SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72886

IMPLEMENTÁCIA KONTROLY IPC DO SYSTÉMU MEDUSA

DIPLOMOVÁ PRÁCA

2018 **Viliam Mihálik**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5384-72886

IMPLEMENTÁCIA KONTROLY IPC DO SYSTÉMU MEDUSA

DIPLOMOVÁ PRÁCA

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 Aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Vedúci záverečnej práce:	Mgr. Ing. Matúš Jókay, PhD.

Bratislava 2018 **Viliam Mihálik**

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Viliam Mihálik
Diplomová práca:	Implementácia kontroly IPC do systému Medusa
Vedúci záverečnej práce:	Mgr. Ing. Matúš Jókay, PhD.
Miesto a rok predloženia práce:	Bratislava 2018

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean et est a dui semper facilisis. Pellentesque placerat elit a nunc. Nullam tortor odio, rutrum quis, egestas ut, posuere sed, felis. Vestibulum placerat feugiat nisl. Suspendisse lacinia, odio non feugiat vestibulum, sem erat blandit metus, ac nonummy magna odio pharetra felis. Vivamus vehicula velit non metus faucibus auctor. Nam sed augue. Donec orci. Cras eget diam et dolor dapibus sollicitudin. In lacinia, tellus vitae laoreet ultrices, lectus ligula dictum dui, eget condimentum velit dui vitae ante. Nulla nonummy augue nec pede. Pellentesque ut nulla. Donec at libero. Pellentesque at nisl ac nisi fermentum viverra. Praesent odio. Phasellus tincidunt diam ut ipsum. Donec eget est. A skúška mäkčeňov a dĺžnov.

Kľúčové slová: Medusa, IPC, Linux, LSM

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Viliam Mihálik
Master's thesis:	Implement IPC control on Medusa system
Supervisor:	Mgr. Ing. Matúš Jókay, PhD.
Place and year of submission:	Bratislava 2018

On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the charms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to ensue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying through shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our power of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to be welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of business it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore always holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or else he endures pains to avoid worse pains.

Keywords: Medusa, IPC, Linux, LSM

Poďakovanie

I would like to express a gratitude to my thesis supervisor.

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

CDMA	Code Division Multiple Access
FIFO	First in, First out
GSM	Global System for Mobile communication
HW	Halo Wars
IPC	Inter-Process Communication
POSIX	Portable Operating System Interface
RCU	read-copy-update
SW	Star Wars

Zoznam algoritmov

Zoznam výpisov

Úvod

Tu bude krasny uvod s diakritikou atd.
A mozno aj viac riadkovy uvod.

1 IPC

Inter-Process Communication (IPC) predstavuje súbor mechanizmov určených na komunikáciu a správu dát medzi viacerými procesmi. Operačný systém Linux obsahuje niekoľko takýchto mechanizmov, medzi najhlavnejšie patria:

- Signály
- Rúry
- Fronty správ
- Semafory
- Zdieľaná pamäť
- Sokety

V nasledujúcich odsekoch si bližšie popíšeme tieto mechanizmy.

1.1 Signály

Ide o jeden z najstarších IPC mechanizmov používaný v Unix systémoch. Signál je asynchrónne upozornenie zaslané procesu alebo konkrétnemu vláknu v rámci toho istého procesu, za účelom upozornenia na udalosť, ktorá sa vyskytla. V momente keď sa signál odošle, operačný systém preruší vykonávanie procesu, ktorý má byť signalizovaný a v tomto procese sa vykoná obslúženie signálu.

Je potrebné si uvedomiť že **signály nie sú** to isté ako **prerušenia**. Rozdiel medzi signálom a prerušením je, že prerušenie je vyvolané procesorom a signál je vyvolaný z jadra systému. Signál je možné vyvolať systémovým volaním **kill**. Toto systémové volanie má dva parametre:

- *pid* - identifikátor procesu, ktorý má byť signalizovaný
- *sig* - typ signálu

Podporované typy je možné zistiť pomocou príkazu *kill -l* alebo v súbore */include/linux/signal.h*.

V prípade že je definovaná obslužná funkcia, táto funkcia sa vykoná, v opačnom prípade je použitá štandardná obsluha signálu. Obslužnú funkciu je možné definovať pomocou funkcie **signal**, avšak správanie tejto funkcie môže byť rozdielne vzhľadom na

platformu. Preto sa odporúča používať funkciu **sigaction**, ktorá bola definovaná v štandarde **POSIX.1**. Toto systémové volanie má parametere:

- *signum* - typ signálu, ktorý chceme obslúžiť
- *act* - definuje akciu, ktorá sa má vykonať pri obsluhe signálu
- *oldact* - definuje starú obsluhu signálu

Signály je možné použiť pre komunikáciu ako aj synchronizáciu avšak ide o veľmi slabý nástroj pre tieto potreby.[1]

1.2 Rúry

Rúry predstavujú jednosmerný tok dát medzi procesmi: všetky dáta zapísané procesom do rúry sú jadrom presmerované do iného procesu, ktorý z nej môže čítať. Poznáme 2 druhy rúr:

- Anonymné rúry - žiadny objekt v súborovom strome
- Pomenované rúry - objekt v súborovom strome

1.2.1 Anonymné rúry

Anonymné rúry je možné vytvoriť pomocou systémového volania **pipe**, alebo taktiež pomocou znaku | vo väčšine *Unix* príkazových riadkoch.Systémové volanie **pipe** obsahuje jeden parameter, ktorým je pole *pipefd* o veľkosti 2. Toto pole obsahuje po návrate z funkcie súborové deskriptory. Tieto dva súborové deskriptory predstavujú konce rúry, *pipefd[0]* je čítací koniec rúry a *pipefd[1]* je zapisovací koniec rúry. Tieto súborové deskriptory je následne možné použiť na zapisovanie a čítanie pomocou systémových volaní *write*[2] a *read*[3]. Tieto operácie sú blokujúce v dvoch prípadoch:

- write - rúra je plná
- read - rúra je prázdna

Niektoré *Unix* systémy ako napríklad *System V Release 4*, implementuje **full-duplex** rúry, teda rúry, pri ktorých oba konce rúry(súborové deskriptory) je možné použiť ako na zapisovanie tak aj na čítanie. Avšak štandard **POSIX** definuje iba **half-duplex** rúry, pričom každý proces musí zatvoriť jeden deskriptor pred použitím druhého.

[1] `http://man7.org/conf/lca2013/IPC_Overview-LCA-2013-printable.pdf`
[2] `http://man7.org/linux/man-pages/man2/write.2.html`
[3] `http://man7.org/linux/man-pages/man2/read.2.html`

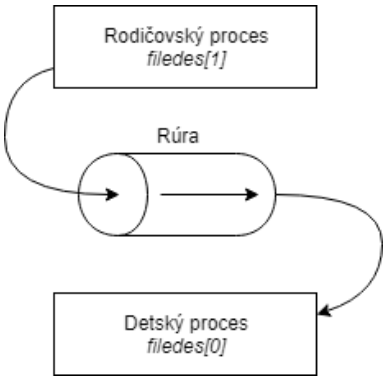
Takto vytvorená rúra umožňuje komunikáciu medzi rodičovským procesom a jeho potomkami. Avšak je potrebné zabezpečiť aby procesy, ktoré medzi sebou chcú komunikovať zdieľali rovnaké súborové deskriptory. Toto je možné jednoducho dosiahnuť tak, že sa rúra vytvorí pred vytvorením detského procesu(*fork*). Zjednodušený pseudokód môžeme vidieť tu 1 spolu s ilustračný obrázkom 1.

```
int filedes[2];

pipe(filedes);

child_pid = fork();
if (child_pid == 0) {
       close(filedes[1]);
} else {
       close(filedes[1]);
}
```

Listing 1: Použite anonymných rúr



Obr. 1: Tok dát medzi procesmi.

Nevýhodou tohto systémového volania je absencia v súborovom strome a teda nie je možné využiť tento IPC mechanizmus na komunikáciu medzi ľubovoľnými dvoma alebo viacerými procesmi.

1.2.2 Pomenované rúry

Pomenované rúry taktiež nazývané aj **FIFO** na rozdiel od anonymných rúr máju meno v súborovom systéme. Jednou z možností ako vytvoriť tento typ rúry je pomocou funkcie **mkfifo**, ktorá je definovaná v štandarde POSIX. Táto funkcia v svojej implementácií volá systémové volanie *mknod* s príznakom, ktorý definuje že ide o pomenovanú rúru. Funkcia **mkfifo** má 2 parametre:

- *pathname* - názov rúry
- *mode* - práva súboru

Návratová hodnota funkcie je 0 v prípade úspechu a v prípade chyby je návratová hodnota -1. Takto vytvorenú rúru je možné rovnako ako anonymné rúry obsluhovať pomocou systémových volaní *read* a *write*.

1.3 Fronty správ

Fronty správ si najlepšie môžme predstaviť ako zreťazený zoznam v adresnom priestore jadra. Správy môžu byť odosielané do fronty v poradí a následne načítané z fronty pomocou niekoľko rôznych spôsobov. Každá fronta správ je jednoznačne identifikovaná pomocou IPC identifikátora. Pre lepšie pochopenie tohto konceptu si popíšeme 3 hlavné štruktúry, ktoré zabezpečujú fungovanie fronty správ v jadre Linuxu.

1.3.1 msgbuf

Táto štruktúra (definovaná v `linux/msg.h`) predstavuje predlohu ako by mala vyzerať správa, ktorú budeme posielať. Táto predloha obsahuje dve položky:

- long **mtype** - umožňuje určiť o aký typ správy ide, napríklad *chybová správa*, *normálna správa* a podobne, možností je nekonečno
- char **mtext**[1] - samotné dáta správy, túto položku je možné ľubovoľne rozšíriť, ako napríklad v ukážke kódu 2, avšak s limitom, ktorý je maximálna dĺžka správy ***MSGMAX***=8192[4]

```
struct message {
    long type;
    struct my_special_struct data;
} msg;
```

Listing 2: Príklad vlastnej štruktúry správy

1.3.2 msg_msg a msg_msgseg

Každá správa je rozdelená na stránky, ktoré sú dynamicky alokované v pamäti. Veľkosť tejto stránky je závislí od architektúry a zisťuje sa nasledovne *sysconf(_SC_PAGESIZE)*. Štruktúra ***msg_msg***, ktorú môžeme vidieť tu 3, predstavuje hlavičku každej správy pričom sa inštancie tejto štruktúry nachádzajú v zreťazenom zozname, ktorý je definovaný položkou *m_list*. V prípade že dĺžka správy je menšia ako

$$\text{PAGE\_SIZE} - \text{sizeof(struct msg\_msg)}$$

[4]Táto veľkosť je definovaná v `linux/msg.h` a môže sa líšiť od verzie jadra. Túto hodnotu je taktiež možné zistiť pomocou príkazu *ipcs -l*.

celý obsah správy sa nachádza v pamäťovej oblasti za štruktúrou *msg_msg*. V opačnom prípade sa prvá časť správy nachádza na rovnakom mieste a ostatné stránky sa nachádzajú v pamäťovej oblasti, na ktorú ukazuje ukazovateľ *next*. Tento ukazovateľ odkazuje na štruktúru *msg_msgseg* pozri 5, ktorá obsahuje ukazovateľ *next* na ďalšiu stránku, pričom v pamäťovej oblasti za touto štruktúrou sa nachádzajú dáta aktuálnej stránky.

```
struct msg_msg {
       struct list_head m_list;
       long m_type;
       size_t m_ts;
       struct msg_msgseg *next;
       void *security;
       /* obsah prvej stránky */
};
```

Listing 3: Štruktúra msg_msg

```
struct msg_msgseg {
       struct msg_msgseg *next;
       /* obsah stránky */
};
```

Listing 4: Štruktúra msg_msgseg

1.3.3 msg_queue

Poslednou štruktúrou, ktorá sa nachádza najvyššie v hierarchií týchto štruktúr je **msq_queue** a jej deklaráciu môžete vidieť na ukážke kódu 5. Najdôležitejšou položkou tejto štruktúry je položka`q_messages`, ktorá predstavuje prvý element zreťazeného zoznamu všetkých správ vo fronte. Ďalšie zreťazené zoznamy, ktoré sa tu nachádzajú sú `q_receivers` a `q_senders`, ktoré obsahujú zreťazené zoznamy procesov, ktoré posielajú správy a procesov, ktoré prijímajú správy. Zaujímavou položkou z pohľadu bezpečnosti je položka `q_perm`, ktorá obsahuje inštanciu štruktúry **`kern_ipc_perm`**. Túto štruktúru si popíšeme v kapitole 1.3.4.

```
struct msg_queue {
       struct kern_ipc_perm q_perm;
       /* meta dáta */
       struct list_head q_messages;
       struct list_head q_receivers;
       struct list_head q_senders;
} __randomize_layout;
```

Listing 5: Štruktúra msq_queue

1.3.4 kern_ipc_perm

Štruktúra `kern_ipc_perm` sa nenachádza len pri fronte správ ale taktiež aj pri semaforoch a zdieľanej pamäti. Táto štruktúra predstavuje sadu meta dát o konkrétnom IPC

objekte a jej položky spolu s typmi môžeme vidieť na ukážke kódu B.1. Vyznám týchto položiek je nasledovný:

- lock - uzamykací mechanizmus pre ochranu IPC objektu
- deleted - príznak, či bol zdroj uvoľnený
- id
- key - jednoznačný identifikátor, v rámci konkrétneho typu objektu, to znamená, že jedna inštancia semafora, zdieľanej pamäte alebo fronty správ môže mať rovnaký identifikátor
- uid - ID používateľa, ktorý vlastní IPC objekt
- gid - ID skupiny, ktorá vlastní IPC objekt
- cuid - ID používateľa, ktorý vytvoril IPC objekt
- cgid - ID skupiny, ktorá vytvorila IPC objekt
- mode - bitová maska oprávnení
- seq - sekvenčné číslo, používané sa na vytvorenie nového identifikátora pre IPC objekt
- security - ukazovateľ na štruktúru, ktorú vytvára zvolené bezpečnostné riešenie v jadre Linuxu
- khtnode
- rcu - RCU synchronizačný mechanizmus
- refcount - počítadlo použití IPC objektu

1.3.5 Požívanie fronty správ

Na vytvorenie fronty správ sa používa systémové volanie `msgget`. Toto systémové volanie má dva parametre, ktorými sú `key`(identifikátor objektu) a `msgflg` príznaky IPC objektu. Nová fronta je vytvorená v prípadoch, že:

- `key` sa rovná `IPC_PRIVATE`
- `key` nemá ešte priradený žiadny IPC objekt a `IPC_CREATE` príznak je definovaný v parametre `msglfg`

Ak je definovaný príznak `IPC_EXCL` spolu s `IPC_CREATE` a identifikátor už existuje, volanie funkcie zlyhá s chybovou správou `EEXIST`, ktorá definuje, že IPC objekt už existuje. Naopak v prípade, že `IPC_EXCL` nie je definované tak návratová hodnota je identifikátor už existujúceho IPC objektu.

Na posielanie a prijímanie správ z fronty sa používajú systémové volania `msgsnd` a `msgrcv`. Funkcia `msgsnd` má nasledovné parametre:

- `msgid` identifikátor fronty získaný z funkcie `msgget`
- `msgp` ukazovateľ na štruktúru, ktorú si používateľ definuje sám a mal by vychádzať zo šablóny ktorú sme si popísali v kapitole 1.3.1
- `msgsz` veľkosť dátovej štruktúry, ktorú chceme prenášať
- `msgflg` príznaky, ktoré definujú čo sa má diať v prípade, že je fronta plná

Funkcia `msgrcv`, odstráni správu z frontu a premiestni do pamäte, na ktorý ukazuje parameter funkcie `msgp`. Ďalšie parametre sú:

- `msgsz` maximálnu veľkosť dát v bytoch pre položku `mtext`, štruktúry, na ktorú ukazuje ukazovateľ `msgp`
- `msgflg` príznaky, ktoré definujú čo sa má diať v prípade, že je fronta prázdna
- `msgtyp` číslo, ktoré definuje ktorý typ správy bude ako prvý vybraný z fronty(nemôže byt definovaný príznak `MSG_COPY`), v prípade že je definovaný príznak `MSG_EXCEPT`, tak sa z fronty vyberá prvá správa z typom odlišným od `msgtyp`

Posledným systémovým volaním tohto mechanizmu je `msgctl`, ktoré vykonáva kontrolné operácie nad objektom. Funkcia má 3 parametre:

- `msgid` identifikátor objektu
- `cmd` typ operácie
- `buf` ide o štruktúru, ktorá sa v jadre ako aj v manuálových stránkach nachádza v 32 bitovej verzií(viď **??**), avšak v jadre sa označuje za zastaralú pričom ju nahrádza 64 bitová verzia(viď **??**), táto štruktúra slúži na prenos meta dát z jadra systému do užívateľského priestoru

Typy operácie nad frontami správ sú nasledovné:

- `IPC_STAT` a `MSG_STAT` kopíruje informácie z jadra do štruktúry na ktorú ukazuje ukazovateľ `buf`
- `IPC_SET` nastavuje položky jadra na základe ukazovateľa `buf`, konkrétne `msg_perm.uid`, `msg_perm.gid`, `msg_qbytes` a posledných 9 bitov `msg_perm.mode`
- `IPC_RMID` odstraňuje frontu správ
- `IPC_INFO` a `MSG_INFO` získava limity a systémové nastavenia pre fronty správ, dáta sa nachádzajú na adrese ukazovateľa `buf`, avšak dáta sú v štruktúre typu `msginfo`(viď **??**) a preto je potrebné pre-typovanie[5]

1.4 Semafory

Semafor ako IPC mechanizmus nepredstavuje nástoj na prenášanie dát ale slúži ako synchronizačný mechanizmus na ochranu zdieľaných zdrojov pri viac procesovom alebo viac vláknovom vykonávaný programu. Všeobecný semafor si môžme predstaviť ako počítadlo, ktoré je možné atomický upravovať. Semafor zvyčajne implementuje dve základné funkcie, ktoré slúžia na zvýšenie(`signal`) a zníženie(`wait`) tohto počítadla. Napríklad ak proces 1 chce vstúpiť do chránenej oblasti(pristúpiť k zdieľaným zdrojom) zníži počítadlo semaforu. Proces 2, ktorý taktiež bude chcieť pristúpiť k týmto zdrojom zníži semafor čo má za následok bloknutie procesu 2, ktorý musí počkať na zvýšenie počítadlo. Proces 1 v prípade že bude opúšťať kritickú oblasť zvýši počítadlo semaforu čo zabezpečí odblokovanie procesu 1.

Semafor *System V* semafory na rozdiel od POSIX semaforov nepredstavujú len jedno počítadlo ale skupiny počítadiel. Každé jedno počítadlo môže chrániť nejakú kritickú oblasť a teda jeden *System V* semafor môže chrániť viacero kritických oblastí. Veľkou výhodou je taktiež schopnosť navrátiť operácie vykonané na semafore v prípade, že proces, ktorý bol v kritickej oblasti neočakávane skončí a zabezpečiť tak aby čakajúci proces mohol vstúpiť do kritickej oblasti.

V nasledujúcich odsekoch si popíšeme interné štruktúry v jadre Linuxu, ktoré zabezpečujú fungovanie semaforov a taktiež aj použitie tohto IPC mechanizmu.

1.4.1 Štruktúry *System V* semaforov

Základná štruktúra, ktorá v sebe nesie hodnotu jedného počítadla má nasledovné položky:

[5] `IPC_INFO` a `MSG_INFO` ako aj `IPC_STAT` a `MSG_STAT` vracajú mierne odlišné dáta a sú platformovo závislé pre viac info pozri `http://man7.org/linux/man-pages/man2/msgctl.2.html`

- `semval` hodnota počítadla
- `semval` PID procesu, ktorý posledný modifikoval semafor
- `lock` uzamykací mechanizmus pre ochranu počítadla
- `pending_alter` a `pending_const` operácie, ktoré čakajú na vykonanie
- `sem_otime` čas posledného volania funkcie `sem_op` nad počítadlom

Nadradenou štruktúrou, ktorá predstavuje skupiny počítadiel je `sem_array`, ktorá má nasledovné položky:

- `sem_perm` štruktúra typu `kern_ipc_perm`, ktorú sme si popísali v kapitole 1.3.4
- `sem_ctime` čas posledného volania funkcie `sem_ctl` nad semaforom
- `pending_alter` a `pending_const` operácie, ktoré čakajú na vykonanie
- `list_id` spätné operácie na celú skupinu semaforov, ide o vlastnosť, ktorú sme si popísali v úvode do kapitoly
- `sem_nsems` počet semaforov
- `complex_count` počet komplexných operácii ktoré čakajú na vykonanie
- `use_global_lock` globálny uzamykací mechanizmus nad celou skupinou semaforov
- **`sems`** pole jednotlivých semaforov/počítadiel

Operácie, ktoré sa nad týmito semaformi vykonávajú sú uložené v štruktúre `sem_queue`. Ide o zreťazený zoznam a každá jedna inštancia tejto štruktúry predstavuje operácie jedného procesu, ktorý je blokovaný(spí) na semafore. Táto štruktúra vyzerá nasledovne:

- `list` ďalšie položky zreťazeného zoznamu
- `sleeper` štruktúra typu `task_struct`, teda proces, ktorý spí
- `undo` spätné operácie
- `sops` pole štruktúr typu `sembuf`, ktoré definuje nevykonané operácie
- `blocking` pole štruktúr typu `sembuf`, ktoré definuje operácie ktoré sú blokovacie
- `nsops` počet operácií

- `alter` príznak, ktorý označuje či operácia modifikuje pole semaforov
- `dupsop` TODO príznak, ktorý označuje či operácia modifikuje pole semaforov

Štruktúry na prácu s operáciami, ktoré majú byť v prípade ukončenia programu obnovené do pôvodného stavu sú štruktúry `sem_undo` a `sem_undo_list`. Každý proces má jednu prislúchajúcu štruktúru `sem_undo` a v prípade že je proces ukončený tak sa tieto operácie vykonajú. Štruktúra `sem_undo_list` zabezpečuje zdieľaný prístup k `sem_undo` štruktúram v prípade že viacero procesov zdieľa jeden list čo je možné zabezpečiť pomocou príznaku `CLONE_SYSVSEM` pri vytváraní nového procesu.

1.4.2 Použitie semaforov

System V semafor sa vytvára pomocou systémového volania `semget`, ktoré je analogické k funkcii `msgget`, ktorú sme si popísali v kapitole 1.3.5. JEdiným rozdielom je parameter **`nsems`**, ktorý definuje koľko jednotlivých semaforov chceme vytvoriť. Takto vytvorený semafor je možné používať a vykonávať nad ním operácie pomocou systémového volania `semop`, ktoré má nasledovné parametre:

- `semid` identiifkátor semaforu
- `sops` operácie nad semaforom
- `nsops` počet operácii v poli

Jednotlivé operácie sú definované pomocou štruktúry `sembuf`, ktorá má nasledujúce položky:

- `sem_num` číslo semaforu nad ktorým chcem operáciu vykonať
- `sem_op` typ operácie
- `sem_flg` príznaky operácie

`sem_flg` môže nadobúdať dve hodnoty, ktoré sú `SEM_UNDO` a `IPC_NOWAIT`. `SEM_UNDO` indikuje že chceme aby daná operácia bola obnoviteľná. Príznak `IPC_NOWAIT` priamo súvisí s parametrom `sem_op`, ktorý môže nadobudnúť tieto stavy:

- `sem_op` > 0 hodnota `sem_op` sa pričíta k hodnota počítadla(vyžaduje sa právo na zápis)
- `sem_op` $== 0$ a `semval` $== 0$ tak operácia okamžite prebehne, inak ak je definovaný príznak `IPC_NOWAIT` operácie skončí s chybou ak však tento príznak nieje definovaný proces čaká pokiaľ bude semafor 0(vyžaduje práva na čítanie semaforu)

- `sem_op` < 0 a zároveň `semval` $>=$ `sem_op` tak je operácia vykonaná okamžite, inak analogicky k predošlému prípadu operácia skončí buď s chybou alebo proces čaká pokiaľ bude zvýšená hodnota počítadla(vyžaduje sa právo na zápis)

Poseldným syste

1.5 Zdieľaná pamäť

1.6 Sokety

2 Medusa

Medusa je bezpečnostný systém pre jadro Linux-u.

3 Úkážka glossaries

Verzia FEIstyle 1.5 používa glossary[6] balík. Code Division Multiple Access (CDMA) je dlhá skratka naopak GSM je skratka v krátkej forme.

[6] https://www.ctan.org/pkg/glossaries?lang=en

4 Recitácia

Citujem všetky zdroje v **bibliography.bib**, [**t00**, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15].
Good luck.

5 Možnosti anonymizácie

Anonymizácia znamená zmena alebo úprava údajov tak, aby sa podľa nich nedala jednoznačne určiť osoba, ktorej tieto údaje patria [1]. Existuje niekoľko spôsobov, ktorými môžeme dosiahnuť rôznu úroveň anonymizácie na internete: od mazania cookies súborov po ukončení prehliadania webových stránok až po používanie operačných systémov, ktoré sú na anonymite založené; od bezplatných možností až po komerčné verzie.
Nasleduje priblíženie niektorých možnosti anonymizácie.

5.1 Súkromné prehliadanie

Najpoužívanejšie internetové prehliadače súčasnosti majú v sebe zabudovanú funkcionalitu, ktorá dokáže čiastočne anonymizovať prístup na internet. Táto funkcionalita blokuje ukladanie navštívených stránok do histórie a nezaznamenáva súbory, ktoré sa stiahnu z internetu. SW a Halo Wars sú skratky.

5.2 Anonymná sieť

Anonymná sieť je sieť serverov, medzi ktorými dáta prechádzajú šifrované. V anonymných sieťach dáta prechádzajú z počítača používateľa, odkiaľ bola požiadavka poslaná, cez viaceré proxy smerovače, z ktorých každý správu doplní o smerovanie a zašifruje vlastným kľúčom. Cesta od ...

5.3 Funkcionalita

Rozšírenie tiež okrem splnenia špecifikácie malo pre prehľadnosť a overenie funkčnosti zobrazovať údaje, ktoré boli na server odoslané. Zoznam údajov odoslaných na server, sa mal ukladať do krátkodobej histórie, aby nemal používateľ k dispozícií len najnovšie údaje, ale aj údaje odoslané v nejakom časovom období. Nejaky listing z priloh C.1.

5.3.1 Funkcionalita2

Samozrejmosťou bolo nastavenie zapnutia rozšírenia pri štarte, prípadne interval zmeny odosielaných údajov.

5.4 Vzhľad

Dôležitou požiadavkou kladenou na rozšírenie bolo príjemné používateľské rozhranie. Z tohto dôvodu malo rozšírenie obsahovať zoznam modifikovaných vlastností a tlačidlo pre prístup k nastaveniam rozšírenia v jednoduchej a praktickej forme. Predpokladaný vzhľad je zobrazený na obrázku č. 2. Dôležitou požiadavkou kladenou na rozšírenie bolo príjemné používateľské rozhranie.[**t00**] Z tohto dôvodu malo rozšírenie obsahovať zoznam

Tabuľka 1: Moduly a ich funkcie pri anonymizácii

	Funkcia													
						Modifikácia								
Modul	zobrazenie hlavičky	blokovanie skriptov	zmena IP	zmena lokalizácie	zmazanie/blokovanie cookies	blokovanie trackerov	popis	používateľský agent	kódové označenie prehliadača	názov prehliadača	verzia prehliadača	platforma	výrobca prehliadača	označenie výrobcu prehliadača
User agent switcher							X	X	X	X	X	X	X	X
Ghostery					X	X								
Better privacy					X									
Anonymox			X	X	X		X	X						
Modify headers					X			X						
Request policy						X								
Live HTTP headers	X													
User agent awitcher for chrome							X	X						
Header hacker							X	X	X	X	X	X	X	X
Mod header							X	X	X	X	X	X	X	X
Script no		X												
No script		X												
Proxify it			X	X										
I'm not here				X										
Get anonymous personal edition		X	X	X	X	X								
Anonymous browsing toolbar			X	X										
Easy hide your IP and surf anonymously			X	X				X	X	X	X			

Obr. 2: Predpokladaný vzhľad rozšírenia.

modifikovaných vlastností a tlačidlo pre prístup k nastaveniam rozšírenia v jednoduchej a praktickej forme. Predpokladaný vzhľad je zobrazený na obrázku č. 2.

```
/* Hello World program */

#include<stdio.h>

struct cpu_info {
    long unsigned utime, ntime, stime, itime;
    long unsigned iowtime, irqtime, sirqtime;
};

main()
{
    printf("Hello World");
}
```

Listing 6: Ukážka algoritmu

Algorithm 1 Ukážka príkazov pre algorithmic

```
  <text>
  if <condition> then
    <text>
  else
    <text>
  end if
  if <condition> then
    <text>
  else if <condition> then
    <text>
  end if
  for <condition> do
    <text>
  end for
  for <condition> to <condition> do
    <text>
  end for
  for all <condition> do
    <text>
  end for
  while <condition> do
    <text>
  end while
  repeat
    <text>
  until <condition>
  loop
    <text>
  end loop
Require: <text>
Ensure: <text>
  return <text>
  print <text> {<text>} and , or , xor , not , to , true, false
```

Záver

Conclusion is going to be where?
Here.

Zoznam použitej literatúry

1. BORGMAN, Christine L. *From Gutenberg to the global information infrastructure: access to information in the networked world*. First. Cambridge (Mass): The MIT Press, 2003. ISBN 0-262-52345-0.
2. GREENBERG, David. Camel drivers and gatecrashers: quality control in the digital research library. In: HAWKINS, B.L a BATTIN, P (ed.). *The mirage of continuity: reconfiguring academic information resources for the 21st century*. Washington (D.C.): Council on Library a Information Resources; Association of American Universities, 1998, s. 105–116.
3. LYNCH, C. Where do we go from here?: the next decade for digital libraries. *DLib Magazine* [online]. 2005, roč. 11, č. 7/8 [cit. 2005-08-15]. ISSN 1082-9873. Dostupné z: `http://www.dlib.org/dlib/july05/lynch/07lynch.html`.
4. DĚŤA, Hugh a RYCHLÍK, Tomáš. *A big paper: Podtitul* [online]. 2. vyd. Praha: Academia, 1991 [cit. 2011-01-12]. Pokusná edice. ISBN 978-44-55-X. Dostupné z: `http://pokus.cz`.
5. DĚŤA, Hugh, RYCHLÍK, Tomáš, DALŠÍ, Pepa, SPOUSTA, Pepa, SKORO, Moc, ALE, Nestačí a HODNĚ. *Úplně úžasná knížka*. 3. vyd. Praha, 1991.
6. DĚŤA, Hugh, RYCHLÍK, Tomáš, DALŠÍ, Pepa, SPOUSTA, Pepa, SKORO, Moc, ALE, Nestačí a HODNĚ. *Úplně úžasná knížka*. 3. vyd. Praha: MIT Press, 1991.
7. FREELY, I.P. A small paper: Podtitulek. *The journal of small papers*. 1997, roč. 1, č. 3, s. 2–5. to appear.
8. JASS, Hugh. A big paper. *The journal of big papers*. 1991, roč. 23.
9. Titulek. *The journal of big papers*. 1991, roč. 12, č. 2, s. 22–44. Dostupné z DOI: `10.112.22/jkn`.
10. KOLLMANNOVÁ, Ludmila, BUBENÍKOVÁ, Libuše a KOPECKÁ, Alena. *Angličtina pro samouky*. 5. vyd. Praha: Státní pedagogické nakladatelství, 1977. Učebnice pro samouky, č. 4. ISBN 80-04-25663-5.
11. NOVOTNÁ, Pepina. Podkapitola. In: KOLLMANNOVÁ, Ludmila, BUBENÍKOVÁ, Libuše a KOPECKÁ, Alena. *Angličtina pro samouky*. 5. vyd. Praha: Státní pedagogické nakladatelství, 1977, kap. 2., s. 22–29. Učebnice pro samouky, č. 4. ISBN 80-04-25663-5.

13. KNUTH, Donald. Journeys of TEX. *TUGBoat*. 2003, roč. 17, č. 3, s. 12–22. ISSN 1222-3333. Dostupné tiež z: `http://tugboat.tug.org/kkk.pdf`.

14. GENIÁLNÍ, Jiří (ed.). *Mimořádně užitečný sborník*. Praha: Academia, 2007. ISBN 978-222-626-222-2.

15. VLAŠTOVKA, Josef. Velmi zajímavý článek. In: GENIÁLNÍ, Jiří (ed.). *Mimořádně užitečný sborník*. Praha: Academia, 2007, s. 22–45. ISBN 978-222-626-222-2.

Prílohy

A Štruktúra elektronického nosiča

/CHANGELOG.md

· file describing changes made to FEIstyle

/example.tex

· main example *.tex* file for diploma thesis

/example_paper.tex

· example *.tex* file for seminar paper

/Makefile

· simply Makefile – build system

/fei.sublime-project

· is project file with build in Build System for Sublime Text 3

/img

· folder with images

/includes

· files with content

/bibliography.bib

· bibliography file

/attachmentA.tex

· this very file

B Algoritmus

```
struct kern_ipc_perm {
        spinlock_t      lock;
        bool            deleted;
        int             id;
        key_t           key;
        kuid_t          uid;
        kgid_t          gid;
        kuid_t          cuid;
        kgid_t          cgid;
        umode_t         mode;
        unsigned long   seq;
        void            *security;
        struct rhash_head khtnode;
        struct rcu_head rcu;
        refcount_t refcount;
} ____cacheline_aligned_in_smp __randomize_layout;
```

Listing B.1: Štruktúra kern_ipc_perm

```
struct msqid64_ds {
        struct ipc64_perm msg_perm;
        __kernel_time_t msg_stime;   /* last msgsnd time */
#if __BITS_PER_LONG != 64
        unsigned long  __unused1;
#endif
        __kernel_time_t msg_rtime;   /* last msgrcv time */
#if __BITS_PER_LONG != 64
        unsigned long  __unused2;
#endif
        __kernel_time_t msg_ctime;   /* last change time */
#if __BITS_PER_LONG != 64
        unsigned long  __unused3;
#endif
        __kernel_ulong_t msg_cbytes; /* current number of bytes on queue */
        __kernel_ulong_t msg_qnum;   /* number of messages in queue */
        __kernel_ulong_t msg_qbytes; /* max number of bytes on queue */
        __kernel_pid_t msg_lspid;    /* pid of last msgsnd */
        __kernel_pid_t msg_lrpid;    /* last receive pid */
        __kernel_ulong_t __unused4;
        __kernel_ulong_t __unused5;
};
```

Listing B.2: Štruktúra msqid64_ds

```
struct msqid_ds {
        struct ipc_perm msg_perm;
        struct msg *msg_first;       /* first message on queue,unused */
        struct msg *msg_last;        /* last message in queue,unused */
        __kernel_time_t msg_stime;   /* last msgsnd time */
```

```
        __kernel_time_t msg_rtime;   /* last msgrcv time */
        __kernel_time_t msg_ctime;   /* last change time */
        unsigned long msg_lcbytes;   /* Reuse junk fields for 32 bit */
        unsigned long msg_lqbytes;   /* ditto */
        unsigned short msg_cbytes;   /* current number of bytes on queue */
        unsigned short msg_qnum;     /* number of messages in queue */
        unsigned short msg_qbytes;   /* max number of bytes on queue */
        __kernel_ipc_pid_t msg_lspid; /* pid of last msgsnd */
        __kernel_ipc_pid_t msg_lrpid; /* last receive pid */
};
```

Listing B.3: Štruktúra msqid_ds

```
struct msginfo {
        int msgpool;
        int msgmap;
        int msgmax;
        int msgmnb;
        int msgmni;
        int msgssz;
        int msgtql;
        unsigned short msgseg;
};
```

Listing B.4: Štruktúra msginfo

C Výpis sublime

```
../../ fei .sublime-project
```

Listing C.1: Ukážka sublime-project