WAIT()

Sınıfta wait() hakkında çok fazla ayrıntıya girmeyeceğim. Man sayfasından (‘man –s2wait’)yaparak ve kitaptan okuyun. Basitçe, wait(int *statusp) herhangibir alt prosesin bitmesi için bekler. Bu yapıldığında ,wait() çocuk prosesin pid’ sini geri döndürür, wait()’e geri dönüş değeri olarak ve çocuk prosesin bitirilip bitirilmediğinin bilgisini içeren statusp’ yi başlatır. Statusp’ yi incelemek için /usr/include/sys/wait.h makroları kullanabilirsin.

Örnek olarak; forkwait.c ,forkwait2.c ve forkwait3.c programlarına bakın. Onlar çocuk prosesin fork’tan nasıl çıktığını ,çocuk proseslerin çıkması için bekleyen gösteren örneklerdir ve prosesin çıktığını görmek için statusp’ yi inceler. Bunların hepsi basittir.

Forkwait3 programında “kill” komutu kullanılarak bir sinyalle çocuk proses öldürülmelidir.Örneğin:

```
UNIX> forkwait3 &
[1] 22326
UNIX> Child (22327) doing nothing until you kill it
Kill the child with 'kill -9 22327'  or just 'kill 22327'
```

Şimdi çocuk prosesi manuel olarak “kill -9 22327” yazıp göndererek kesin öldürme sinyali ile (signal number 9) ya da “kill 22327” yazarak signal 15 gönderilir. İkisinide dene:

```
UNIX> forkwait3 &
[1] 22326
UNIX> Child (22327) doing nothing until you kill it
Kill the child with 'kill -9 22327'  or just 'kill 22327'

(hit return a few times)
UNIX> kill -9 22327
UNIX> Parent: Child done.
  Return value: 22327
  Status:       9
  WIFSTOPPED:   0
  WIFSIGNALED:  1
  WIFEXITED:    0
  WEXITSTATUS:  0
  WTERMSIG:     9
  WSTOPSIG:     0
```

Tekrar dene:

```
UNIX> forkwait3 &
[1] 22328
UNIX> Child (22329) doing nothing until you kill it
Kill the child with 'kill -9 22329'  or just 'kill 22329'

UNIX> kill 22329
```

```
UNIX> Parent: Child done.
  Return value: 22329
  Status:       15
  WIFSTOPPED:   0
  WIFSIGNALED:  1
  WIFEXITED:    0
  WEXITSTATUS:  0
  WTERMSIG:     15
  WSTOPSIG:     0
```

Forkwait3a.c genel bir segmentasyon hatası yapan bir çocuk prosese sahiptir ve biz sinyal 11 ile sonlanan çocuk prosesin ebeveynini tanıyabilir ve görebiliriz. Başka bir derste sinyalleri detaylı olarak göreceğiz.

Tamam. Şimdi forkwait4.c ye bakalım. Acil çıkış yapan çocuk proses ve 4 saniye bekleyen ebeveyn prosesin olduğu bir durumdan "ps x" komutunun çıktısı ile yazdırılır ve wait()sinyali çağrılır. Bu sırada ebeveynin sistem çağrılarına("ps x") açık olması gerekir ve çocuk proses çıkış yapar. Böylece,çocuk proses için "ps x" komutunda herhangi bir listeleme yapılmaz, muhtemelen wait() ile sonsuza dek bekleyebilir çocuk proses tamamlana kadar. Ancak bu durum böyle değildir.

Ne zaman bir çocuk proses çıkarsa,onun prosesleri "zombie" olarak adlandırılır ta ki onun ebeveyn prosesi ölürse yada onun için wait()ile çağrılırsa. Biz "zombi" dediğimizde onun için kaynak olmadığını ve çalıştırılamadığını anlarız fakat şimdi işletim sistemi tarafından muhafaza edilir ve böylece ebeveyn wait()'i çağırdığında çocuk doğru bilgi alacak.Forkwait4' ün çıktılarına bak:

```
UNIX> forkwait4
Child (1624) calling exit(4)
  PID TT STAT  TIME COMMAND
...
  381 p2 S     0:02 -sh (csh)
 1623 p2 S     0:00 forkwait4
 1624 p2 Z     0:00
 1625 p2 S     0:00 sh -c ps x
 1626 p2 R     0:00 ps x
...

Parent: Child done.
  Return value: 1624
  Status:       1024
  WIFSTOPPED:   0
  WIFSIGNALED:  0
  WIFEXITED:    1
  WEXITSTATUS:  4
  WTERMSIG:     0
  WSTOPSIG:     4
UNIX> ps x
...
  381 p2 S     0:02 -sh (csh)
```

1627 p2 R 0:00 ps x
...

1624 nolu proses bir zombie proses olup sermaye Z ile “ps x” komutunun çıktısını ifade eder.Forkwait4 (proses 1623) wait()’i çağırdığında ,proses boyunca gider.

Ebeveyn wait()’i çağırmadan çıkarsa ne olur? Sonra çocuk zombi proses /sbin/init ‘e ebeveynlik transferi gerekir. Bunun yerine çocuk basitçe uzaklaştırılır.

Ne zaman bir çocuk çıkış yapsa wait() döndürülür.Bir proseste birden fazla çocuk varsa , belirli bir çocuk beklemek için zorla wait() yaptırılamaz.
Basitçe yapılabilir, daima ilk çocuk çıkar. Örneğin multichild.c’ye bak.Bu program 4 çocuk prosese çatallar ve 4 zaman wait()’i çağırır.Çocuklar zamanın rastgele periyodlarında uyur ve sonra çıkar. Gördüğünüz gibi ilk çocuğa ilk wait() çağrısı döndürüldü:

```
UNIX> multichild
Fork 0 returned 14160
Fork 1 returned 14161
Fork 2 returned 14162
Fork 3 returned 14163
Child 1 (14161) exiting
Wait returned 14161
Child 3 (14163) exiting
Wait returned 14163
Child 0 (14160) exiting
Wait returned 14160
Child 2 (14162) exiting
Wait returned 14162
UNIX>
```

Şimdi, sen belirli bir prosesi beklemek için waitpid() kullanabilirsin ayrıca belirli proses çıkış yapmadıkça daima onu döndürebilirsin.Ben şahsen waitpid() kullanmanın kötü bir form olduğunu düşünüyorum ve kesinlikle kullanılan versiyonlarda anında döndürür gerçekten kötü bir formdur.

Sen hiçbir şekilde wait() kullanmaya izin verme senin jsh laboratuarında.Eğer siz waitpid() kullanırsanız NOHANG setiyle TA’ ya acımasız olması için talimat vereceğim.

Execve()

Yeni süreçlerin nasıl oluşturulacağı son derste gördük. -- fork() fonksiyonunu kullanarak. Bu derste execve() fonsiyonunun nasıl kullanılacağını görüceğiz.

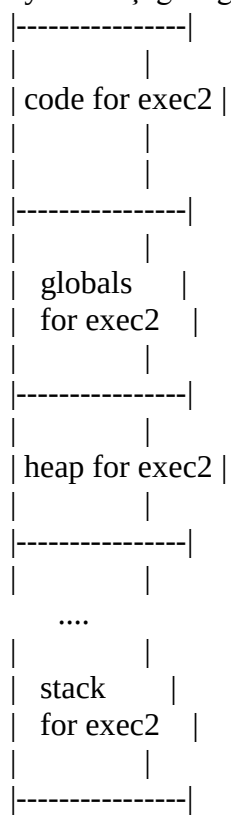
Execve() basit bir kavramdır:

```
int execve(char *path, char **argv, char **envp);
```

Execve () bu yolu yürütülebilir bir dosyanın adı olarak varsayar.. Argv boş olarak sonlandırılmış dizelerden oluşan bir string ifadelerdir, son elemanı NULL olduğu gibi, ve envp boş olarak sonlandırılmış dizelerin bir başka boş sonlandırılmış dizileridir.
Bu argv argümanlarla yolda dosyayı çalıştırır böylece execve () geçerli işlemi yazar, ile envp içinde çevre değişkenleri Execve() bir hata ile karşılaşmadığı sürece geriye değer döndürmez, böyle bir dosya yol üzerinde mevcut değil, veya çalıştırılabilir dosya yok mesajı yollar.
Bu kafa karıştırıcı görünebilir. Niçin execve() fonksiyonu geri dönmez? exec2.c örneğini inceleyelim:

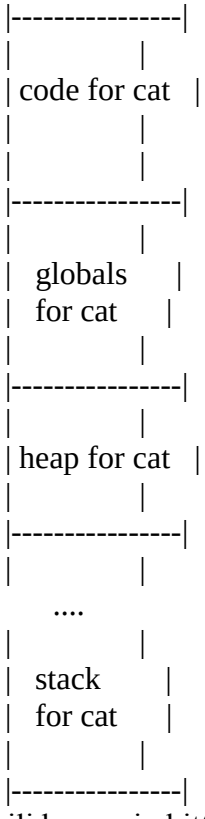
```
#include

main(int argc, char **argv, char **envp)
{
  char *newargv[3];
  int i;

  newargv[0] = "cat";
  newargv[1] = "exec2.c";
  newargv[2] = NULL;

  i = execve("/bin/cat", newargv, envp);
  perror("exec2: execve() failed");
  exit(1);
}
```

exec2 programını derlediğimizi varsayalım. Ardından argümansız çalıştırırız.
execve() fonsiyonunu çağırdığımız zaman bellek durumu aşağıdaki gibi olur:



Şimdi, execve() fonsiyonu çağrıldı. Bu bir sistem çağrısıdır ve şöyle işler:" /bin/cat dosyası içinden programı çalıştır." Argümanlarla birlikte "cat" "exec2.c". execve() fonsiyonu

tamamlandığında,bellek durumu değişmiş olacak, bu yüzden main() fonsiyonunda argv ve argc'yi düzgün bir şekilde ayarlayalım.



exec2.c ile ilgili her şeyin bittiğini fark ediceksiniz. Bunun nedeni, bellek durumu cat'i çalıştırmak için üzerine yazmasından dolayıdır.Sol tarafta exec2 ile ilgili hiçbir iz yokturBu execve() fonsiyonunun hatasız çalışması durumunda neden geri dönmediğinin sebebidir -- the state to which it might have returned has been overwritten. O tamamiyle bitti. cat işlemi sonlandığında process yok edilir.

Kabukta bu cat'i çalıştırdığımızda nasıl oluyor da geriye döüyormuş gibi oluyor? Kabuk çağrıları yüzünden: fork - exec - wait.

execve() fonsiyonunun 6 tane kullanım çeşitleri vardır—Ben bunları aşağıda özetlemeye çalıştım:

execl(char *path, char *arg0, char *arg1, ..., char *argn, NULL): Bu geçerli envp kullanır, ve Eğer parametre olarak argv belirtmek yerine, işaretçiler bir dizi oluşturmayı sağlar. Tam olarak yolu belirtmeniz gerekir.

execv(char *path, char **argv): Bu aynen execve fonsiyonu gibidir, fakat sizin geçerli envp 'nizi kullanır.

execle(char *path, char *arg0, char *arg1, ..., char *argn, NULL, char **envp): Bu aynen execl versiyonu gibidir, fakat bunda tam yolu belirtmeniz gerekir.

execve(char *path, char **argv, char **envp).

execlp(char *path, char *arg0, char *arg1, ..., char *argn, NULL): Bu da aynen execl versiyonu gibidir, tek farkı göreceli bir dosya olup olmadığı PATH değişkeni içerisinde sürekli kontrol edilir.

execvp(char *path, char **argv): Bu aynen execv versiyonu gibidir, tek farkı PATH değişkeni yolu bulmak için aranır.

Bunların içerisinde execvp() ve execlp() daha kullanışlıdırlar. Bunların hepsi execve() fonksiyonunun en üstünde uygulanıralar.

Eğer execve() fonksiyonu doğru bir şekilde çalışmaz ise -1 değerini döndürür.Örneğin;exec1.c uygulamasını inceleyelim.

Bu program var olmayan "./cat" dosyayı çalıştırmaya çalışsın.Böylece execve() fonsiyonu hata verir, ve perror çalışır.

DR. PLANK'S CARDINAL SIN OF EXEC

Asla execve() fonsiyonunu veya türevlerini geri dönüş değerinin olup olmadığına bakmadan çağırmayın!!!!!!!

---

execcat*x*.c programlarını inceleyelim.

execcat1.c kapalı 3 prosses çatallanmasının hepsi "cat f1" de çalıştırılır. execvp() fonksiyonu kullanırken bir ortam değişkeni gerekmez, ve "cat" i bulmak için PATH içinde arama yapar.

```
UNIX> execcat1
This is file f1
This is file f1
This is file f1
UNIX>
```

Şimdi, execcat2.c execvp() yerine execv(). Çalıştırdığınızda, görünürde hiçbirşey olmuyor:

```
UNIX> execcat2
UNIX>
```

Peki ne oluyor? execv() çağırımı başarısız . Bunun anlamı Bu exec çağrı i = -1 ile döner ve daha sonra child prosses devam ediyor. O da döngü ve fork() çağrısı ile gidecek. Açıklamaya ek olarak, execcat3.c uygulamasına bakalım, Her fork () çağrılmasından önce hangi j değeri ve işlem numarasını yazdırır:

```
UNIX> execcat3
I am 4794.  j = 1
I am 4795.  j = 2
I am 4796.  j = 3
```

I am 4795. j = 3
I am 4794. j = 2
I am 4799. j = 3
I am 4794. j = 3
UNIX>
Görüldüğü gibi, fork() 7 kere çağrıldı, 3 kere değil, çünkü execv çağrısı başarısız olduğu sürece döngü devam eder. Bu j = 3 için kötü değildir, ancak 3 10 idi, sonra fork () 1023 kez olacağını söyledi. (i.e. fork çağrıldı *2^n-1* kere eğer n=3 ise). Bu yıkıcı olabilir. Hatayı düzeltmek için; execcat4.c' deki gibi execv()' nin geri dönüş değerini kontrol ederiz:

UNIX> execcat4
execcat4: No such file or directory
execcat4: No such file or directory
execcat4: No such file or directory
UNIX>