Stack Printing Programs (Yığın Çıktı Programları)

Şimdiye kadar Assembly hakkında özellikle hangi fonksiyonu çağıracağı hakkında birçok şey öğrendik. Ancak, Biz Assembly ile genellikle program yazmayız. Öyleyse C dilinde stack çağrısı nasıldır? Muhtemelen birkaç farklı içerikle beraber stack çağrısı tam olarak Assembly ile aynı görünür.

İlk denememiz, **printstack.c** kaynak kodu:

```
safechar(char c)
{
  if (c >= 'a' && c <= 'z') return(c);
  if (c >= 'A' && c <= 'Z') return(c);
  if (c >= '0' && c <= '9') return(c);
  if (c == ' ') return(c);
  if (c == '.') return(c);
  if (c == '-') return(c);
  if (c == ',') return(c);
  if (c == '(') return(c);
  if (c == ')') return(c);
  if (c == '[') return(c);
  if (c == ']') return(c);
  if (c == '{') return(c);
  if (c == '}') return(c);
  return '@';
}
main(int argc, char **argv, char **envp)
{
  char **s;
  char *s2;
  int i, top;

  if (argc < 2) {
    fprintf(stderr, "usage: printstack nentries (& other junk if you
want)\n");
    exit(1);
  }

  top = atoi(argv[1]);

  printf("&argc = 0x%x\n", &argc);
  printf("&argv = 0x%x\n", &argv);
  printf("&envp = 0x%x\n", &envp);
  printf("\n");

  s = (char **) &s;
  for (i = 0; i < top; i++) {
    s++;
    s2 = (char *) s;
    printf("0x%x : %15d 0x%-8x %2c %2c %2c %2c\n", s, *s, *s,
safechar(s2[0]),
              safechar(s2[1]), safechar(s2[2]), safechar(s2[3]));
  }
}
```

İlk argüman olarak bir sayı alır, ve sonra s adresiyle başlayan stack üzerinde birçok kelime yazdırılır. Bir integer, bir hexedecimal ve dört karakter gibi bu üç yolla her kelime yazdırılır. Normal olan bir karakter değilse "@" işareti ile yazdırılır.

Aşağıda bunun bazı değerler ihmal edilerek 32-bit linux (debian)daki çıkışı bulunmaktadır. Lütfen argc, argv ve envp yapılarına odaklanın.

```
UNIX> gcc printstack.c
UNIX> a.out 200 Jim Plank
&argc = 0xbfffde10
&argv = 0xbfffde14
&envp = 0xbfffde18

0xbfffde08 :     -1073750456 0xbfffde48  H  @  @  @
0xbfffde0c :      1107391881 0x42017589  @  u  @  B
0xbfffde10 :               4 0x4         @  @  @  @  <--------- argc
0xbfffde14 :     -1073750412 0xbfffde74  t  @  @  @  <--------- argv
0xbfffde18 :     -1073750392 0xbfffde88  @  @  @  @  <--------- envp
0xbfffde1c :       134513466 0x804833a   @  @  @  @
0xbfffde20 :       134514528 0x8048760   @  @  @  @
.................
.................
0xbfffde6c :               0 0x0         @  @  @  @
0xbfffde70 :               4 0x4         @  @  @  @
0xbfffde74 :     -1073743572 0xbffff92c  ,  @  @  @  <--------- argv[0]
0xbfffde78 :     -1073743566 0xbffff932  2  @  @  @  <--------- argv[1]
0xbfffde7c :     -1073743561 0xbffff937  7  @  @  @  <--------- argv[2]
0xbfffde80 :     -1073743557 0xbffff93b  @  @  @  @  <--------- argv[3]
0xbfffde84 :               0 0x0         @  @  @  @  <--------- argv is
NULL terminated
0xbfffde88 :     -1073743551 0xbffff941  A  @  @  @  <--------- envp[0]
0xbfffde8c :     -1073743539 0xbffff94d  M  @  @  @  <--------- envp[1]
0xbfffde90 :     -1073743524 0xbffff95c  @  @  @  @  <--------- envp[2]
0xbfffde94 :     -1073743506 0xbffff96e  n  @  @  @  <--------- envp[3]
0xbfffde98 :     -1073743137 0xbffffadf  @  @  @  @  <--------- envp[4]
.................
.................
0xbfffdf24 :               0 0x0         @  @  @  @  <--------- envp is
NULL terminated
0xbfffdf28 :              16 0x10        @  @  @  @
0xbfffdf2c :      1072430079 0x3febfbff  @  @  @  @
.................
.................
0xbffff92c :      1970220641 0x756f2e61  a  .  o  u  <--------- *argv[0]
0xbffff930 :       808583284 0x30320074  t  @  2  0
0xbffff934 :      1241526320 0x4a003030  0  0  @  J
0xbffff938 :      1342205289 0x50006d69  i  m  @  P
0xbffff93c :      1802396012 0x6b6e616c  l  a  n  k
0xbffff940 :      1163089152 0x45535500  @  U  S  E  <--------- *envp[0]
0xbffff944 :      1969765714 0x75683d52  R  @  h  u
0xbffff948 :      1785163361 0x6a676e61  a  n  g  j
0xbffff94c :      1196379136 0x474f4c00  @  L  O  G
0xbffff950 :      1162690894 0x454d414e  N  A  M  E
0xbffff954 :      1635084349 0x6175683d  @  h  u  a
0xbffff958 :         6973294 0x6a676e    n  g  j  @
0xbffff95c :      1162694472 0x454d4f48  H  O  M  E
0xbffff960 :      1869098813 0x6f682f3d  @  @  h  o
0xbffff964 :      1747936621 0x682f656d  m  e  @  h
0xbffff968 :      1735287157 0x676e6175  u  a  n  g
0xbffff96c :      1095762026 0x4150006a  j  @  P  A
0xbffff970 :       792545364 0x2f3d4854  T  H  @  @
0xbffff974 :       796029813 0x2f727375  u  s  r  @
```

threeprocs.c programı iki prosedür çağrısı yapar ve daha sonra stack in en üstündeki 100 öğesini yazdırır.

threeprocs.c kaynak kodu:

```
a(int j, int *k)
{
  char **s;
  int i;

  s = (char **) &s;

  printf("a: &i = 0x%x, &j = 0x%x, &k = 0x%x\n", &i, &j, &k);

  for (i = 0; i < 100; i++) {
    printf("0x%x : %15d 0x%-8x\n", s, *s, *s);
    s++;
  }
}

b(int j)
{
  int i;

  j++;
  i = j+15;

  printf("b: &i = 0x%x, &j = 0x%x\n", &i, &j);

  a(49, &j);
}


main()
{
  int i;

  i = 333;

  b(i);
}
```

32-BCD üzerinde çıktısı da aşağıda bulunmaktadır. Özel bir not; kesin olarak stack yapıları farklı sistemlerde farklı görünür, bu yapılar genel olarak çağrı çerçevesi içindekileri(aşağıdaki gibi func_a,func_b ve func_main) kapsar. Ayrıca, threeprocs.c 64 bit makinelerde çalışmayacaktır.

```
b: &i = 0xbfffea64, &j = 0xbfffea70
a: &i = 0xbfffea44, &j = 0xbfffea50, &k = 0xbfffea54
0xbfffea40 :     -1073747392 0xbfffea40    <-------- func_a:s
0xbfffea44 :               0 0x0           <-------- func_a:i
0xbfffea48 :     -1073747352 0xbfffea68    <-------- old fp
0xbfffea4c :       134513832 0x80484a8     <-------- old pc+4
0xbfffea50 :              49 0x31          <-------- func_a:j
0xbfffea54 :     -1073747344 0xbfffea70    <-------- func_a:k
0xbfffea58 :     -1073747344 0xbfffea70
0xbfffea5c :     -1073747212 0xbfffeaf4
0xbfffea60 :     -1073747304 0xbfffea98
0xbfffea64 :             349 0x15d         <-------- func_b:i
0xbfffea68 :     -1073747320 0xbfffea88    <-------- old fp
0xbfffea6c :       134513864 0x80484c8     <-------- old pc+4
0xbfffea70 :             334 0x14e         <-------- func_b:j
0xbfffea74 :      1073819680 0x40013020
0xbfffea78 :     -1073747304 0xbfffea98
```

```
0xbfffea7c :       134513633 0x80483e1
0xbfffea80 :       134518224 0x80495d0
0xbfffea84 :             333 0x14d         <-------- func_main:i
0xbfffea88 :     -1073747256 0xbfffeac8    <-------- old fp
0xbfffea8c :      1107391881 0x42017589    <-------- old pc+4
0xbfffea90 :               1 0x1           <-------- func_main:argc (not
declared in threeproc.c)
0xbfffea94 :     -1073747212 0xbfffeaf4    <-------- func_main:argv (not
declared in threeproc.c)
0xbfffea98 :     -1073747204 0xbfffeafc    <-------- func_main:envp (not
declared in threeproc.c)
0xbfffea9c :       134513326 0x80482ae
0xbfffeaa0 :       134513936 0x8048510
0xbfffeaa4 :               0 0x0
.........................
.........................
0xbfffeae4 :      1073789204 0x4000b914
0xbfffeae8 :     -1073747220 0xbfffeaec
0xbfffeaec :               0 0x0
0xbfffeaf0 :               1 0x1
0xbfffeaf4 :     -1073743567 0xbffff931    <-------- argv[0]
0xbfffeaf8 :               0 0x0           <-------- argv array terminates
with NULL
0xbfffeafc :     -1073743556 0xbffff93c    <-------- envp[0]
0xbfffeb00 :     -1073743544 0xbffff948    <-------- envp[1]
0xbfffeb04 :     -1073743529 0xbffff957    <-------- envp[2]
.........................
.........................
0xbfffeb98 :               0 0x0           <-------- envp array terminates
with NULL
0xbfffeb9c :              16 0x10
0xbfffeba0 :      1072430079 0x3febfbff
0xbfffeba4 :               6 0x6
0xbfffeba8 :            4096 0x1000
```

Yukarıdaki iki örnek ile, temel çağrı stack yapıları anlaşılmış olmalıdır. Aşağıda, daha karmaşık bir ortamda bu temel yapıları vurgulamak için Assembler 2 dersinden gelen örneklerden biri bulunmaktadır.

```
main()
{
  int *a, a2[3], i;

  i = 6;
  a = &i;
  a2[1] = i+2;
  *a = 200;
  *(a2+2) = i+5;
}
```

- i değişkeni **[fp]** olacak.
- **a2[0]**, **a2[1]** ve **a2[2]** dizi değişkenleri **[fp-12], [fp-8] ve [fp-4]** olacak.
- **a** değişkeni **[fp-16]** olacak.