setjmp()/longjmp()

Setjmp() ve **longjmp(),** C/Unix'in karmaşık akış kontrolünü yapabilmemizi sağlayan program kısımlarıdır.

setjmp() ve **longjmp()**'i anlamamızı sağlayan ince noktalardan biri geçtiğimiz birkaç hafta "assembler" ve "malloc" notlarında tanımladığımız "machine layout" u anlamaktır. Bir programın durumu o programın memory'sinin içeriğine (örn; kod, globals, heap ve stack) ve register'ların içeriğine bağlıdır. Register'ların içeriğinde stack pointer (**sp**), frame pointer (**fp**) ve program counter (**pc**) bulunur. **setjmp()**'ın yaptığı iş register'ların içeriğini, daha sonra **longjmp()** restore edebilsin diye kaydetmektir. Bu yöntemle, **setjmp()** çağırıldığında **longjmp()** programın durumunu döndürür.

Spesifik olarak:

```
#include < setjmp.h >
int setjmp(jmp_buf env);
```

Bu kod registerların şimdiki durumunu **env** içine kaydetmesini söyler.
Eğer **/usr/include/setjmp.h** içine bakarsanız,**jmp_buf** 'ın aşağıdaki şekilde tanımlandığını göreceksiniz:

```
#define _JBLEN  9

typedef struct { int _jb[_JBLEN + 1]; } jmp_buf[1];
```

Bu, **jmp_buf**'ın **_JBLEN+1** tam sayının array'i olduğunu söylemenin pek de uygun olmayan bir yoludur.

Yani **setjmp()**'u çağırmak istediğinizde, onu tamsayılardan oluşan array'in adresi olarak çağırırsınız, ve o bu arraydeki registerların değerini kaydeder. Eğer bu yolla çağırırsanız **Setjmp()**, 0 döndürür.

```
longjmp(jmp_buf env, int val);
```

Longjmp() registerların değerini **env** içine kaydedildiği şekilde yeniden başlatır. ***sp***, ***fp*** *ve* ***pc*** *içerir*. Bu **longjmp()** 'in hiçbir şey döndürmediği anlamına gelir. Böyle yapmak yerine, onu çağıracağınız zaman, sadece **setjmp()**'i çağırmış gibi yapıp kaydedilmiş olan **env**'i çağırarak bir şey döndürebilirsiniz. Bunun nedeni **pc**'nin diğer registerlarla birlikte geri yüklenmesidir (restore edilmesidir). **Setjmp()** , **longjmp()** **'**nin **val** değişkenini döndürür. Bu **val** değişkeni sıfır olamaz (anasayfayı okuyunuz). Dolayısıyla, **setjmp()** sıfır olmayan bir değer dündürdüğünde **longjmp()** in çağırıldığını ve **longjmp()**'nin **setjmp()** döndürdüğünü bilirsiniz.

Örnek olarak, aşağıdaki koda bakınız (in **sj1.c**):

```
#include < setjmp.h >

main()
{
  jmp_buf env;
  int i;

  i = setjmp(env);
  printf("i = %d\n", i);

  if (i != 0) exit(0);

  longjmp(env, 2);
  printf("Does this line get printed?\n");

}
```

Bunu run ettiğimizde, aşağıdaki sonuçları elde ederiz:

```
UNIX> sj1
i = 0
i = 2
UNIX>
```

Yani, ilk olarak, **setjmp()** çağırırız ve o sıfır döndürür. **longjmp()**'i 2 değeriyle çağırırız, bu da kodun **setjmp()**'den 2 değeriyle dönmesine neden olur. Bu değer basılır (yazdırılır) , ve kod exit eder.

Setjmp() ve **longjmp()** genel olarak bir hata belirlendiğinde işlem grubunun uzun string'ini çağırmak için kullanılır, hata üzerinde üst seviye çağrılar yapılarak ince bir şekilde uğraşılabilir. Örneğin, **sj2.c** 'ye bakınız. Karmaşık gibi görünebilir, fakat gerçekten öle değildir. Olan biten şudur ki, **proc_4** vasıtasıyla -- **proc_1** karmaşık işlem grubu çağrıları serisi vardır. Eğer **proc_4**'ün değişkeni sıfırsa, o **longjmp()** çağırarak hata verir. Aksi takdirde, her şey normal bir şekilde işler. Gördüğünüz gibi, eğer **sj2** yi pozitif değişkenlerle çağırırsanız her şey yolundadır. Buna rağmen, eğer onları sıfır olarak çağırırsanız, o **longjmp()** çağırır ve hata verir:

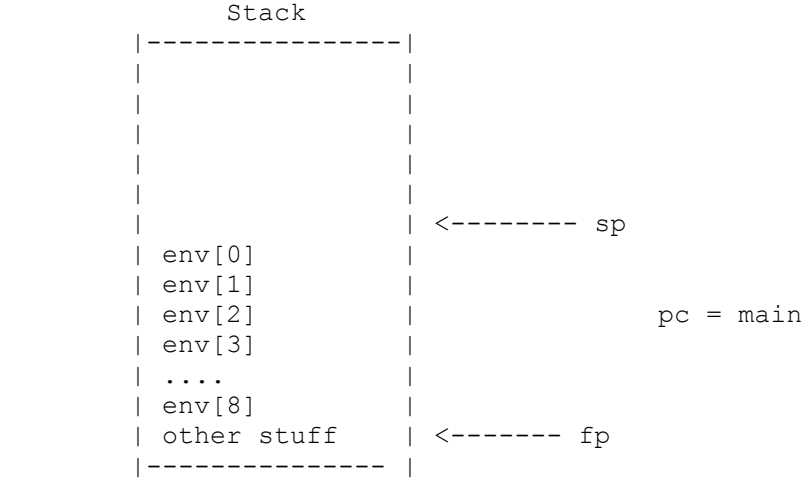
```
UNIX> sj2 1 2 3 4
proc_1(1, 2, 3, 4) = 4
UNIX> sj2 0 0 0 0
Error -- bad value of i (0), j (0), k (0), l (0)
UNIX>
```

Şimdi, **setjmp(), sp** ve **fp** de dahil olmak üzere bütün register'ları kaydeder. Bunun anlamı, eğer **setjmp()** döndüren bir işlem grubu döndürürseniz, bu **setjmp()**'nin **env** arabelleği geçerliliğini kaybeder. Neden? Çünkü **env** arabelleği çağıran işlem grubunun sp ve fp'sini içerir. Eğer bu işlem grubu dönerse, **sp** ve **fp yi** restore ettiğinizde, stack önceki durumundan farklı bi state'te olacaktır, hata alırsınız. Örneğin **sj3.c** 'e bakınız.

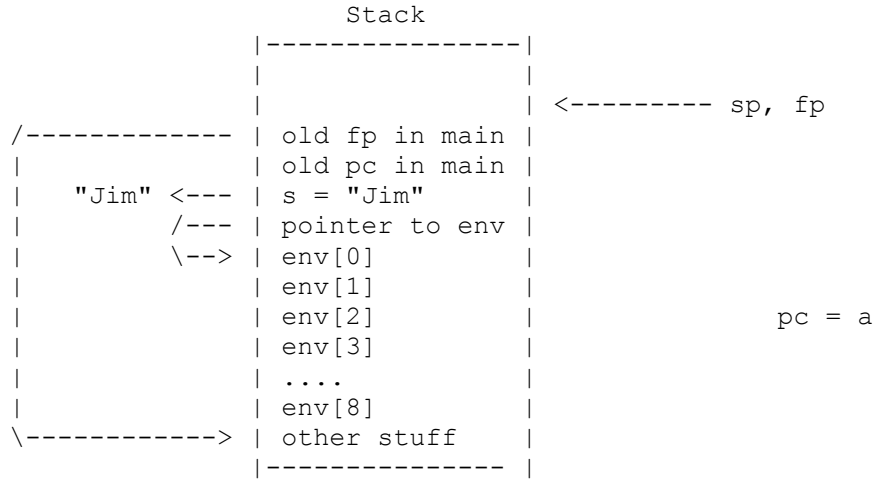
Bunu çalıştırdığımızda, aşağıdakini elde ederiz:

```
UNIX> sj3
Setjmp() returned -- i = 0
s = Jim
In B: i=3.  Calling longjmp(env, i)
Setjmp() returned -- i = 3
Segmentation fault (core dumped)
UNIX>
```

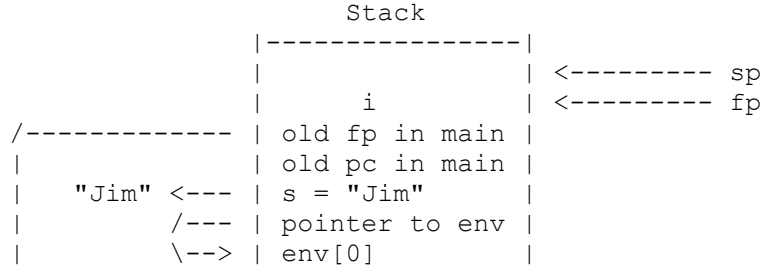
Yani, tam olarak ne olmaktadır? **main()** parçası ilk olarak çağırıldığında stack aşağıdaki gibi görünür:

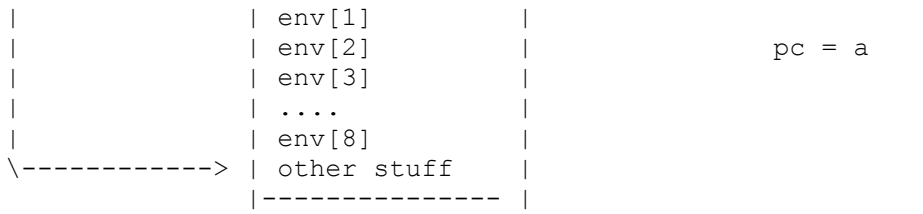


Şimdi, **main(), a()**'yı çağırır. İlk olarak o sonuçları stack'e ters sırada sürer, sonra **jsr** çağırılır, bu stack üzerinde **pc** eski **fp'**nin dönmesini tetikler. **fp** ve **sp, a()**'ya boş stack çerçevesi oluşturmak için değişir.

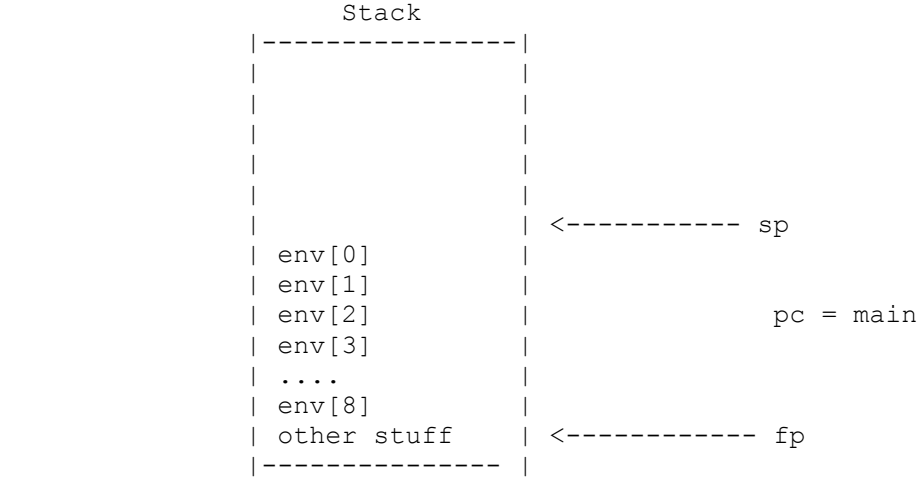


a()'nın yaptığı şey, yerel değer i için oda ayırmaktır:

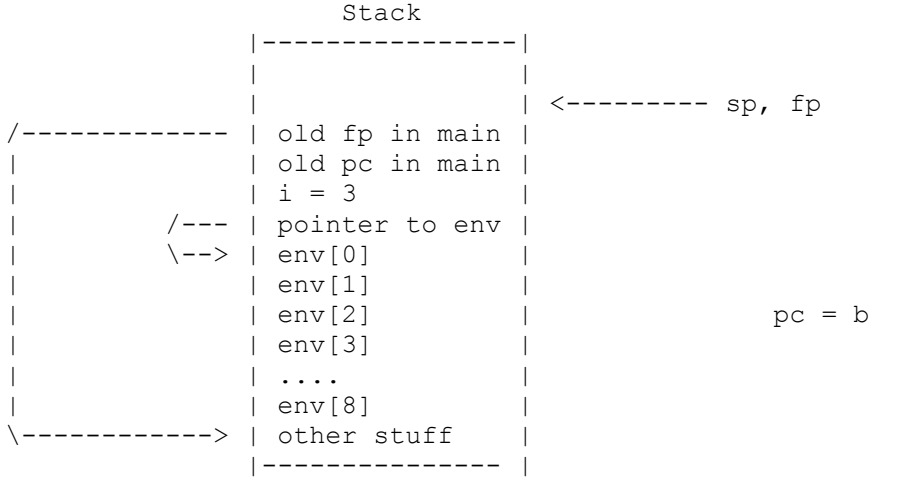




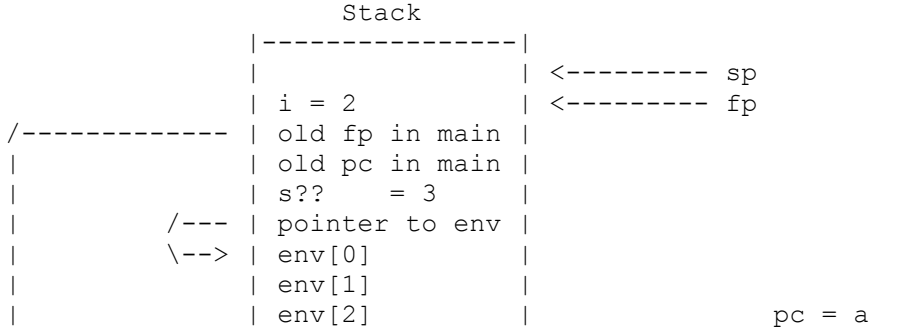
Sonra o **setjmp()** çağırır.Bu registerların şimdiki durumlarını kaydeder. Başka bir deyişle, **sp**, **fp**, ve **pc** nin o an ki değerlerini kaydeder. Şimdi, **a(),** "**i = 0**" ve "**s = Jim**" yazdırır, ve sonra **main()** kısmına geri döner**. a()** çağırıldığında **env**'nin machine durumuna göre başlatılması haricinde şimdi stack eskisi gibi görünür.

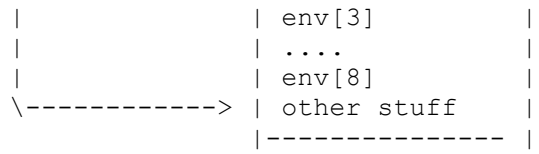


Şimdi, **main()**, **b()** 'yi çağırır, stack aşağıdaki gibi görünür:



Sonra **longjmp()** çağırılır. **a()**, **setjmp()**'yı çağırdığında, registerlar değerlerine restore edilir ve **setjmp()** içindeki **a()**dan **pc** döner**.** Aksi takdirde stack içindeki değerler eskiden **b()** için olduklarıyla aynı olur:





Problemi görüyor olmalısınız. Stack kötü bir durumda. Daha ayrıntılı incelersek, **a(), s**'in olduğu yerde **(char *)** olmasını bekler ancak bunun yerine bir tamsayı değeri olan 3 bulunur. Bu yüzden, o **s**'i yazdırmak istediğinde, memory yeri 3te olan bir string bulmaya çalışır ve çekirdeği dump eder.

setjmp() ve **longjmp()** 'de çok yaygın bir bug'dır**.** Onları uygun bir şekilde kullanmak için ***setjmp()*** ÇAĞIRAN BİR İŞLEM GRUBUNDAN DÖNEMEZSİNİZ. Gördüğünüz gibi bu bug çok ustaca, **b() için olan stack frame i daha çok a()** içinmiş gibi görünmektedir ve bu yüzden bir süre fark edilemeyebilir.

Setjmp() ve İşaretler

setjmp() ve **longjmp()** hakkındaki güzel şeylerden biri şudur ki, sinyal işleyiciden **longjmp()** çıkarabilir ve programınıza geri alabilirsiniz. Bu, sinyalleri yeniden yakalamınızı sağlar.

sh4.c'a bakınız:

Dikkat ediniz ki bu program **alarm_handler**'dan **longjmps** yapar ve 8 saniye sonra "**Gave up**" yazdırır. Emin olunuz ki bu yöntemle farklı şeyler yapabilirsiniz.

---

png.c

Ok – program içinde

png.c ye bakınız (which stands for "prime number generator"),gerçekten çok cool şeyler var. Onu kopyalayın ve çalıştırın (you'll have to exit with **CNTL-C**),sonra neler olup bittiğini anlayabiliyomusunuz görün. Bu program çalışmıyo da olabilir bunu bilin. Neden olduğunu çıkarabildiniz mi? Şansınıza küsün ki, bu programın üzerine gitmek için yeterli vaktim yok, fakat onu anlayıp anlamadığınıza kendiniz bakmanız iyi bir egzersiz olur.

Bariz bir şekilde, buna benzer bir şey uygulamak için işi parçalara ayırmanız daha iyi olur.