A9Nマイクロカーネル

伊組烈火

“理解しやすい”カーネル

“理解しやすい”カーネル

年	トークン数	比率	累積
1991	2,964	0.00%	0.00%
1992	30,740	0.03%	0.03%
1993	46,910	0.04%	0.07%
1994	38,443	0.03%	0.11%
1995	95,498	0.09%	0.19%
1996	204,573	0.19%	0.38%
1997	370,193	0.34%	0.72%
1998	436,941	0.40%	1.11%
1999	775,706	0.70%	1.81%
2000	1,469,450	1.33%	3.15%
2001	828,530	0.75%	3.90%
2002	2,475,002	2.24%	6.14%
2003	1,310,598	1.19%	7.33%
2004	2,023,705	1.83%	9.16%
2005	2,575,195	2.33%	11.49%
2006	2,449,966	2.22%	13.71%
2007	3,118,829	2.83%	16.54%
2008	4,410,921	4.00%	20.54%
2009	5,461,240	4.95%	25.49%
2010	4,805,525	4.35%	29.84%
2011	5,859,906	5.31%	35.15%
2012	6,072,494	5.50%	40.65%
2013	6,440,436	5.84%	46.49%
2014	6,091,508	5.52%	52.01%
2015	8,968,298	8.13%	60.14%
2016	7,622,786	6.91%	67.04%
2017	11,028,463	9.99%	77.04%
2018	9,248,269	8.38%	85.42%
2019	10,843,056	9.83%	95.24%
2020*	5,248,662	4.76%	100.00%

cf. https://www.linuxfoundation.jp/wp-content/uploads/2020/09/JA2_2020-Linux-Kernel-History-Report.pdf
“Linux Foundation - 2020年度版 Linuxカーネルの歴史レポート” P. 6

近年, コードの肥大と共にオペレーティングシステムを学ぶハードルが高くなってきている

→Linuxカーネルは, 2020年時点でトークン数が**1700倍以上**に拡大している

(2000万行！)

では, コード量の少ないカーネルはどうだろう？

Unix V6 (約1万行)

・古すぎる(1975年リリース)

X86に移植されたxv6が存在するが, 基本的にPDP-11向けのコード

Pre K&R Cで書かれていて, 現代では考えられないようなコードが存在する

・移植容易性が不十分

ほぼ全てのコードがPDP-11に依存していて, なおかつ分離が不十分

・分かりにくい

現代のプログラムでは推奨されないテクニックが山盛りになっている(e.g. リターンアドレスの書き換え)

変数名の省略が分かりにくい

```
// /usr/sys/dmr/rk.c.
#define    RKADDR    0177400
struct {
    int rkds;
    int rker;
    int rkcs;
    int rkwc;
    int rkba;
    int rkda;
};
devstart(bp, &RKADDR->rkda, rkaddr(bp), 0);
// 無名構造体でレジスタにアクセスしている！
```

では, コード量の少ないカーネルはどうだろう？

MINIX (3万行)

・これも古い

　　日本語の解説書籍が絶版になっている(オペレーティングシステム 設計と実装)

・マクロの多用

　　多数のコンパイラに対応させるためとはいえ分かりにくくなっている

・変数名の省略が分かりにくい

→これじゃダメだろ！

“理解しやすい”コード例

```c
typedef struct {
    unsigned char   e_ident[EI_NIDENT]; /* File identification. */
    Elf64_Half  e_type;     /* File type. */
    Elf64_Half  e_machine;  /* Machine architecture. */
    Elf64_Word  e_version;  /* ELF format version. */
    Elf64_Addr  e_entry;    /* Entry point. */
    Elf64_Off   e_phoff;    /* Program header file offset. */
    Elf64_Off   e_shoff;    /* Section header file offset. */
    Elf64_Word  e_flags;    /* Architecture-specific flags. */
    Elf64_Half  e_ehsize;   /* Size of ELF header in bytes. */
    Elf64_Half  e_phentsize;    /* Size of program header entry. */
    Elf64_Half  e_phnum;    /* Number of program header entries. */
    Elf64_Half  e_shentsize;    /* Size of section header entry. */
    Elf64_Half  e_shnum;    /* Number of section header entries. */
    Elf64_Half  e_shstrndx; /* Section name strings section. */
} Elf64_Ehdr;
```

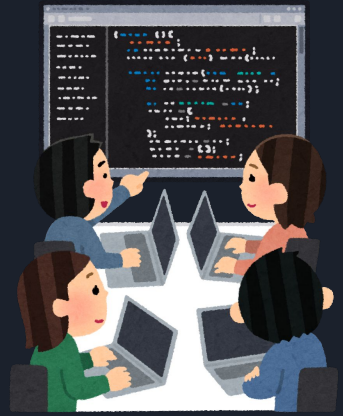
FreeBSDのELF構造体: e_shstrndxが分かりにくい

```c
typedef struct
{
    unsigned char identifier[16];
    elf64_half type;
    elf64_half machine;
    elf64_word version;
    elf64_address entry_point_address;
    elf64_offset program_header_offset;
    elf64_offset section_header_offset;
    elf64_word flags;
    elf64_half size;
    elf64_half program_header_size;
    elf64_half program_header_number;
    elf64_half section_header_size;
    elf64_half section_header_number;
    elf64_half section_header_string_table_index;
} elf64_header;
```

A9NLoaderのELF構造体: 省略せず理解しやすい命名にしている

OS開発サイクルを作り出す

A9NでOSを学ぶ
→最初の段階に入りやすく



A9Nの開発に参加する

よりよいOSへ

マイクロカーネルの素晴らしさ

MINIXの再生サーバー (Reincarnation Server, RS)

カーネル空間ではなくユーザー空間でドライバが動作しているため, 再生サーバーがクラッシュを検知し自動で再起動させることができる

→マイクロカーネルならフォールトトレランスで安定した動作が可能である

cf. https://youtu.be/vIOsy0PZZyc

“MINIX 3 at the Embedded World Exhibition in Nuremberg”, Andrew S. Tanenbaum

抽象化 (Abstraction)

車を操作するとき, 内部でギアとエンジンがどう協調して動いているか知っているか？

→大部分の人間が”いいえ”だと思う

アクセルとハンドルによって操作が抽象化されているから　, 新しい車でもすぐに運転が可能である

マイクロカーネルと**抽象化**

カーネルが**抽象化**されたデバイスに対して操作する

→具象に直接依存しないことで，ハードウェアとソフトウェアの境界を明確にする

→ハードウェアを抽象化し，移植時に変更されるコード量を減らす

抽象化され分離された HALが違いを吸収し，

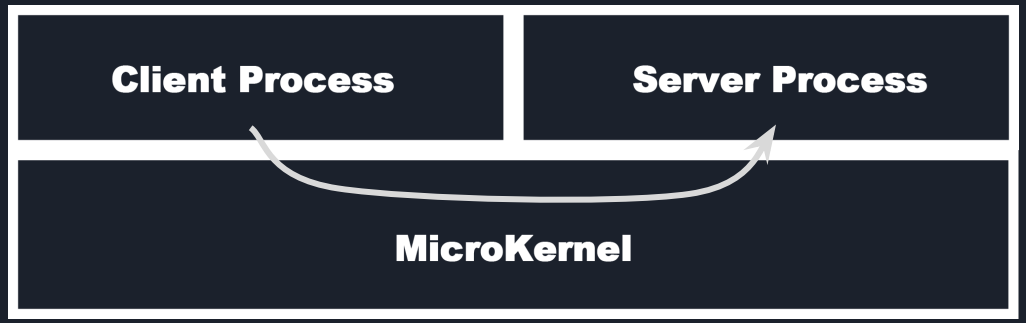
移植容易性を実現する

メッセージパッシング

A9Nでは, プロセス間通信がメッセージングにより実現される

→”メッセージ”という形に**抽象化**することで, 凝集性の高いコンポーネント分割が可能になる

→カーネルは最小単位の処理+メッセージングだけを行うので, 規模が小さくなる



理解しやすいコードになり , 保守性が向上する

A9Nで達成したいこと

・マイクロカーネル学習のハードルを下げる

　　これからの世代の人間とOSを作り上げていき, 低レイヤーの面白さを伝えたい

・Linuxを超えるOSにしたい

　　一人では無理でも, 面白さを伝えていければ実現が可能

　　マイクロカーネルで世界を埋め尽くす

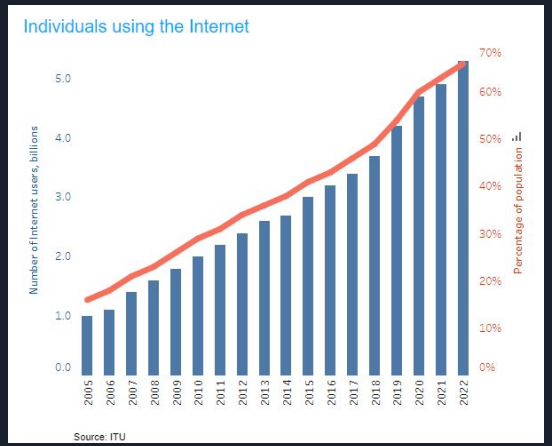
・抽象化のパワーを伝えたい

　　プログラミングの本質とは”抽象化”であると考えている

これからの未来

情報化が進み, 世界中の人間が情報端末を持つようになってきている

→インターネット普及率の爆発的増加からも明らかである



cf. https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

“ITU - Statistics”

マイクロカーネルと適切に設計された HALからなるOSにより,
世界は安定性と統一プラットフォームを手に入れる
→快適なOSからイノベーションが生まれ , 技術進化を後押しする

ご清聴ありがとうございました