LINKÖPINGS UNIVERSITET

TDIU16: Process- och operativsystemprogrammering

Skapa stack till `main`

Klas Arvidsson, Daniel Thorén, Filip Strömbäck

INSTITUTIONEN FÖR DATAVETENSKAP

Vårterminen 2023

2020-03-06

1 Mål

När ett program anropar en funktion sker ofta överföring av information till funktionen via parametrar, och från funktionen via returvärden. I ett operativsystem kan man på ett liknande sätt se program som en sorts ”funktioner”. Ett sätt att överföra information till ett program är via kommandoradsparametrarna. Den delen ska vi implementera i den här laborationen.

Den första funktionen som körs i ett program är funktionen `_start`, som i princip direkt anropar `main`. Båda dessa funktioner tar två parametrar: `argc` och `argv` som innehåller de kommandoradsargument som skickades till programmet. Eftersom `_start` och `main` är vanliga C-funktioner förväntar sig de att parametrarna i vanlig ordning ligger på programmets stack. Som operativsystem är vi därför ansvariga för att se till att lägga dit parametrarna korrekt innan vi startar programmet.

I den här laborationen ska vi alltså implementera en funktion som tar en sträng av kommandoradsparametrar, hitta de olika orden i strängen och med hjälp av dessa konstruera arrayen `argv` samt antalet ord för `argc`. Vi ska sedan lagra den datan på stacken för den nya processen så att det ser ut som att någon annan funktion anropade `main` på vanligt sätt. Vi börjar med att implementera denna funktionaliteten separat från Pintos så att det är enklare att felsöka, och sedan flyttar vi in den färdiga funktionaliteten i Pintos när den är klar.

Mer information om hur parameteröverföring fungerar i pintos finns i Pintos-Wiki under rubriken ”Minne”, se särskilt underrubriken ”Parameteröverföring vid funktionsanrop”.

2 Uppgift

Du skall skriva ett program som läser in en ”kommandorad”, allokerar minne för en teoretisk stack, och placerar kommandoraden på denna stack så som funktionen `main` förväntar sig. Filen `setup-argv.c` i mappen `src/standalone/labx5/` i Pintos-repositoryt innehåller ytterligare idéer på hur du kan gå tillväga. Som exempel på hur stacken skall se ut kan du studera följande kommandorad och resultat:

```
Kommandorad: " this will be arguments  to  main " (34 + 1 byte)
Punkt används här för att ange noll-tecknet. '.' => '\0'
Adress och innehåll anges hexadecimalt.

Adress    Innehåll  Datatyp
--------  --------  ---------
C0000000  <PHYS_BASE i pintos kernel space, oläsbar>
BFFFFFFC  n...      char (x4)
BFFFFFF8   mai      char (x4)
BFFFFFF4   to.      char (x4)
BFFFFFF0  nts.      char (x4)
BFFFFFEC  gume      char (x4)
BFFFFFE8  e.ar      char (x4)
BFFFFFE4  ll.b      char (x4)
BFFFFFE0  s.wi      char (x4)
BFFFFFDC   thi      char (x4) <rad STR>
BFFFFFD8  00000000  (char*)
BFFFFFD4  BFFFFFF9  (char*)
BFFFFFD0  BFFFFFF5  (char*)
BFFFFFCC  BFFFFFEA  (char*)
BFFFFFC8  BFFFFFE7  (char*)
BFFFFFC4  BFFFFFE2  (char*)
BFFFFFC0  BFFFFFDD  (char*)  <pekar till 't' på rad STR>
BFFFFFBC  BFFFFFC0  (char**) <pekar till raden ovanför>
```

```
BFFFFFB8  6         (int)
BFFFFFB4  00000000  (void (*)(void)) <funktionspekare>
```

Standardfunktionen för att beräkna längden av C-strängar (pekare till konstanta tecken där sista tecknet är noll-teckenet) heter `strlen` och finns i `<string.h>`. Notera att `strlen` inte räknar med det avslutande noll-tecknet, men att en extra byte lagringsutrymme krävs för detta i alla C-strängar. Standardfunktion för att dela upp en sträng i ”tokens” heter `strtok_r` och finns i `<string.h>`. *Notera att originalsträngen förstörs av strtok_r!*

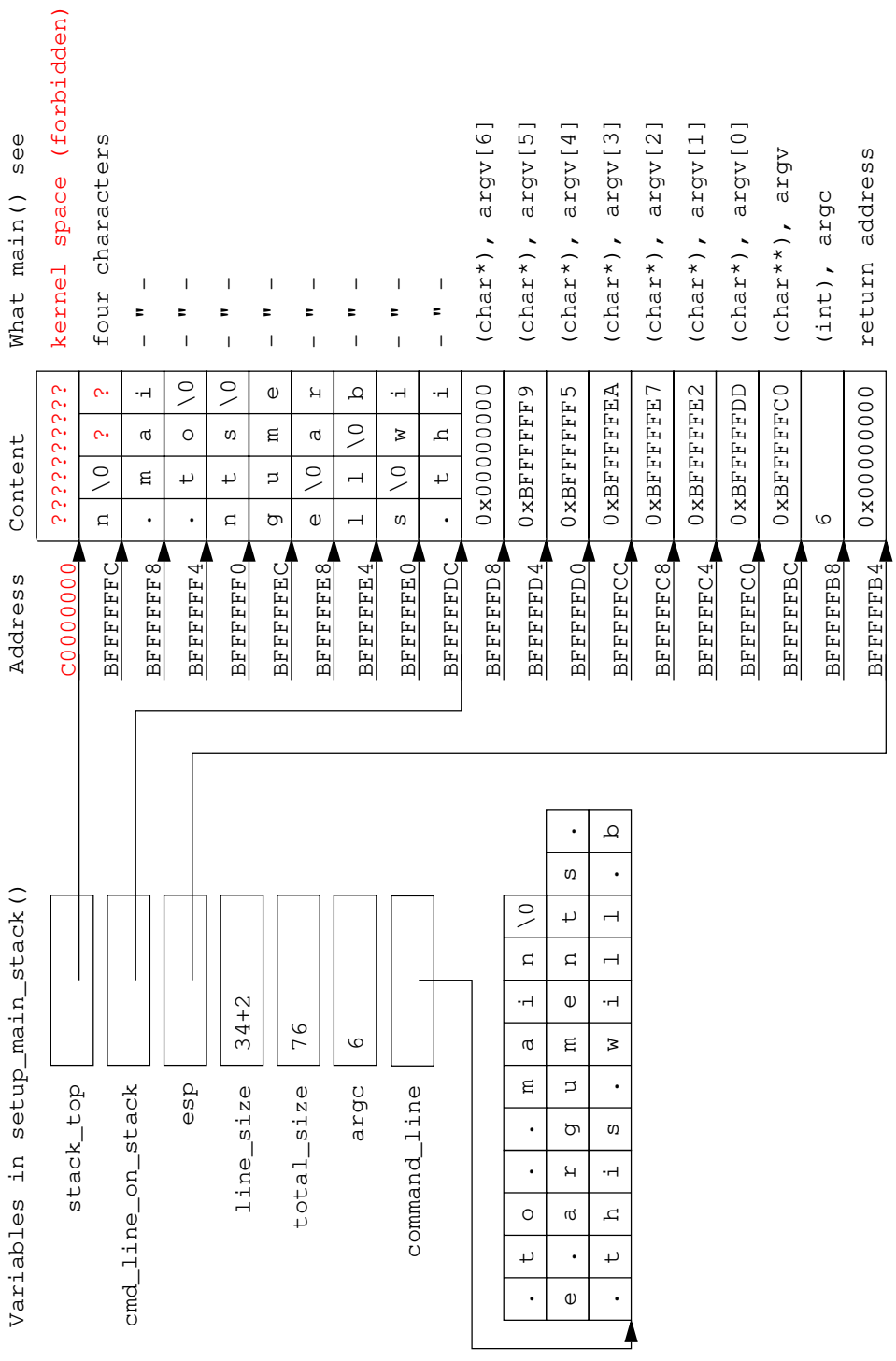
Ett exempel som skriver ut varje delsträng i originalsträngen `orig`:

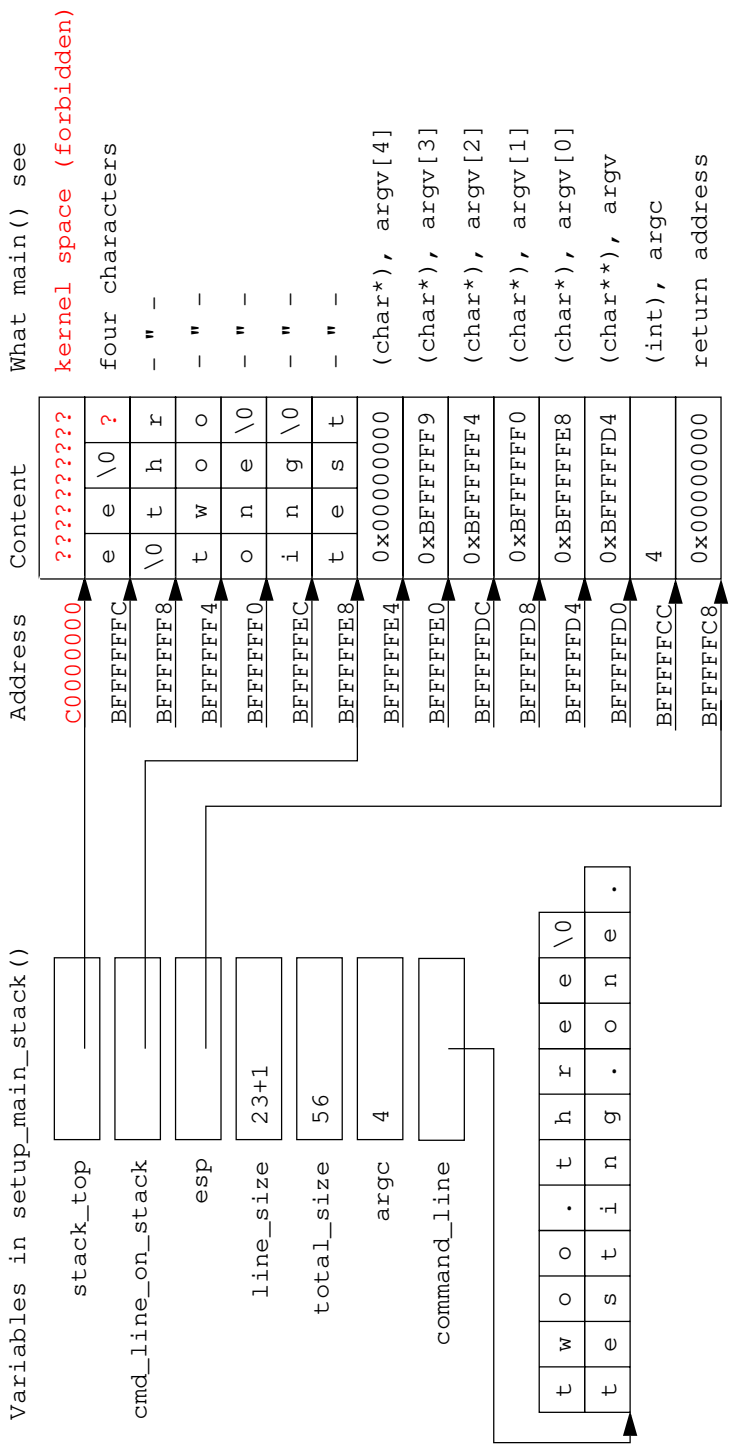
```
for (token = strtok_r (orig, " ", &save_ptr); token != NULL;
     token = strtok_r (NULL, " ", &save_ptr))
  printf ("'%s'\n", token);
```

OBS! Exemplet bygger på situationen i Pintos precis innan `main` startas. I denna uppgift kommer du se andra adresser. Utskriften av stacken kommer inte heller se ut som ovan, utan endast en byte per rad skrivs ut, dels hexadecimalt (på jämna adresser), och dels som tecken (alla adresser).

Mer exempel ges på kommande sidor. Där är `line_size` angiven med stränglängden plus avrundningen upp till ett tal jämnt delbart med fyra. Står det 21+3 avrundas stränglängden 21 alltså uppåt med tre byte för att använda 24 byte totalt.

Överkurs: Den som vill vara lite mer avancerad kan ta bort dubblerade blanksteg i kommandoraden för att spara ett par byte på stacken. Inte heller det inledande och avslutande blanksteget från kommandoraden behöver lagras på stacken. Att detta ändå lagras beror på att koden blir enklare så.


Variables in setup_main_stack()
stack_top
cmd_line_on_stack
esp
line_size
34+2
total_size
76
argc
6
command_line
Address
Content
What main() see
C0000000
????????
kernel space (forbidden)
BFFFFFFC
n \0 ? ?
four characters
BFFFFFF8
. m a i
- " -
BFFFFFF4
. t o \0
- " -
BFFFFFF0
n t s \0
- " -
BFFFFFEC
g u m e
- " -
BFFFFFE8
e \0 a r
- " -
BFFFFFE4
l l \0 b
- " -
BFFFFFE0
s \0 w i
- " -
BFFFFFDC
. t h i
- " -
BFFFFFD8
0x00000000
(char*), argv[6]
BFFFFFD4
0xBFFFFFF9
(char*), argv[5]
BFFFFFD0
0xBFFFFFF5
(char*), argv[4]
BFFFFFCC
0xBFFFFFEA
(char*), argv[3]
BFFFFFC8
0xBFFFFFE7
(char*), argv[2]
BFFFFFC4
0xBFFFFFE2
(char*), argv[1]
BFFFFFC0
0xBFFFFFDD
(char*), argv[0]
BFFFFFBC
0xBFFFFFC0
(char**), argv
BFFFFFB8
6
(int), argc
BFFFFFB4
0x00000000
return address
. t o . . m a i n \0
e . a r g u m e n t s .
. t h i s . w i l l .


Variables in setup_main_stack()
stack_top
cmd_line_on_stack
esp
line_size
23+1
total_size
56
argc
4
command_line
Address
Content
What main() see
C0000000
??????????
kernel space (forbidden)
BFFFFFFC
e e \0 ?
four characters
BFFFFFF8
\0 t h r
- " -
BFFFFFF4
t w o o
- " -
BFFFFFF0
o n e \0
- " -
BFFFFFEC
i n g \0
- " -
BFFFFFE8
t e s t
- " -
BFFFFFE4
0x00000000
(char*), argv[4]
BFFFFFE0
0xBFFFFFF9
(char*), argv[3]
BFFFFFDC
0xBFFFFFF4
(char*), argv[2]
BFFFFFD8
0xBFFFFFF0
(char*), argv[1]
BFFFFFD4
0xBFFFFFE8
(char*), argv[0]
BFFFFFD0
0xBFFFFFD4
(char**), argv
BFFFFFCC
4
(int), argc
BFFFFFC8
0x00000000
return address
t w o o . t h r e e \0
t e s t i n g . o n e .