IT-Universitetet i København 3. september 2014

Operativsystemer og C

Ugeseddel 2 (uge 36) 2-8/9 2013

Forelæsning:

Fredag d. 5. september kl. 10.00 - 11.50 i 2A56

Emner

Systemkald og systemprogrammer. Operativsystems opbygning. Lagdelt opbygning. Mikrokerner. Moduler. Virtuelle maskiner. Implementation og boot af kernen. Debugging. Make. Programmering i C.

Litteratur og andet kursusmateriale

Silbersschatz, kapitel 2.

Følgende vejledninger på nettet om bygning af Ubuntu Linux kernen er meget detaljeret og god læsning - især hvis I løber ind i problemer undervejs.

- `https://help.ubuntu.com/community/Kernel/Compile`

Til `make` kan følgende vejledninger på nettet anbefales:

- `http://mrbook.org/tutorials/make/` (meget kort tutorial med udgangspunkt i C++ - men dækkende for opgave 1 - erstat `g++` → `gcc`, `.cpp` → `.c`, etc.)
- `http://www.gnu.org/software/make/manual/make.html`

Til dem der ønsker at lave bash scripts - eller blot finde information om bash:

- `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html`

Disse 4 ovenstående links er **ikke** del af pensum.

Øvelser: Fredag d. 5. september kl. 12.00 - 13.50 i 4A16.

Forberedelser (lav dette før øvelserne går i gang)

*** Læs hele opgaveteksten igennem så du ved hvad du skal til øvelserne. ***

Til øvelserne i denne uge skal du bruge

- En virtuel maskine med Linux installeret (følg fremgangen i boscvm.pdf).
- C compiler `gcc` og en tekst editor.
- Kerne-kildekoden til Linux kernen (opgave 2 går ud på at bygge Linux kernen - du opfordres til at lave denne opgave som forberedelse!).

Opgave 1. `make` ($\frac{1}{2}$ time)

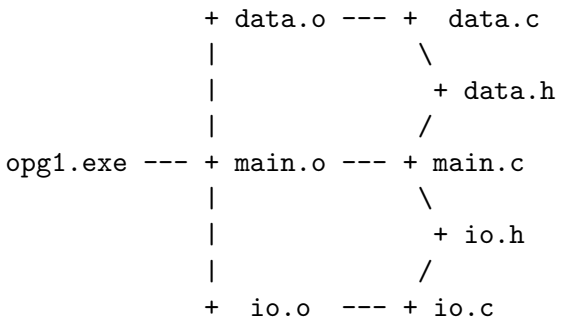
*** Denne opgave kan laves mens kernen bygges i opgave 2 ***

Målet med denne opgave er, at du kan

- Bruge `make` til at bygge større C programmer.

På kursusbloggen kan downloades `makeopgave.zip`, der indeholder kildekode (`main.c`, `data.c`, `io.c`) og headerfiler (`data.h`, `io.h`) for et simpelt C program.

1.1) I denne opgave skal du opbygge en Makefil med følgende afhængighedsgraf:



Du må godt bruge de indbyggede `make` regler, men kan også - for øvelsens skyld - lave dine egne fra bunden (brug `make -r` for at køre `make` uden indbyggede regler).

Husk at afprøve din Makefile ved at ændre i kildekoden og tjekke at `make` gør det rigtige (altså kun oversætter den fil du ændrede i).

1.2) Tilføj et target „clean“, som vha. `rm -f` fjerner alle objekt (`.o`) filer, sådan at du kan bygge hele programmet på ny med:

```
make clean
make
```

Opgave 2. Bygning af en Linux kerne ($1-2$ timer)

Målet med denne opgave er, at du kan

- Foklare hvad der kræves for at få en kerne til at fungere.
- Finde oplysninger om den hardware, der anvendes i computeren, og bruge disse informationer til at konfigurere kernen.
- Bygge din egen kerne og beskrive de skridt, der er nødvendige.

2.1) Følg vejledningen i `linuxkernen.pdf` til at bygge og installere din egen kerne.

Opgave 3. Antallet af linjer i kerne-kildekoden ($\frac{1}{2}$ time)

*** Denne opgave kan laves mens kernen bygges i opgave 2 ***

Målet med denne opgave er, at du kan

- Få øvelse i at bruge Linux-kommandoerne `find`, `xargs`, og `wc`.

I næste uge skal vi lære om processor og hvordan de kan kommunikere vha. pipes

Her er en praktisk anvendelse af dette. Det er f.eks muligt at finde antallet af linjer i en fil ved at udskrive den med `cat` via en pipe `|` over til `wc`:

```
root@bosc:~/uge2/opg3# cat main.c | wc -l
10
```

I dette tilfælde er der 10 linjer i `main.c`.

3.1) Hvor mange linier C- og assembler-kode er der i kildekoden til linux-kernen? Brug kommandoerne `find`, `xargs`, `cat`, og `wc` til at tælle antal linier i filer, der indeholder henholdsvis C- og assembler-kode.

Det kan antages, at C-kode findes i filer der ender på `.c` eller `.h`, og assembler-kode findes i filer der ender på `.s` eller `.S`.

Opgave 4. Tilføjelse af et nyt systemkald (2 timers kompilering, 20minutters arbejde)*(antal fejl)

Målet med denne opgave er, at du kan

- Implementere dit eget systemkald i en Linux kerne.
- Beskrive hvordan brugerprogrammer kan anvende operativsystemets systemkald.

Part 1:

Det nemmeste „eksperiment“ er at lave en simpel ændring i den eksisterende kernekildekode. F.eks. kan du ændre i kernen for at få den til at skrive en meddelelse ud ved opstart; et simpelt „Hello World!“ for Linux kerner. Udskrivning af tekst fra kernen skal gøres med funktionen `printk()`. Den modsvarer `printf` fra standardbiblioteket i C.

Din meddelelse kan f.eks. tilføjes filen `init/main.c`. Det vil betyde, at din tekst udskrives ved initialisering af kernen.

Åben filen `init/main.c` med en tekst-editor. Denne fil består af mange hundrede linjers kode, men hvis du bruger editoren til at søge efter `calibrate_delay()` finder du følgende sted i kildekoden:

```
if (late_time_init)
   late_time_init();
sched_clock_init();
calibrate_delay();
pidmap_init();
```

Et godt sted at indsætte din meddelelsen er lige efter linjen `calibrate_delay();`.

Gem filen og kompiler/installer den (se skridt 3 i vejledningen `linuxkernen.pdf`).

Du kan næsten med sikkerhed ikke nå at se din meddelelse ved opstart så du kan f.eks. udføre følgende kommando når du er logget ind:

```
dmesg | grep -i hello
```

Kommandoen `dmesg` giver en liste med alle meddelelser fra kernen siden opstart. `grep` matcher til teksten „hello“ og flaget `-i` gør at der ikke skelnes mellem store og små bogstaver.

Part 2:

I den anden del vil vi tilføje vores eget system kald.

1) Lav dit nye system kald i 'hello.c' i 'kernel/' som indeholder:

```
#include <linux/kernel.h>

asmlinkage long sys_hello(void) {
    printk("Hello World\n");
    return 0;
}
```

2) Tilføj dit objekt til 'Makefile' i 'kernel/' hvor du også lagde 'hello.c'. Dette gøres ved at tilføje 'hello.o' til objekterne listet efter 'obj-y'.

3) Giv dit system kald et nummer i tabellen og bind det din funktion. Tilføj:

```
#define __NR_hello 274
__SYSCALL(__NR_hello, sys_hello )
```

til

include/uapi/asm-generic/unistd.h

Tallet 274 afhænger hvor mange system kald der er i din version af kernen. Vælg det mindste ledige.

4) På samme vis tilføj:

```
314  64 hello sys_hello
```

til

arch/x86/syscalls/syscall_64.tbl

Dette antager selvfølgelig du har en 64bit version af kernen.

5) Endeligt tilføj dit system kald til linux include filerne så vi kan bruge det:

```
asmlinkage long sys_hello(void);
```

til

include/linux/syscalls.h

6) Nu kan du compilere kernen som beskrevet i pdf'en, men du skal huske det tager en times tid. Så det er en god idé at sikre at alle skridt er fuldt til punkt og prikke før du går igang.

7) Du kan imens der kompileres lave et lille helloworld.c program der bruger dit nye systemkald. Når du bygger dit eget helloworld program er det vigtigt at de rigtige headerfiler (altså dem du har editeret i) includeres i `gcc`. Dette kan du gøre nemmest ved at angive stien til headerfilerne direkte, f.eks:

```
gcc -I~/Documents/uge2/linux-3.13/include main.c -o main
```

Programmet 'main.c' kunne se sådan ud:

```
#include <sys/syscall.h>
#include <stdio.h>
#include <unistd.h>

int main( int argc, char **) {
  printf("calling kernel, remember the output is only visible by calling dmesg..");
  syscall( 314 );
}
```

Tillykke! Du er nu en kerne hacker eller en taber afhængigt af din success.