Obligatorisk Innlevering 2 Minne

Temaet for denne obligatoriske øvelsen er minnebruk og hvordan operativsystemet håndterer denne ressursen. Det er viktig å bruke en virtuell maskin med Linux for denne øvelsen, siden Windows eller Mac kan håndtere minne forskjellig. Hvis du allerede bruker Linux, trenger du selvfølgelig ikke å bruke en virtuell maskin.

Oppgave 1

I denne oppgaven skal du undersøke hvordan det virtuelle minneområdet til et program ser ut. Mer presist, du skal finne de virtuelle minneadressene til disse delene av programmet:

- Stacken
- Heapen
- Programkoden
- Globale variabler

Oppsett

Som et sikkerhetstiltak skjer det noe randomisering av det virtuelle minneområdet. For å kunne reprodusere resultatene ønsker vi å deaktivere dette for denne oppgaven. Det er en enkel måte å fikse det på ved å bruke kommandoen `setarch`. Den kan for eksempel brukes til å starte et nytt skall hvor randomisering er slått av:

```
setarch -R bash
```

Se manualen for mer detaljer.

Når du kompilerer, pass på å ikke bruke optimalisering, siden vi ikke ønsker at kompilatoren skal optimalisere bort noe. Og ta gjerne med **-static** for å inkludere biblioteker i den kjørbare filen.

Innlevering

Når du har samlet inn verdiene, skal du manuelt lage en tekstfil der du viser med **ASCII**-tegn hvordan minneområdet ser ut. Bruk minneadressene du fikk ut, filen skal følge omtrent figur 13.3 i boken:

```
=========
[adresse]  [navn til området]
--------------
Ikke brukt
--------------
[adresse] [navn til området]
=========
```

Du skal også levere koden du har skrevet og en forklaring på hvor verdiene du fant kommer fra. Under følger mer detaljer om hva som forventes til de forskjellige delene av programmet.

Stacken

For stacken vil vi vite hvilken minneadresse en variabel på stacken får, men også i hvilken retning stacken vokser. Du kan bruke en rekursiv funksjon til å se hvordan adressene endrer seg etter hvert som stacken vokser. Du kan starte med en funksjon som denne:

```
void f(int i) {

    if (i == 0)

        return;

    // Din kode her

    f(i - 1);

}
```

Heapen

Alloker en relativt stor mengde minne (>1MB) og se hvilken minneadresse du får. Du trenger ikke undersøke heapen grundigere nå; det kommer tilbake i neste oppgave.

Programkoden

Det er ikke så åpenbart å si hvor programkoden befinner seg, men det er flere måter å finne det ut på. Du kan for eksempel skrive ut adressen til en funksjon i programmet eller bruke en debugger.

Globale Variabler

Til slutt, finn ut hvor globale variabler havner. Har det noen betydning om de er initialisert eller ikke?

Oppgave 2

I denne oppgaven skal du se nærmere på hvordan heapen fungerer, og hva malloc egentlig gjør.

Oppsett

Start med et program som ikke terminerer umiddelbart, for eksempel ved å ha en **scanf**-linje på slutten. Før det, alloker minne på heapen som i oppgave 1 (>1MB) og skriv ut minneadressen du får. Når du kjører programmet, skal du bruke verktøyet `pmap` til å undersøke minne utenfra. Start med å lese manualen til pmap. Som du ser, trenger vi prosess-ID-en til programmet for å bruke pmap. Du kan enten skrive denne ut fra programmet eller finne den utenfra med `ps`, `pgrep`, `top` eller lignende verktøy. Det kan kombineres på en linje som dette:

```
pmap -x $(pgrep a.out)
```

Innlevering

For hver deloppgave, lever inn koden du har skrevet og en skriftlig besvarelse på det oppgaven spør om.

2a

Ved å sammenligne minneadressen som malloc gir deg med tilsvarende adresse i pmap, ser vi at addressen malloc gir oss er 16 bytes (**0x10**) mer enn hva pmap viser. Finn en måte å lese dette minne på, enten ved å endre på koden eller ved å bruke en debugger som `lldb`. Hva blir lagret her? Og hva tror du malloc bruker dette minne til?

2b

Med `pmap -x` kan du se størrelsen på et minneområde, men også hvor mye minne som faktisk befinner seg i fysisk minne (**RSS**). Som du kanskje husker, så mappes det virtuelle minne til fysisk minne i pages med en fast størrelse. Prøv å endre programmet slik at du skriver noe til området malloc gir deg. Kan du med dette finne ut hvor stor en page er? Og hvorfor får vi en page med en gang, selv når vi ikke har skrevet noe til minne?

2c

Bruk kommandoen `free -h` til å se hvor mye minne du har tilgjengelig. Prøv gradvis å allokere mer minne i ett enkelt kall til malloc. Når går noe galt? Prøv deretter med flere kall til malloc med mindre verdi per kall. Hvor mye minne kan du allokere nå? Til slutt, gjenta det siste steget, men denne gangen skriv noe til minne også. Hvordan krasjer programmet nå?

Du må laste dette verktøyet i eit nytt nettlesarvindauge

Last Obligatorisk Innlevering 2 Minne i eit nytt vindauge