

Netlink e Netlink para Wifi

Comunicações
Móveis 2021/2022
DETI - UA
Daniel Corujo

Esboço

- Tomadas
- Biblioteca Netlink (libnl)
- biblioteca nl80211

Biblioteca Netlink (libnl) em um slide

- Permite o uso de tomadas Netlink Comunicação
 - Conectar/Desconectar
 - Envio/Recebimento de dados
 - Construção/partilhamento de mensagens
 - Máquinas de recepção de mensagens de estado
 - Estruturas de dados
- Utiliza o Protocolo Netlink
 - Mecanismo de comunicação inter-processo baseado em soquetes
 - Processos de espaço do utilizador <-> kernel
 - Processo de espaço do utilizador <-> Processo de espaço do utilizador
 - É um serviço orientado a datagramas

(O que é uma tomada?)

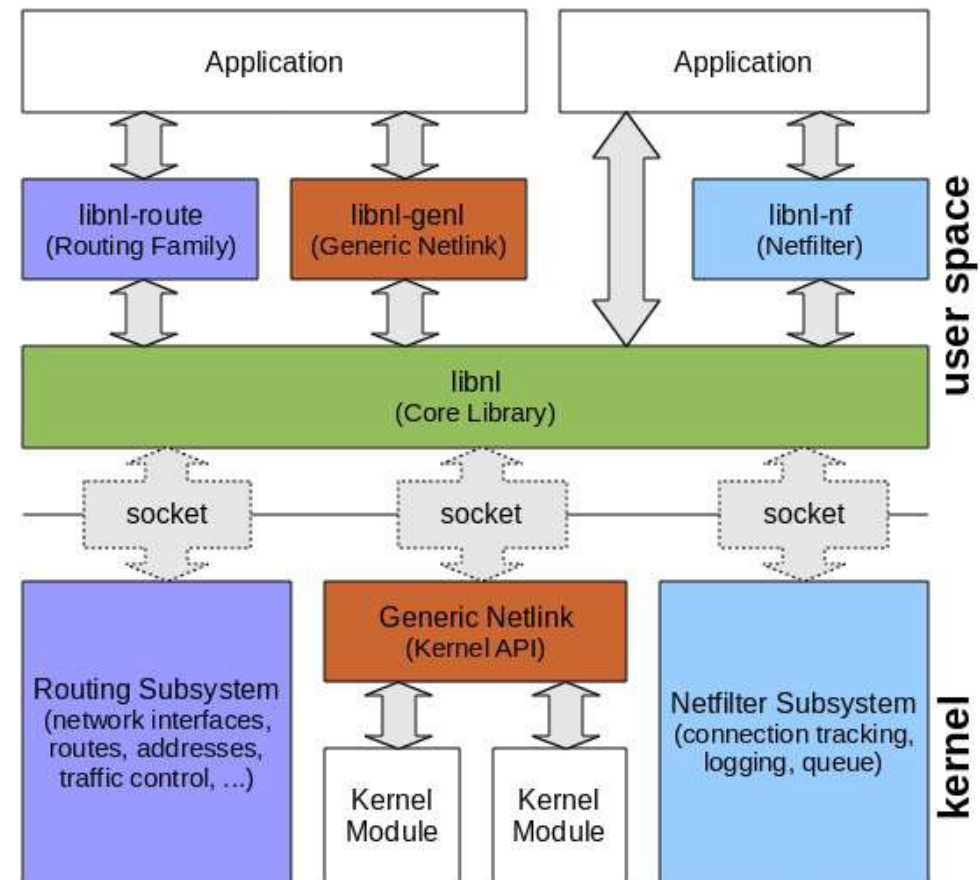
- Abstracção de um ponto final para o intercâmbio de dados entre processos
 - Gasoduto
 - `história | grep gcc`
 - Tomadas de Rede
 - Entre diferentes anfitriões
 - Tomadas Berkeley
 - Unix Tomada de domínio / Tomada IPC
 - Tomada de Internet
 - Tomadas Netlink
 - Apenas para anfitriões
 - Baseado em tomadas BSD

(O que é uma tomada)

- A tomada Berkeley utiliza um Descritor de Ficheiros
 - Identifica um objeto para os recursos do sistema
 - Os descritores de arquivo não podem ser acessados diretamente pelos processos de espaço do usuário.
 - Os processos de espaço do usuário fazem uma chamada de sistema ao kernel, fornecendo-lhe a referência do descritor de arquivo, e o kernel acessa o recurso (ou seja, input/output) em seu nome
- Tomada Netlink usada para usar o PID
 - Não mais (multi-tarefas!)
- Agora usa um número de porta de 32 bits

Biblioteca Netlink (libnl)

- Libnl
 - Biblioteca Netlink
 - Manuseamento de tomadas
 - Envio e recepção
 - Construção e análise de mensagens
- Libnl-genl
 - Biblioteca Netlink Genérica
 - Controlador API
 - Registo familiar e de comando



Famílias de tomadas Netlink

- AF_NETLINK
 - Suporta diferentes subsecções, cada uma visando diferentes componentes e mensagens do kernel (ou seja, famílias de endereços)
- Tipos de soquetes
 - SOCK_RAW
 - SOCK_DGRAM
- família Netlink
 - Selecciona o módulo do kernel ou grupo netlink para comunicar com

Famílias Netlink

- NETLINK_ROUTE
- NETLINK_W1
 - 1 fio
- NETLINK_USERSOCK
- NETLINK_FIREWALL
- NETLINK_SOCK_DIAG
- NETLINK_NFLOG
 - Netfilter/iptables log
- NETLINK_XFRM
- NETLINK_SELINUX
- NETLINK_ISCSI
- NETLINK_AUDIT
- NETLINK_FIB_LOOKUP
- NETLINK_CONNECTOR
- NETLINK_NETFILTER
- NETLINK_GENERIC
- ...

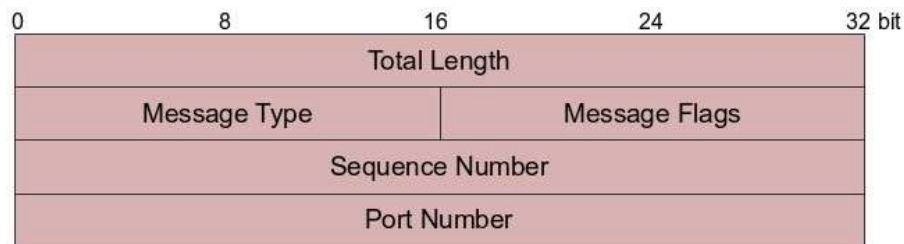
Protocolo Netlink

Protocolo Netlink

- Mecanismo de comunicação inter-processo baseado em soquetes
 - Processos de espaço do utilizador <-> kernel
 - Processo de espaço do utilizador <-> Processo de espaço do utilizador
- É um serviço orientado a datagramas

Formato da Mensagem de Datagramas

- Tipo de mensagem
 - Tipo de carga útil
- Bandeiras de mensagens
 - Modificar o comportamento do tipo de mensagem
 - Solicitação
 - Multicast
 - Agradecimentos
 - Echo



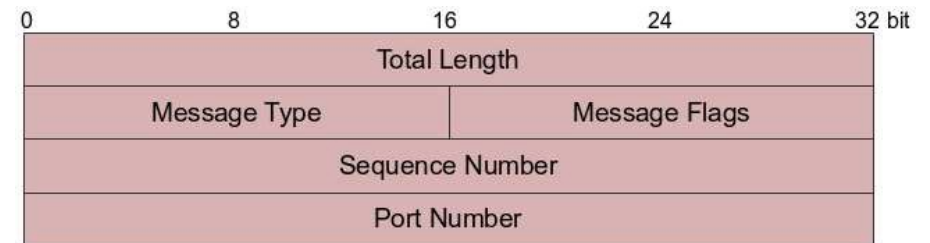
Tipos de Mensagens

Netlink

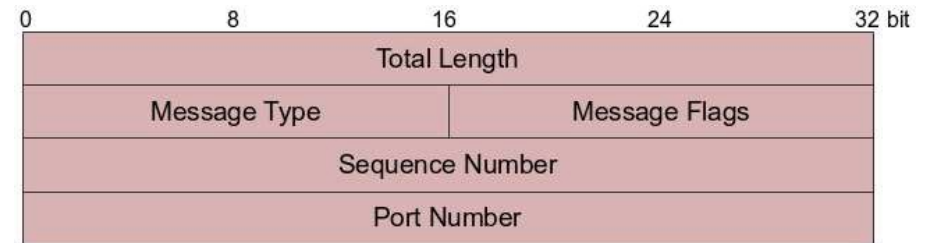
- Pedidos
- Notificações
- Respostas

Tipos de mensagem (parâmetro datagramas)

- NLMSG_NOOP
 - Nenhuma operação, mensagem deve ser descartada
- NLMSG_ERROR
 - Mensagem de erro ou ACK
- NLMSG_DONE
 - Fim da sequência multipartes
- NLMSG_OVERRUN
 - Notificação de anulação (Erro)



Bandeiras de mensagens



- NLM_F_REQUEST - Mensagem é um pedido
- NLM_F_MULTI - Mensagem multipartes
- NLM_F_ACK - Mensagem ACK solicitada
- NLM_F_ECHO - Solicitação de ecoar a solicitação
- Bandeiras universais para pedidos de GET
 - NLM_F_ROOT - Retorno com base na raiz da árvore.
 - NLM_F_MATCH - Devolver todas as entradas correspondentes.
 - NLM_F_ATOMIC - Obsoleto, uma vez usado para solicitar uma operação atômica.
 - NLM_F_DUMP - Retorna uma lista de todos os objetos (NLM_F_ROOT|NLM_F_MATCH).

Tomadas Netlink

Tomadas Netlink

- Necessidade de usar o protocolo Netlink
- É onde você envia e recebe mensagens de protocolo

Exemplo de programa

- Vamos construir um programa que espera por notificações NETLINK_ROUTE. Quando um é recebido, ele chama uma função do usuário (ou seja, **função de retorno de chamada**)
- Temos de o fazer:
 - Criar uma tomada
 - Indicar qual é a função de retorno de chamada
 - Ligue a tomada à NETLINK_ROUTE
 - Inscrever-se para receber notificações específicas, de um grupo multicast específico
 - Mantenha o programa em execução, ouvindo as notificações
 - *(uma notificação é um evento que é enviado quando alguma ação específica ocorreu)*

Exemplo de programa

//este programa espera por notificações NETLINK_ROUTE. Quando um é recebido, ele chama uma função do usuário (ou seja, **função de retorno de chamada**)

//função

```
static int my_func(struct nl_msg *msg, void *arg){  
    printf("Foi recebida uma mensagem! Esta é a minha  
    função"); retornar 0; }
```

Exemplo de programa

//Alocar uma nova tomada

```
stuct nl_sock * sk = nl_socket_alloc();
```

//Disable sequence number checking, uma vez que estamos apenas a usar notificações

//Netlink cuida automaticamente dos números de sequência, ao utilizar o nl_send_auto() para enviar mensagens.


// No entanto, se estivermos usando um protocolo netlink sem pedido/resposta, devemos desativá-lo explicitamente:

Exemplo de

programa
`nl_socket_disable_seq_check(sk);`

Exemplo de programa

//Callback função que chama "meu_func".



```
nl_socket_modify_cb(sk, NL_CB_VALID, NL_CB_CUSTOM,  
my_func,  
nulo);
```

//Conectar-se ao protocolo de
roteamento de rede

```
nl_connect(sk, NETLINK_ROUTE);
```

```
int nl_socket_modify_cb(struct nl_sock *sk, enum nl_cb_type type, enum nl_cb_kind kind,  
                        nl_recvmsg_msg_cb_t func, void *arg);
```

Exemplo de programa

```
//subscrição para ligar notificações multicast  
grupo nl_socket_add_membership(sk,  
RTNLGRP_LINK,0);
```

```
//Ouvir as mensagens
```

```
//Isso acionará a função de retorno de chamada, quando  
uma notificação do protocolo NETLINK_ROUTE for recebida
```

```
enquanto (1)
```

```
    nl_rcvmsgs_default(sk);
```

Funções relacionadas com a tomada

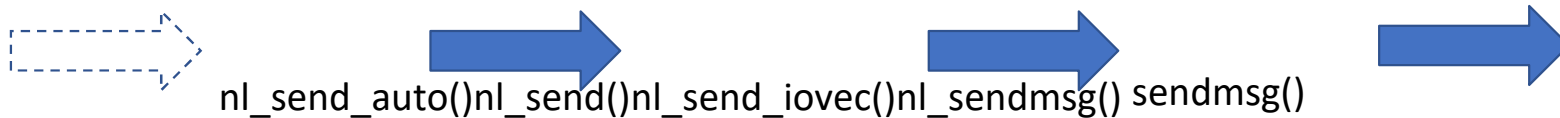
- `nl_socket_set_local_port`
- `nl_socket_set_peer_port`
- `nl_socket_get_fd`
- `nl_socket_set_buffer_size`
- ...

Envio de mensagens através de uma tomada netlink

- De duas maneiras
 - Automático
 - `nl_send_auto()`
 - Completa automaticamente os bits que faltam no cabeçalho da mensagem netlink
 - Trata automaticamente do endereçamento com base nas informações atuais definidas no soquete
 - Passa a mensagem para `nl_send()`
 - Cria a mensagem final
 - Manual (quando o enchimento automático não é adequado)
 - Usar diretamente `nl_send()`
 - Incorpora a mensagem numa estrutura 'iovec' e passa-a para `nl_send_iovec()`
 - `nl_send()` também pode ser sobrescrito via `nl_cb_overwrite_send()`

Envio de mensagens através de uma tomada netlink

- Os dois caminhos levam a
 - `nl_send_iovec()`
 - Preenche o cabeçalho da mensagem - tenta encontrar o par, ou deixa-o para o kernel
 - Leva a:
 - `nl_sendmsg()`
 - Tira a mensagem final e o cabeçalho opcional
 - Envia a mensagem final total para `sendmsg()`



Recepção de mensagens através de uma tomada netlink

- O nosso exemplo de programa recebeu mensagens (Notificações) do kernel usando a seguinte função:

enquanto (1)

```
nl_recvmsgs_default(sk);
```

- Função mais fácil
 - Recebe mensagens baseadas em como configuramos o socket
 - Normalmente, o comportamento padrão é suficiente
 - Obtém a função callbackfunction (`cb = nl_socket_get_cb(sk)`)
 - chamadas `nl_recvmsgs()`

Recepção de mensagens através de uma tomada netlink

- `nl_recvmsgs()`
 - Loop de recepção de mensagens reais
 - Se precisarmos de características específicas de recepção, podemos providenciar uma implementação completa do mecanismo de recepção
 - `nl_cb_overwrite_recvmsgs()`

Mensagens de análise

- As mensagens são 4 bytes alinhadas em todos os limites
- Há dois métodos de análise
 - Interface de baixo nível (análise manual)
 - Interface de alto nível (Implementar um analisador como parte das operações de cache)
- O que está recebendo uma mensagem de protocolo netlink em uma tomada netlink?
 - O que você recebe de uma tomada netlink é tipicamente um fluxo de mensagens.
 - Ser-lhe-á dado um amortecedor e o seu comprimento
 - O buffer pode conter qualquer número de mensagens netlink.
 - O primeiro cabeçalho da mensagem começa no início do fluxo de mensagens
 - Você pode alcançar o próximo cabeçalho chamando `nlmsg_next()` no cabeçalho anterior
 - Posição = Número_restante_de_bytes - tamanho_de_mensagem_atual

Mensagens de análise

- Apesar de termos `nlmsg_next()`, não sabemos se há mais mensagens
- Devemos assumir que mais mensagens seguem até que todos os bytes do fluxo tenham sido processados.
 - `nlmsg_ok()`
 - Retorna verdadeiro se outra mensagem se encaixar no número restante de bytes no fluxo de mensagens
 - `nlmsg_valid_hdr()`
 - Verifica se uma mensagem contém pelo menos um mínimo de carga útil

Criação de uma função de análise de mensagens: exemplo

```
#include <netlink/msg.h>
```

```
void my_parse(void *stream, int length){
```

```
    struct nlmsghdr *hdr = stream;
```

```
    while (nlmsg_ok(hdr, length)) {
```

```
        // Mensagem parse aqui
```

```
        hdr = nlmsg_next(hdr, &length);
```

```
    }
```

```
}
```

Mensagens de análise

- Acesso à carga útil das mensagens
 - Lembre-se que o cabeçalho tem alinhamento
 - Alguns dos seus campos, e o próprio cabeçalho, podem ter acolchoamento
 - `nlmsg_data()` retorna um ponteiro para o início da carga útil
 - `nlmsg_dataen()` retorna a duração da carga útil da mensagem
 - `nlmsg_tail()` devolve um ponteiro para o fim da carga útil, incluindo o acolchoamento



Atributos

- A maioria das mensagens do protocolo netlink usa atributos netlink
- Isto significa que, a carga útil da mensagem é composta por
 - Cabeçalho do Protocolo (+ estofamento)
 - Atributos (+ estofamento)
- `nlmsg_attrdata()` retorna um ponteiro para o início da seção de atributos
- `nlmsg_attrlen()` retorna o comprimento da seção de atributos

função de análise

```
int nlmsg_parse(struct nlmsg_hdr *hdr, int hdrlen, struct nlattr **attrs,  
               int maxtype, struct nla_policy *policy);
```

- nlmsg_parse()
 - Começa por validar o cabeçalho
 - Se hdrlen>0, chama nlmsg_valid_hdr()
 - Preenche um array com indicações para cada atributo
- nlmsg_validate()
 - Similar, mas não cria a matriz
- Existem também variantes de atribuição desta função
 - nla_parse()
 - nla_validate() não cria o array

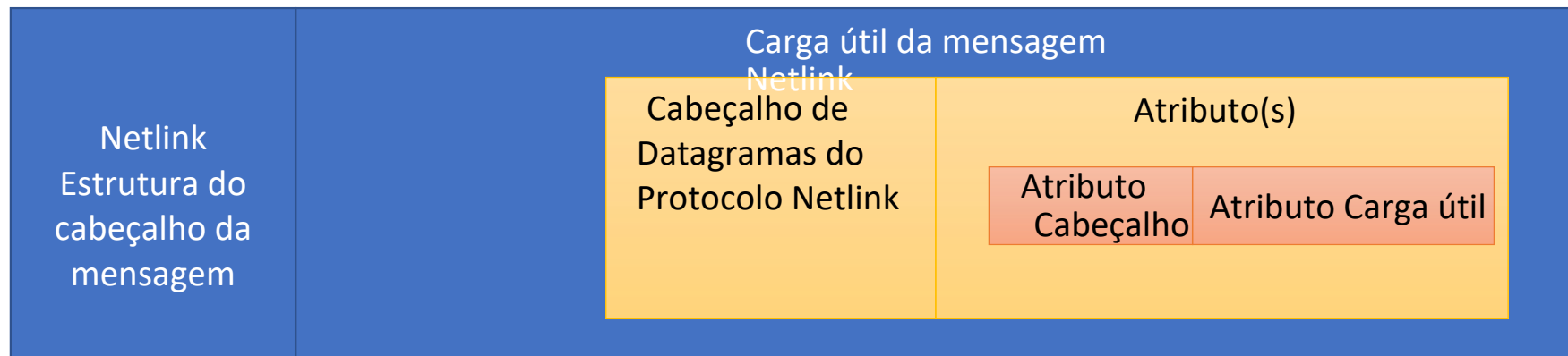
Criar uma mensagem

- Precisamos de alocar uma mensagem `nl_message` estrutural
 - Mantém o cabeçalho da mensagem e a carga útil
- `nlmsg_alloc()` é a função utilizada para atribuição de mensagens
 - Utiliza o tamanho máximo padrão de mensagem (uma página, normalmente 4K)
 - O tamanho por defeito pode ser alterado com `nlmsg_set_default_size(size_t)`
 - `nlmsg_alloc_size()` é uma variante que permite a definição do tamanho no momento da alocação
 - Alternativa #1: Se quisermos reutilizar um cabeçalho já conhecido, podemos chamar `nlmsg_inherit()`
 - Cabeçalho de anexo
 - Alternativa #2: `nlmsg_alloc_simple()` leva um tipo de mensagem e uma bandeira
 - Cria e anexa o cabeçalho

Criar uma mensagem

- Se não criarmos automaticamente um cabeçalho (`nlmsg_alloc_simple()` ou `nlmsg_inherit()`) precisamos de adicionar o cabeçalho nós próprios
- `nlmsg_put()`
 - `nlmsg_type`
 - `nlmsg_flags`
 - `seqnr` (`NL_AUTO_SEQ`)
 - `Porto` (`NL_AUTO_PORTO`)

Não fique confuso!



Exemplo

```
#include <netlink/msg.h>
```

```
structure nlmsg_hdr *hdr; //header
```

```
structure nl_msg *msg; //message
```

```
structure myhdr {
```

```
    uint32_t foo1, foo2;
```

```
hdr = { 10, 20 }; //meu cabeçalho
```

Exemplo (contd)

```
msg = nlmsg_alloc(); //alocar mensagem de tamanho máximo padrão
```

```
// Criar cabeçalho com preenchimento automático da porta e número de  
sequência
```

```
//Note que o espaço para hdr está reservado na mensagem
```

```
hdr = nlmsg_put(msg, NL_AUTO_PORT, NL_AUTO_SEQ, MY_MSGTYPE,  
sizeof(hdr), NLM_F_CREATE);
```

```
//copy customised header into payload
```

```
memcpy(nlmsg_data(hdr), &hdr,
```

Exemplo
`sizeof(hdl);`
(conta)

E quanto à carga útil?

- A carga útil deve ser codificada como atributos netlink sempre que possível
- Os atributos são alinhados em múltiplos de 4bytes
 - `nla_padlen()` retorna o número de bytes de acolchoamento necessários
- Codificação de Atributos
 - Comprimento (16bit)
 - Tipo (16bit)
 - Carga útil

Atributos de análise

- Você pode obtê-los como um array, quando você analisar a mensagem com `nlmsg_parse()`
- Também o podemos fazer manualmente.
 - Navegue pelos atributos individuais usando `nla_next()` após obter um ponteiro para o primeiro usando `nlmsg_attrdata()`
 - `nla_ok()` determina se outro atributo cabe no número restante de bytes ou não

Atributos de Parsing

- Cada atributo tem um cabeçalho e uma carga útil
- Acesse o cabeçalho: `nla_len()`
- Acesse o tipo: `nla_type()`
- Acesso à carga útil: `nla_data()`
 - Evite fundir em algo maior que 4bytes devido ao alinhamento obrigatório
- Alternativa mais fácil: `nla_parse()`
 - Itera todos os atributos no fluxo
 - Valida cada atributo
 - Quando a validação é bem sucedida, armazena apontadores para atributos em array

Validação de Atributos

- Precisamos de assegurar que os atributos são formatados da forma correcta.
- Nós definimos estruturas que compõem as políticas
 - Indique qual é a estrutura do cabeçalho do atributo

estruturar nla_policy {

uint16_t tipo;//NLA_U32/16/8/4

uint16_t

minlen;//comprimento mínimo de carga útil

uint16_t maxlen; }; //comprimento máximo de

Validação de Atributos

carregamento
útil

S

Exemplo de análise de atributos

//headers são ignorados neste

exemplo #include <netlink/msg.h>

#include <netlink/attr.h>

enum {

MY_ATTR_FOO = 1, MY_ATTR_BAR, __MY_ATTR_MAX,};

#define MY_ATTR_MAX (__MY_ATTR_MAX - 1)

Exemplo de análise de atributos (contd)

estrutura estática nla_policy my_policy[MY_ATTR_MAX+1] = {

 //Política de validação

 [MY_ATTR_FOO] = { .type = NLA_U32 },

 [MY_ATTR_BAR] = { .type =

 NLA_STRING,

 .maxlen = 16 },

Exemplo de análise de atributos (;contd)

Exemplo de análise de atributos (contd)

```
void parse_msg(struct nlmsgghdr *nlh)
```

```
    struct nlattr *attrs[MY_ATTR_MAX+1];
```

```
    if (nlmsg_parse(nlh, 0, attrs, MY_ATTR_MAX, my_policy) < 0) /* error */
```

```
    if (attrs[MY_ATTR_FOO]) {
```

```
        /* MY_ATTR_FOO está presente na mensagem */
```

```
        printf("valor: %u\n", nla_get_u32(attrs[MY_ATTR_FOO]));
```

```
    } }
```


Eu quero encontrar um único atributo!

- Há funções que iteram sobre todos os atributos, procuram um correspondente e retornam um ponteiro ao seu cabeçalho
- `nla_find()`;
- `nlmsg_find_attr`

Atributos aninhados

- Atributos incluídos como carga útil de atributos de contentores
 - Tipo: NLA_NESTED
- Atributos podem ser armazenados dentro de uma estrutura em árvore
- É a forma comum de transmitir listas de objetos

Atributos de Parsing Nested

- `nla_parse_nested()`
 - Idêntico a `nla_parse()`, mas
 - Utiliza uma estrutura `nlattr` como argumento
 - Utiliza a carga útil como fluxo de atributos

Construindo Atributos Aninhados

- Podemos aninhar atributos ao cercá-los com
 - `nla_nest_start()`
 - Adicionar cabeçalho de atributo, sem carga útil
 - Todos os dados adicionados a partir deste ponto, serão parte do contentor
 - `nla_nest_end()`
 - Fecha o atributo do recipiente

Criar Mensagem Netlink com Atributos

// Vamos colocar este comportamento dentro de uma função
//So tudo o que você vai ver a seguir será colocado dentro desta função

```
struct nl_msg *build_msg(int ifindex, struct nl_addr *lladdr, int mtu)
{
    //Código a partir dos próximos slides estarão aqui
}
```

Criar Mensagem Netlink com Atributos (cont)

```
struct nl_msg *msg;
```

```
struct nlattr *info,
```

```
*vlan; struct ifinfomsg
```

```
ifi = {
```

```
    .ifi_family = AF_INET,
```

```
    .ifi_index = ifindex,
```

Estrutura predefinida sem ligação à rede:

```
struct ifinfomsg {  
    unsigned char ifi_family; /* AF_UNSPEC */  
    unsigned short ifi_type; /* Device type */  
    int ifi_index; /* Interface index */ unsigned  
    int ifi_flags; /* Device flags */ unsigned  
    int ifi_change; /* change mask */ };
```

Criar Mensagem Netlink com Atributos (cont)

Criar Mensagem Netlink com Atributos (cont)

/* Atribuir uma mensagem netlink de tamanho padrão */

//ROTEIRO REDE FAMILIAR

se (!(msg = nlmsg_alloc_simple(RTM_SETLINK, 0)))

retornar NULL;

Criar Mensagem Netlink com Atributos (cont)

`/* Anexar o cabeçalho específico do protocolo (struct ifinfomsg)*/`

`if (nlmsg_append(msg, &ifi, sizeof(ifi), NLMSG_ALIGNTO) < 0)`

`goto nla_put_failure;`

Criar Mensagem Netlink com Atributos (cont)

`/* Anexar um atributo inteiro de 32 bits para carregar o MTU */`

```
NLA_PUT_U32(msg, IFLA_MTU, mtu);
```

`/* Anexar um atributo não específico para carregar o endereço da`

`camada de ligação */ NLA_PUT_ADDR(msg, IFLA_ADDRESS, lladdr);`

Criar Mensagem Netlink com Atributos (cont)

```
/* Anexar um recipiente para atributos aninhados para transportar
```

```
informação de link */ se (!(info = nla_nest_start(msg,
```

```
IFLA_LINKINFO)))
```

```
goto nla_put_failure;
```

```
/* Coloque um atributo string no recipiente */
```

Criar Mensagem Netlink com Atributos

(cont) `NLA_PUT_STRING(msg, IFLA_INFO_KIND, "vlan");`

Criar Mensagem Netlink com Atributos (cont)

```
/* Anexar outro recipiente dentro do recipiente aberto para transportar  
* atributos específicos da vlan */
```

```
if (!(vlan = nla_nest_start(msg, IFLA_INFO_DATA)))
```

```
goto nla_put_failure;
```

```
/* adicionar atributos de informação específicos da vlan aqui... */
```

Criar Mensagem Netlink com Atributos (cont)

/* Termine de aninhar os atributos de vlan e feche o segundo recipiente. */

```
nla_nest_end(msg, vlan);
```

/* Termine de aninhar o atributo de informação do link e feche o primeiro recipiente.

```
*/ nla_nest_end(msg, info);
```

```
devolver msg;
```

Criar Mensagem Netlink com Atributos (cont)

```
//código alias para  
falhas: nla_put_failure:  
    nlmsg_free(msg);  
    retornar NULL;  
  
}
```

Parsing a Netlink Message com Atributos

// Vamos colocar este comportamento dentro de uma função
// So tudo o que você vai ver a seguir será colocado dentro desta função

```
int parse_message(struct nlmsghdr *hdr)
{
    // O código dos próximos slides estará aqui
}
```


Parsing a Netlink Message com Atributos

```
/* A política define dois atributos: um inteiro de 32 bit e um  
recipiente para atributos aninhados. */
```

```
struct nla_policy attr_policy[] = {  
    [ATTR_FOO] = { .type = NLA_U32 },  
    [ATTR_BAR] = { .type = NLA_NESTED }    ,};
```

```
structure nlattnr  
*attrs[ATTR_MAX+1]; int err;
```

Parsing a Netlink Message com Atributos

```
/* A função nlmsg_parse() irá garantir que a mensagem contenha  
carga útil suficiente para manter o cabeçalho (struct my_hdr), valida  
quaisquer atributos anexados às mensagens e armazena um ponteiro  
para cada atributo no array attrs[] acessível por tipo de  
atributo. */
```

```
if ((err = nlmsg_parse(hdr, sizeof(struct my_hdr), attrs, ATTR_MAX,  
attr_policy)) < 0)  
    goto errout;
```

```
if (attrs[ATTR_FOO]) {
```

```
    /*É seguro acessar diretamente a carga útil do atributo sem  
    qualquer outra verificação desde que nlmsg_parse() aplicou a política.  
    */
```

```
    uint32_t foo = nla_get_u32(attrs[ATTR_FOO]);
```

```
}
```

```
if (attrs[ATTR_BAR]) {  
    structure *nested[NESTED_MAX+1];  
    /* Os atributos aninhados em um recipiente podem ser analisados da mesma forma que os  
  
    atributos de nível superior. */ err = nla_parse_nested(aninhado, NESTED_MAX,  
  
    attrs[ATTR_BAR], nested_policy);  
  
    se (err < 0)  
        goto errout;  
  
    // Atributos do processo aninhados aqui.
```

}

Mais detalhes

- Guia do Desenvolvedor da Biblioteca Central
 - <https://www.infradead.org/~tgr/libnl/doc/core.html>
- Referência API
 - https://www.infradead.org/~tgr/libnl/doc/api/group_core.html

Biblioteca Genery Netlink

Libn-genl

- Uma das desvantagens do protocolo Netlink é que o número de famílias de protocolos é limitado a 32 (MAX_LINKS).
- Por este motivo, foi criada a "Família Netlink Genérica".
 - Fornece apoio para adicionar um maior número de famílias
 - Atua como um multiplexador Netlink
 - Trabalha com uma única família Netlink: NETLINK_GENERIC
 - O protocolo genérico Netlink é baseado no protocolo Netlink e utiliza o seu API

Biblioteca Netlink Genérica - Controller API

- O controlador é um componente no kernel que resolve os nomes das famílias Netlink genéricas para os seus identificadores numéricos. Este módulo fornece funções para consultar o controlador para acessar a funcionalidade de resolução.
- Exemplo:
 - `expectedId = genl_ctrl_resolve(sk, "nl80211");`

Biblioteca Netlink Genérica - Controller API

- Genlmsg_put adiciona cabeçalhos Genéricos Netlink às mensagens Netlink
 - `genlmsg_put(msg, 0, 0, driver_id, 0, NLM_F_DUMP, NL80211_CMD_GET_SCAN, 0);` // Configurar qual comando a executar.

Biblioteca Netlink Genérica - Controller API

- Registo familiar e de comando
- Registra a definição da família Netlink genérica especificada juntamente com todos os comandos associados. Após o registo, as mensagens Netlink genéricas recebidas podem ser passadas para [genl handle msg\(\)](#) que validará as mensagens, procurará um comando correspondente e chamará a respectiva função de callback automaticamente.

Como posso usar a lib-nl e a lib-genl

- Você precisa instalar a libs
 - libnl-3-dev
 - libnl-genl-3-dev
- O código que você faz terá que incluir cabeçalhos relacionados
 - #incluir "netlink/netlink.h".
 - #incluir "netlink/genl/genl.h".
 - #incluir "netlink/genl/ctrl.h".

Para compilar

- `gcc exemplo.c -o exemplo -I/usr/include/libnl3/ -lnl-genl-3 -lnl-3`
- O `-I` detém a localização dos ficheiros de cabeçalho `Include`, que são referenciados na directiva `#include`. Por exemplo `-I/home/mynome/include` irá procurar por um ficheiro de cabeçalho em `"/home/mynome/include"`.
- Como no Unix/Linux os nomes das bibliotecas dos sistemas operacionais começam com `"lib"` (por exemplo, `"libkuku.a"`), então o prefixo `"lib"` é omitido do parâmetro `-l`: `"-lkuku"`.
- (Não usado aqui, mas o `-L` contém a localização dos arquivos da biblioteca, e `-l` contém o nome de um arquivo da biblioteca, que deve ser incluído na

Para
fase Link da construção do programa).

compilar

O que tem isto a ver com Wi-Fi?

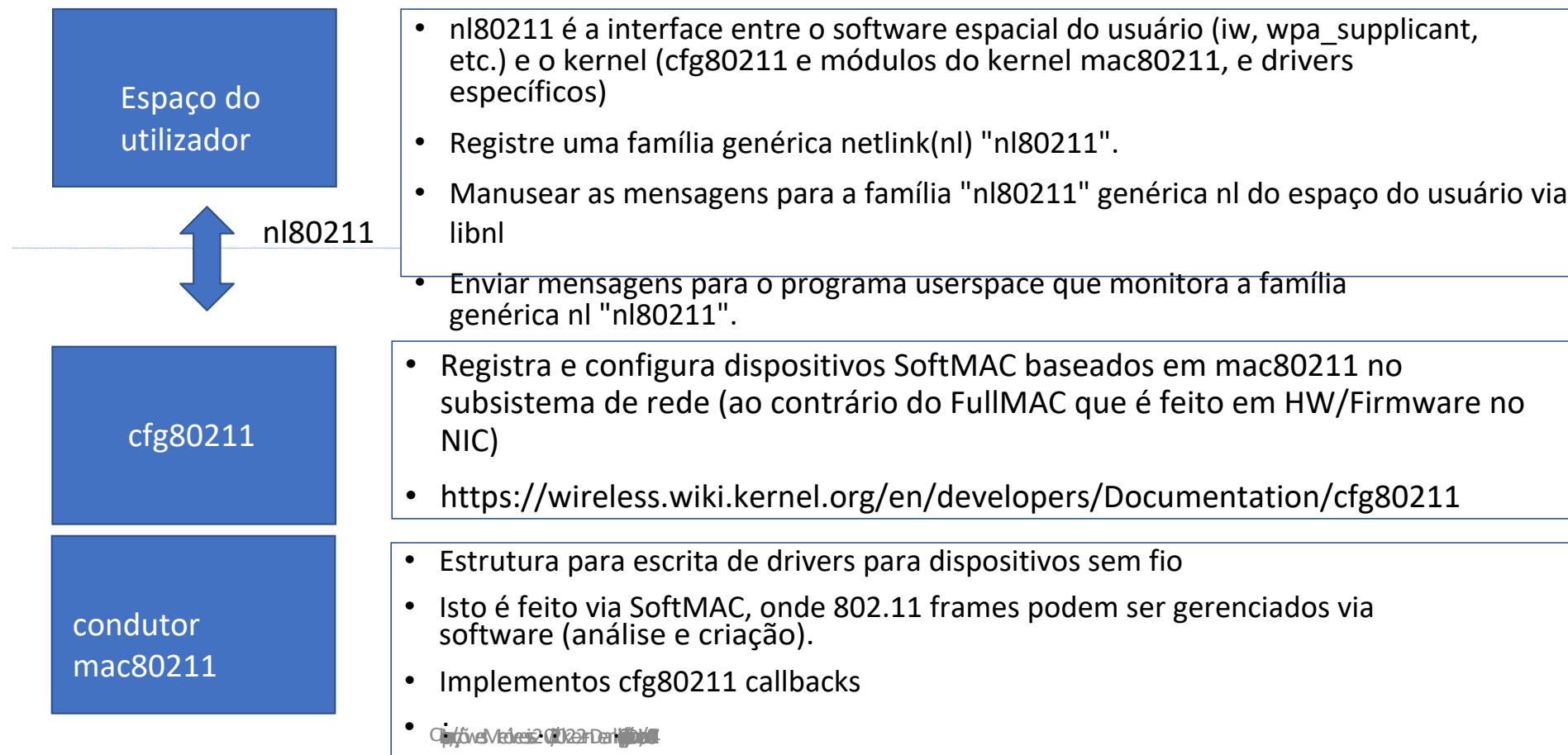
- Até agora, temos abordado como funciona a biblioteca netlink
- Vamos agora tratar de uma família relacionada com Wi-Fi...

NL80211

NL80211

- 802.11 interface userspace para Netlink
- Permite o controle de drivers wireless a partir do espaço do usuário, usando Comunicações IPC entre o kernel e o espaço do usuário
- Antes:
 - Chamadas de sistema de controle de entrada/saída (ioctl)
 - Extensões sem fio API genéricas

NL80211



Posso usar isto?

- Só funciona em interfaces de rede cujos drivers são compatíveis com a Netlink.
- Teste isso executando a `lista de Au`.
- Alguns dos comandos utilizados no código requerem privilégios de 'root'.
- Os arquivos de cabeçalho da biblioteca já vêm instalados com o kernel

Exemplos de aplicações baseadas no NL80211

- iw
 - Dispositivos sem fio Ferramenta de configuração CLI
- crda
 - Ajudante para conformidade com os domínios regulamentares utilizando a gestão de dispositivos udev
 - Código do país, canal de calibração, largura de banda de calibração
- hostapd
 - Ponto de acesso e daemon de autenticação
- wpa_supplicant (-Dnl80211)
- iwd
 - Serviço de gestão sem fios
- NetworkManager
 - Gerenciador de conexões baseado em iwd
- ConnMan
 - Outro Gestor de Conexões (CLI)

Papel da nl80211.h

- Fornece vários enumeros relacionados com o 802.11
- Em seguida, utilize construções genéricas nl80211 (isto é, tomadas, mensagens, atributos, comandos) para enviar comandos para o cartão sem fio

Estrutura genérica de um programa nl80211

1 - Alocar novo socket netlink na
estrutura de memória nl_sock* sk =
nl_socket_alloc();

2 - Criar um descritor de arquivo e criar o
socket genl_connect(sk);

3 - Encontre a identificação do condutor nl80211.
int driver_id = genl_ctrl_resolve(socket, "nl80211");

Estrutura genérica de um programa nl80211

- Daqui, você pode fazer o que quiser.
- Tipicamente:
 - Criar uma função de callback e ligá-la à tomada
 - Criar uma mensagem com um comando a ser enviado para o cartão wifi
 - Cabeçalho + atributo(s)
 - Envie a mensagem e aguarde a resposta
 - Pode envolver a adesão a algum grupo multicast
 - Quando a resposta chega, a sua função de retorno é chamada
 - Dentro dessa função, você constrói um código que processa a carga útil e os atributos da mensagem (ou seja, imprime informações na tela).

Criar uma função de retorno de chamada e associá-la ao

- `nl_socket_modify_cb(sk, NL_CB_VALID, NL_CB_CUSTOM, yourFunctionName, NULL);`

Criar uma mensagem com um comando a ~~para~~ para envio wifi

```
//alocar uma mensagem
struct nl_msg* msg = nlmsg_alloc();

// configuração da interface de pedido
enum nl80211_commands* cmd = NL80211_CMD_GET_INTERFACE;
int ifIndex = if_nametoindex("wlan0");
int flags = 0;

// configurar a mensagem
genlmsg_put(msg, 0, 0, driver_id, 0, flags, cmd, 0);

//atributos de mensagem adicta
NLA_PUT_U32(msg, NL80211_ATTR_IFINDEX, ifIndex);
```

Envie a mensagem e aguarde a resposta

//enviar a mensagem (isto liberta-o)

```
ret = nl_send_auto_complete(sk, msg);
```

//block para mensagem de

retorno

```
nl_recvmsgs_default(sk);
```

Função chamada de retorno

```
static int nlCallback(struct nl_msg* msg, void* arg) {
    struct nlmsg_hdr* ret_hdr = nlmsg_hdr(msg);
    struct nlattr *tb_msg[NL80211_ATTR_MAX + 1];

    if (ret_hdr->nlmsg_type != expectedId)return NL_STOP;

    struct genlmsg_hdr *gnlh = (struct genlmsg_hdr*) nlmsg_data(ret_hdr);
    nla_parse(tb_msg, NL80211_ATTR_MAX, genlmsg_attrdata(gnlh, 0), genlmsg_attrlen(gnlh, 0), NULL);

    if (tb_msg[NL80211_ATTR_IFTYPE]) {
        int type = nla_get_u32(tb_msg[NL80211_ATTR_IFTYPE]);
        printf("Type: %d", type);
    }
}
```

Juntar-se a um grupo multicast

```
int mcid = nl_get_multicast_id(socket, "nl80211", "scan");  
nl_socket_add_membership(socket, mcid);
```

Executando uma varredura

- Use o comando "NL80211_CMD_TRIGGER_SCAN" para iniciar uma varredura
 - Se você tentar iniciar outro quando um está correndo, isso falharia.
- Ouça para que a varredura seja concluída quando você receber um NL80211_CMD_NEW_SCAN_RESULTS
- Você pode então enviar um comando NL80211_CMD_GET_SCAN para pedir os resultados
 - Você receberá uma mensagem de volta para cada estação encontrada, então esteja pronto para lidar com várias mensagens.

Exemplos de comandos

- Explicação detalhada: nl80211.h
- Comandos
 - NL80211_CMD_GET_INTERFACE
 - Solicitar a configuração de uma interface
 - Para todas as interfaces ou uma específica
 - Para uma única interface é necessário enviar a solicitação com o atributo NL80211_ATTR_IFINDEX
 - NL80211_CMD_SET_BEACON
 - Alterar o farol na interface de um ponto de acesso usando atributos
 - NL80211_CMD_GET_STATION
 - Obter atributos para estação identificada pelo NL80211_ATTR_MAC na interface identificada pelo NL80211_ATTR_IFINDEX
 - NL80211_CMD_GET_REG
 - Obter informações do domínio regulatório

Mais informações

- Infelizmente, não existe um guia do desenvolvedor para o nl80211
- A melhor fonte de informação (devidamente codificada), é olhar para as implementações de referência (iw)
- Alguns fragmentos de código simplifier podem ser encontrados em projetos individuais online
 - <https://github.com/Robpol86/libnl>