Expected \"(\"".to_owned()),
        Some(token::Token::Bracket(token::Bracket::Left)) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected \"(\"".to_owned(), t,
        ),
    }).and_then(|id| parse_declaration(handle).and_then(|(sid, tp)|
        match tp {
            rpc::Type::Integer(i) =>
                Ok(rpc::SwitchingType::Integer(i)),
            rpc::Type::Unsigned(u) =>
                Ok(rpc::SwitchingType::Unsigned(u)),
            rpc::Type::Named(rpc::NamedType::Enum(en)) =>
                Ok(rpc::SwitchingType::Enum(en)),
            tp => Error::not_switching_type(tp),
        }.map(|stp| (id, (sid, stp)))
    )).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("Expected \"(\"".to_owned()),
        Some(token::Token::Bracket(token::Bracket::Right)) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected \"(\"".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("No union body".to_owned()),
        Some(token::Token::Bracket(token::Bracket::LeftCurly)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected union body \"{\"".to_owned(), t,
        ),
    }).and_then(|(id, (sid, stp))| parse_union_body(handle)
        .map(|mut un| {
            un.value = sid;
            un.switch_type = stp;
            (id, un)
        }
    )).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
```

Листинг 3.46 – Разбор объединения (часть 3)

```
            "Union definition wasn't finished".to_owned(),
        ),
        Some(token::Token::Bracket(token::Bracket::RightCurly)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Union definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Union definition wasn't finished".to_owned(),
        ),
        Some(token::Token::Separator(token::Separator::Semicolon)) =>
            Ok(pass),
        Some(t) => Error::expression_not_closed(
            "Union definition wasn't finished".to_owned(), t,
        ),
    }).and_then(|(id, en)| {
        handle.defined_types.unions.insert(id.clone());
        Ok((id, en))
    });
    handle.pending_types.unions = None;
    out
}

enum UnionItem {
    Regular(rpc::Value, (String, rpc::Type)),
    Default(String, rpc::Type),
}

fn parse_union_item(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<UnionItem> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Matching value expected".to_owned()
        ),
        Some(token::Token::Keyword(token::Keyword::Case)) =>
            parse_value(handle)
                .and_then(|pass| match handle.tokens.next() {
                    None => Error::unexpected_eof(
                        "Colon expected".to_owned(),
                    ),
                    Some(
                        token::Token::Separator(
                            token::Separator::Colon
                        )
                    ) => Ok(pass),
```

Листинг 3.47 – Разбор объединения (часть 4)

```
                    Some(t) => Error::unexpected_token(
                        "Colon expected".to_owned(), t,
                    ),
                }).and_then(|v| {
                    parse_declaration(handle)
                        .map(|decl| UnionItem::Regular(v, decl))
                }),
        Some(token::Token::Keyword(token::Keyword::Default)) => {
            match handle.tokens.next() {
                None => Error::unexpected_eof(
                    "Colon expected".to_owned(),
                ),
                Some(
                    token::Token::Separator(
                        token::Separator::Colon
                    )
                ) => Ok(()),
                Some(t) => Error::unexpected_token(
                    "Colon expected".to_owned(), t,
                ),
            }.and_then(|_| {
                parse_declaration(handle)
                    .map(|(id, tp)| UnionItem::Default(id, tp))
            })
        },
        Some(t) => Error::unexpected_token(
            "Matching value expected".to_owned(), t,
        ),
    }
}

fn parse_union_body(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Union> {
    let mut un = rpc::new_union();
    let mut error: Option<Error> = None;
    while match parse_union_item(handle).and_then(|item|
        match item {
            UnionItem::Regular(v, decl) => match un.arms.get(&v) {
                Some(_) => Error::union_arm_redefined(v),
                None => {
                    un.arms.insert(v, decl);
                    Ok(true)
                },
            },
            UnionItem::Default(id, tp) => match &un.default {
                Some(_) => Error::union_default_redefined(),
```

Листинг 3.48 – Разбор объединения (часть 5)

```
                None => {
                    un.default = Some((id, tp));
                    Ok(false)
                },
            },
        }.and_then(|next| match handle.tokens.next() {
            None => Error::unexpected_eof(
                "Union arm declaraion wasn't finished".to_string(),
            ),
            Some(
                token::Token::Separator(
                    token::Separator::Semicolon
                )
            ) => Ok(next),
            Some(t) => Error::unexpected_token(
                "Union arm declaraion wasn't finished".to_string(), t,
            ),
        }).and_then(|next| match next {
            false => Ok(false),
            true => match handle.tokens.next() {
                None => Ok(false),
                Some(token::Token::Bracket(
                    token::Bracket::RightCurly
                )) => {
                    handle.tokens.push_back(token::Token::Bracket(
                        token::Bracket::RightCurly
                    ));
                    Ok(false)
                },
                Some(t) => {
                    handle.tokens.push_back(t);
                    Ok(true)
                },
            }
        })
    ) {
        Ok(next) => next,
        Err(err) => {
            error = Some(err);
            false
        },
    } {}
    match error {
        None => Ok(un),
        Some(err) => Err(err),
    }
}
```

Листинг 3.49 – Разбор программы (часть 1)

```
fn parse_program_definition(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(rpc::Value, rpc::Program)> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Program identifier expected".to_owned()
        ),
        Some(token::Token::Identifier(id)) => {
            match handle.namespace.get(&id) {
                Some(_) => Error::identifier_redefined(format!(
                    "Program identifier \"{id}\" already exists"
                )),
                None => Ok(id),
            }
        },
        Some(t) => Error::unexpected_token(
            "Program identifier expected".to_owned(), t,
        ),
    }.and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("No program body".to_owned()),
        Some(
            token::Token::Bracket(token::Bracket::LeftCurly)
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected program body \"{\"".to_owned(), t,
        ),
    }).and_then(|id| parse_program_versions(handle).map(|mut pr| {
        pr.name = id;
        pr
    })).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Program body not closed".to_owned(),
        ),
        Some(
            token::Token::Bracket(token::Bracket::RightCurly),
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Program body not closed".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Number not assigned to program".to_owned(),
        ),
        Some(
            token::Token::Operator(token::Operator::Assign),
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
```

Листинг 3.50 – Разбор объединения (часть 2)

```
            "Number not assigned to program".to_owned(), t,
        ),
    }).and_then(|pr| parse_value(handle).map(|v| (v, pr)))
    .and_then(|(v, pr)| match handle.assigned_numbers.get(&v) {
        Some(_) => Error::program_number_reassigned(v),
        None => Ok((v, pr)),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Program definition not closed".to_owned(),
        ),
        Some(
            token::Token::Separator(token::Separator::Semicolon),
        ) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Program definition not closed".to_owned(), t,
        ),
    }).and_then(|(v, pr)| {
        handle.namespace.insert(pr.name.clone());
        handle.assigned_numbers.insert(v.clone());
        Ok((v, pr))
    })
}

fn parse_program_versions(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Program> {
    let mut out = rpc::new_program();
    let mut error: Option<Error> = None;
    let mut version_names =
        std::collections::HashSet::<String>::new();
    let mut version_values =
        std::collections::HashSet::<rpc::Value>::new();
    while match parse_version(handle)
        .and_then(|(v, proc)| match version_values.get(&v) {
            Some(_) => Error::version_number_reassigned(v),
            None => match version_names.get(&proc.name) {
                Some(_) => Error::identifier_redefined(format!(
                    "Version with identifier \"{}\" already exists in \
                     current program", proc.name,
                )),
                None => {
                    version_values.insert(v.clone());
                    version_names.insert(proc.name.clone());
                    out.versions.insert(v, proc);
                    Ok(())
                }
            },
```

Листинг 3.51 – Разбор объединения (часть 3)

```
        })
        .and_then(|_| Ok(match handle.tokens.next() {
            None => false,
            Some(
                token::Token::Bracket(token::Bracket::RightCurly),
            ) => {
                handle.tokens.push_back(
                    token::Token::Bracket(token::Bracket::RightCurly)
                );
                false
            },
            Some(t) => {
                handle.tokens.push_back(t);
                true
            },
        })) {
        Ok(next) => next,
        Err(err) => {
            error = Some(err);
            false
        }
    } {}
    match error {
        Some(error) => Err(error),
        None => Ok(out),
    }
}
```

Листинг 3.52 – Разбор версии (часть 1)

```
fn parse_version(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(rpc::Value, rpc::Version)> {
    match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Version identifier expected".to_owned(),
        ),
        Some(token::Token::Keyword(token::Keyword::Version)) => Ok(()),
        Some(t) => Error::unexpected_token(
            "Version identifier expected".to_owned(), t,
        ),
    }.and_then(|_| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Version identifier expected".to_owned(),
        ),
        Some(token::Token::Identifier(id)) => Ok(id),
        Some(t) => Error::unexpected_token(
```

Листинг 3.53 – Разбор версии (часть 2)

```
            "Version identifier expected".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("No version body".to_owned()),
        Some(token::Token::Bracket(token::Bracket::LeftCurly)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected version body \"{\"".to_owned(), t,
        ),
    }).and_then(|id| parse_version_procedures(handle).map(|mut ver| {
        ver.name = id;
        ver
    })).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Version body not closed".to_owned(),
        ),
        Some(token::Token::Bracket(token::Bracket::RightCurly)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Version body not closed".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Number not assigned to version".to_owned(),
        ),
        Some(token::Token::Operator(token::Operator::Assign)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Number not assigned to version".to_owned(), t,
        ),
    }).and_then(|ver| parse_value(handle).map(|v| (v, ver)))
    .and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Version definition not closed".to_owned(),
        ),
        Some(token::Token::Separator(token::Separator::Semicolon)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Version definition not closed".to_owned(), t,
        ),
    })
}

fn parse_version_procedures(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Version> {
    let mut out = rpc::new_version();
```

Листинг 3.54 – Разбор версии (часть 3)

```
    let mut error: Option<Error> = None;
    let mut procedure_names =
        std::collections::HashSet::<String>::new();
    let mut procedure_values =
        std::collections::HashSet::<rpc::Value>::new();
    while match parse_procedure(handle)
        .and_then(|(v, proc)| match procedure_values.get(&v) {
            Some(_) => Error::procedure_number_reassigned(v),
            None => match procedure_names.get(&proc.name) {
                Some(_) => Error::identifier_redefined(format!(
                    "Procedure with identifier \"{}\" already exists \
                     in current version", proc.name,
                )),
                None => {
                    procedure_values.insert(v.clone());
                    procedure_names.insert(proc.name.clone());
                    out.procedures.insert(v, proc);
                    Ok(())
                }
            },
        })
        .and_then(|_| Ok(match handle.tokens.next() {
            None => false,
            Some(
                token::Token::Bracket(token::Bracket::RightCurly)
            ) => {
                handle.tokens.push_back(
                    token::Token::Bracket(token::Bracket::RightCurly)
                );
                false
            },
            Some(t) => {
                handle.tokens.push_back(t);
                true
            },
        })) {
        Ok(next) => next,
        Err(err) => {
            error = Some(err);
            false
        }
    } {}
    match error {
        Some(error) => Err(error),
        None => Ok(out),
    }
}
```

Листинг 3.55 – Разбор процедуры (часть 1)

```
fn parse_procedure(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<(rpc::Value, rpc::Procedure)> {
    parse_type(handle).and_then(|tp| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Procedure identifier expected".to_owned()
        ),
        Some(token::Token::Identifier(id)) => Ok((tp, id)),
        Some(t) => Error::unexpected_token(
            "Procedure identifier expected".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof("No procedure body".to_owned()),
        Some(token::Token::Bracket(token::Bracket::Left)) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Expected procedure body \"{\"".to_owned(), t,
        ),
    }).and_then(|(tp, id)| parse_procedure_args(handle)
        .map(|mut proc| {
            proc.name = id;
            proc.return_type = tp;
            proc
        })
    ).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Procedure body not closed".to_owned()
        ),
        Some(token::Token::Bracket(token::Bracket::Right)) => Ok(pass),
        Some(t) => Error::unexpected_token(
            "Procedure body not closed".to_owned(), t,
        ),
    }).and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Number not assigned to procedure".to_owned(),
        ),
        Some(token::Token::Operator(token::Operator::Assign)) =>
            Ok(pass),
        Some(t) => Error::unexpected_token(
            "Number not assigned to procedure".to_owned(), t,
        ),
    }).and_then(|proc| parse_value(handle).map(|v| (v, proc)))
    .and_then(|pass| match handle.tokens.next() {
        None => Error::unexpected_eof(
            "Procedure definition not closed".to_owned()
        ),
        Some(token::Token::Separator(token::Separator::Semicolon)) =>
            Ok(pass),
```

Листинг 3.56 – Разбор процедуры (часть 2)

```
        Some(t) => Error::unexpected_token(
            "Procedure definition not closed".to_owned(), t,
        ),
    })
}

fn parse_procedure_args(
    handle: &mut Handle<impl Iterator<Item=token::Token>>,
) -> Result<rpc::Procedure> {
    let mut out = rpc::new_procedure();
    let mut error: Option<Error> = None;
    while match parse_type(handle)
        .and_then(|tp| match tp {
            rpc::Type::Void if 0 == out.arguments.len() => Ok(false),
            _ => {
                out.arguments.push(tp);
                Ok(match handle.tokens.next() {
                    None => false,
                    Some(token::Token::Separator(
                        token::Separator::Comma
                    )) => true,
                    Some(t) => {
                        handle.tokens.push_back(t);
                        false
                    },
                })
            }
        }) {
        Ok(next) => next,
        Err(err) => {
            error = Some(err);
            false
        }
    } {}
    match error {
        Some(error) => Err(error),
        None => Ok(out),
    }
}
```

Из соображений о размере отчета, код генерации модулей приведен не будет.

3.5 Конфигурирование генерируемых модулей

Для обеспечения некоторой гибкости генерируемых модулей, была добавлена возможность их конфигурации, а именно: возможность выбора порта и количества потоков для сервера; возможность указания ip и порта назначения сервера, а также требуемой версии для клиента. Для реализации использовался макрос `module_param`. Исходный код представлен на листингах 3.57–3.58.

Листинг 3.57 – Конфигурирование сервера

```
1 static unsigned short port = 0;
2 module_param(port, ushort, 0);
3
4 static unsigned int threads = 1;
5 module_param(threads, uint, 0);
```

Листинг 3.58 – Конфигурирование клиента

```
1  static unsigned int version = 1;
2  module_param(version, uint , 0);
3
4  static unsigned int host_ip_size = 4;
5  static unsigned char host_ip[4] = {127, 0, 0 ,1};
6  module_param_array(host_ip, byte, &host_ip_size, 0);
7
8  static unsigned short port = 0;
9  module_param(port, ushort, 0);
10
11 static __be32 decode_host_ip(void) {
12     __u8 decode_buffer[4];
13
14     for (size_t i = 0; host_ip_size > i; i++) {
15         decode_buffer[i] = host_ip[i];
16     }
17
18     for (size_t i = host_ip_size; 4 > i; i++) {
19         decode_buffer[i] = 0;
20     }
21
22     return *(__be32*)decode_buffer;
23 }
```

3.6 Тестирование разработанного приложения

Результаты тестирования представлены в таблицах 3.1–3.2 и на рисунках 3.1–3.2.

Таблица 3.1 – Тестирование лексического анализатора

№	Исходные данные	Ожидаемый результат
1	Ping программа из стандарта [2]	Соответствующий набор токенов
2	Bakery из курса операционных систем	Соответствующий набор токенов
3	Разбор неиспользованных типов `enum` и `union`	Соответствующий набор токенов

```
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/worktrees/develop/lib/rpc_lexer]
$ cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.03s
     Running unittests src/lib.rs (target/debug/deps/rpc_lexer-a4d709837d71a14e)

running 3 tests
test test::ping ... ok
test test::union_enum ... ok
test test::bakery ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Рисунок 3.1 – Тестирование лексического анализатора

Таблица 3.2 – Тестирование синтаксического анализатора

№	Исходные данные	Ожидаемый результат
1	Ping программа изстандарта	Определенией программы PING_PROGR, содержащее 2 версии: PING_VERS_PINGBACK, состоящей из двух процедур PINGPROC_NULL и PINGPROC_PINGBACK; PING_VERS_PINGBACK, состоящей из одной процедуры PINGPROC_NULL, а также определение константы PING_VERS
2	Bakery из курса операционных систем	Определение констант, определение структуры BAKERY,определение программы BAKERY_PROG с версией BAKERY_VER и процедурой BACKERY_PROC
3	Разбор неиспользованных типов enum	Определение enum

```
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/worktrees/develop/lib/rpc_parser]
$ cargo test
    Finished `test` profile [unoptimized + debuginfo] target(s) in 0.03s
     Running unittests src/lib.rs (target/debug/deps/rpc_parser-ae2417d2ce627fb4)

running 3 tests
test test::enum_test ... ok
test test::ping ... ok
test test::bakery ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

Рисунок 3.2 – Тестирование синтаксического анализатора

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование.

— Операционная система: Arch [6] Linux 6.12.7 [7] x86_64.

— Память: 16 Гб.

— Процессор: AMD Ryzen™ 5 4600H CPU @ 3.0G ГГц [8].

Исследование проводилось на ноутбуке. Во время тестирования устройство было нагружено только встроенными приложениями окружения.

4.2 Демонстрация работы приложения

Для демонстрации работы программы было разработано приложение для удаленного получения информации о процессах системы. Файл спецификации приведен на листинге 4.1.

Листинг 4.1 – Файл спецификации примера

```
const NAME_LEN = 16;
const MAX_TASKS = 10;
struct task {
    string name<NAME_LEN>;
    int pid;
    unsigned int state;
    unsigned int flags;
};
typedef struct task tasks<MAX_TASKS>;
program example {
    version initial {
        tasks get_tasks(void) = 1;
    } = 1;
} = 0x20000001;
```

Демонстрация работы программы представлена на рисунках 4.1–4.2.

```
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/shared/report/example]
$ sudo modprobe sunrpc
[sudo] password for daniil:
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/shared/report/example]
$ sudo rpcbind
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/shared/report/example]
$ sudo insmod example_server.ko
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/shared/report/example]
$ sudo insmod example_api.ko
[daniil@LAPTOP-813EJBL /storage/Daniil/work/lab/os/course_proj/shared/report/example]
$ sudo insmod usage.ko
```

Рисунок 4.1 – Последовательность загрузки модулей

```
Jan 31 14:44:30 LAPTOP-813EJBL kernel: [example] RPC server started at port: 37599
Jan 31 14:44:35 LAPTOP-813EJBL sudo[180226]:   daniil : TTY=pts/8 ; PWD=/storage/Daniil/work/lab/os/course_
mple ; USER=root ; COMMAND=/usr/bin/insmod example_api.ko
Jan 31 14:44:35 LAPTOP-813EJBL sudo[180226]: pam_unix(sudo:session): session opened for user root(uid=0) by
Jan 31 14:44:35 LAPTOP-813EJBL kernel: [example] Client side api loaded
Jan 31 14:44:35 LAPTOP-813EJBL sudo[180226]: pam_unix(sudo:session): session closed for user root
Jan 31 14:44:41 LAPTOP-813EJBL sudo[180253]:   daniil : TTY=pts/8 ; PWD=/storage/Daniil/work/lab/os/course_
mple ; USER=root ; COMMAND=/usr/bin/insmod usage.ko
Jan 31 14:44:41 LAPTOP-813EJBL sudo[180253]: pam_unix(sudo:session): session opened for user root(uid=0) by
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - swapper/0, pid - 0, state - 0, flags - 69206018
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - systemd, pid - 1, state - 1, flags - 4194560
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kthreadd, pid - 2, state - 1, flags - 2129984
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - pool_workqueue_, pid - 3, state - 1, flags - 2129984
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/R-rcu_g, pid - 4, state - 1026, flags - 69238880
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/R-sync_, pid - 5, state - 1026, flags - 69238880
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/R-slub_, pid - 6, state - 1026, flags - 69238880
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/R-netns, pid - 7, state - 1026, flags - 69238880
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/0:0H, pid - 10, state - 1026, flags - 69238880
Jan 31 14:44:41 LAPTOP-813EJBL kernel: + self - kworker/R-mm_pe, pid - 13, state - 1026, flags - 69238880
```

Рисунок 4.2 – Результат работы

Заключение

При выполнении курсовой работы был проведен анализ протоколов ONC PRC и XDR, а также их реализация в ядре операционной системы Linux, были выделены и рассмотрены структуры для описания предоставляемых (запрашиваемых) процедур и соответствующие фунции ядра для запуска сервера и выполнения запроса. Были разработаны алгоритмы для преобразования спецификации на языке RPCL в поток токенов, для его последующего преобразования в поток определений и генерации на его основе кода загружаемых модулей ядра. Была разработан минимальный макет взаимодействия загружаемых модулей ядра, и утилита, способная генерировать код загружаемых модулей ядра, взаимодействующих по модели клиент сервер согласно протоколу ONC RPC на основе файла спецификации на языке RPCL, подобно приложению `rpcgen` из пространства пользователя.

Проведенное исследование показало, что разработанное приложение полностью соответствует техническому заданию, а именно позволяет генерировать код загружаемых модулей ядра, взаимодействующих по сетевой модели, согласно протоколу ONC RPC и файлу спецификации на языке RPCL, а также предоставляет возможность конфигурации полученных модулей, а именно указание порта и ip адреса сервера, количества потоков обработчиков и используемой версии.

Список использованных источников

1 NFS: Network File System Protocol specification. Режим доступа: URL – https://www.rfc-editor.org/info/rfc1094. (Дата обращения: 5 февраля 2025 г.).

2 Thurlow Robert. RPC: Remote Procedure Call Protocol Specification Version 2 [Электронный ресурс]. Режим доступа: URL – https://www.rfc-editor.org/info/rfc5531. (Дата обращения: 5 февраля 2025 г.).

3 Srinivasan Raj. Binding Protocols for ONC RPC Version 2 [Электронный ресурс]. Режим доступа: URL – https://www.rfc-editor.org/info/rfc1833. (Дата обращения: 5 февраля 2025 г.).

4 Eisler Mike. XDR: External Data Representation Standard. Режим доступа: URL – https://www.rfc-editor.org/info/rfc4506. (Дата обращения: 5 февраля 2025 г.).

5 index : kernel/git/stable/linux.git [Электронный ресурс]. Режим доступа: URL – https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/?h=v6.12.7. (Дата обращения: 5 февраля 2025 г.).

6 Arch Linux [Электронный ресурс]. Режим доступа: URL – https://archlinux.org/. (Дата обращения: 5 февраля 2025 г.).

7 Linux – Getting Started [Электронный ресурс]. Режим доступа: URL – https://linux.org. (Дата обращения: 5 февраля 2025 г.).

8 Процессор AMD Ryzen™ 5 4600H [Электронный ресурс]. Режим доступа: URL – https://www.amd.com/en/support/downloads/drivers.html/processors/ryzen/ryzen-4000-series/amd-ryzen-5-4600h.html. (Дата обращения: 5 февраля 2025 г.).