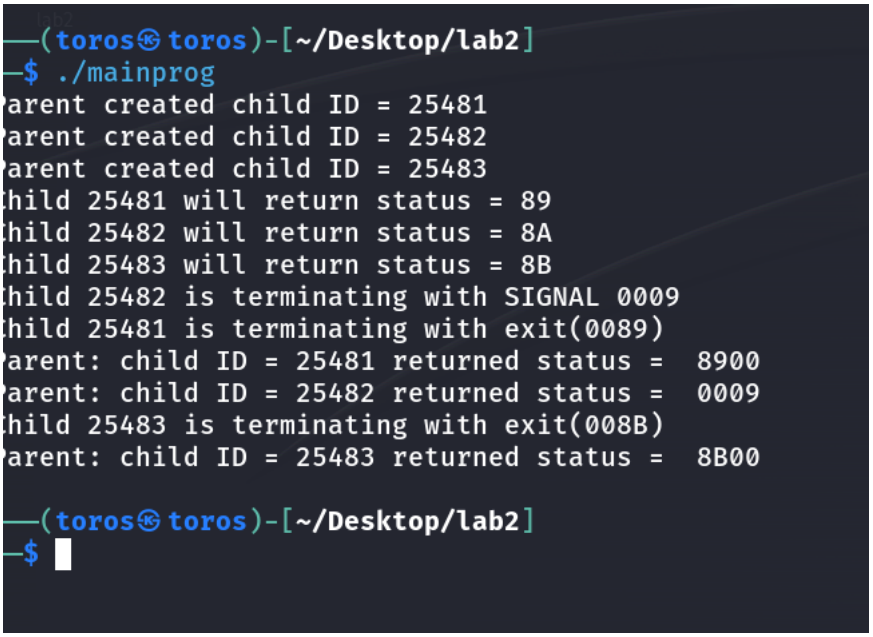
1.1)

1.2)



mainprog, 3 kez çocuk süreç oluşturan bir ebeveyn süreçtir. Bu çocuk süreçler, child adlı başka bir programa geçiş yapar ve her biri bir indeks değeri ile çağrılır. Bu indeks değeri, çocuk süreçlerin kaçıncı kez oluşturulduğunu temsil eder.

fork(): Bu sistem çağrısı, ebeveyn süreç tarafından çağrıldığında, yeni bir çocuk süreç oluşturur. Ebeveyn süreç, çocuk süreç oluşturulduktan sonra ve çocuk süreç, **execl()** veya benzeri bir çağrı ile başka bir programa geçtikten sonra eş zamanlı olarak devam eder.

execl(): Bu sistem çağrısı, belirtilen programı çalıştırmak için kullanılır. Mainprog içindeki çocuk süreçler, bu çağrıyı kullanarak **child** adlı başka bir programa geçerler. Bu durumda, child programına indeks değeri olarak bir argüman eklenir.

wait(): Bu sistem çağrısı, ebeveyn sürecin çocuk süreçlerinin tamamlanmasını beklemesini sağlar. Ebeveyn süreç, bu çağrı ile bir çocuk sürecin tamamlanmasını bekler ve çocuk sürecin çıkış durumuyla ilgili bilgileri alır. Bu durumda, wait**()** döngü içinde kullanılarak tüm çocuk süreçlerin tamamlanması beklenir.

Fork, ebeveyn süreç tarafından çağrıldığında yeni bir çocuk süreç oluşturulur. Ebeveyn süreç, çocuk süreç oluşturduktan sonra genellikle wait**()** çağrısı ile çocuk sürecin tamamlanmasını bekler. Bu şekilde, ebeveyn süreç ve çocuk süreçlerin eşzamanlı olarak çalışmasını sağlar.

1.3)

```
┌──(toros㉿toros)-[~/Desktop/lab2]
└─$ ./mainprog
Parent created child ID = 25638
Parent created child ID = 25639
Parent created child ID = 25640
Child 25638 will return status = 26
Child 25640 will return status = 28
Child 25639 will return status = 27
Child 25639 is terminating with SIGNAL 0009
Parent: child ID = 25639 returned status =  0009
Child 25640 is terminating with exit(0028)
Parent: child ID = 25640 returned status =  2800
Child 25638 is terminating with exit(0026)
Parent: child ID = 25638 returned status =  2600
```

Ana sürecin çocuk süreçlerinin bitmesini beklemeden hemen sona erebileceğini gözlemleyebilirsiniz. Bu, wait() sistem çağrısını kaldırmak, yani ana sürecin çocuk süreçlerini beklemeden bağımsız olarak sonlanmasına neden olabilir.

2.2)

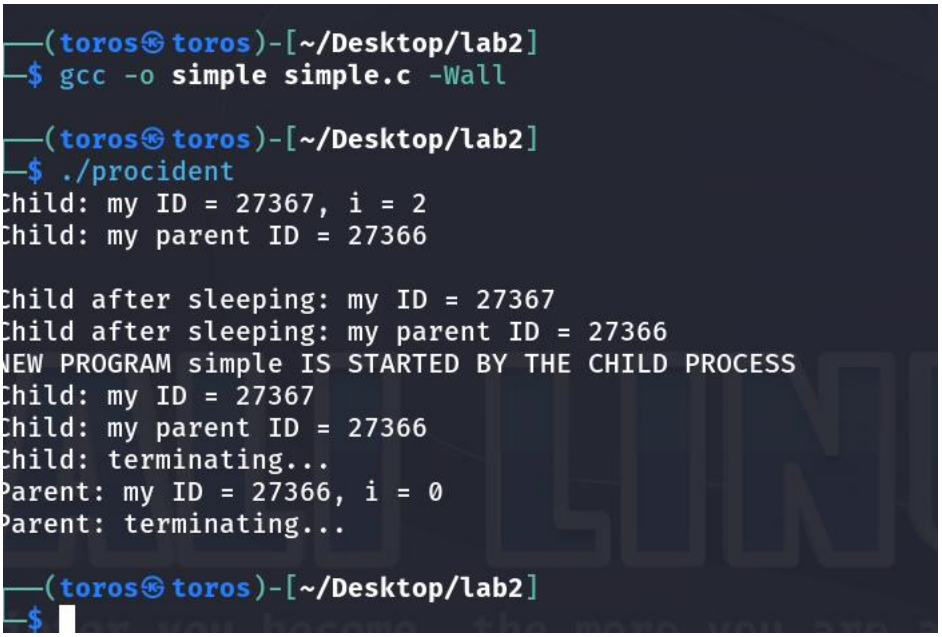
```
┌──(toros㉿toros)-[~/Desktop/lab2]
└─$ ./procident
Child: my ID = 25938, i = 2
Child: my parent ID = 25937
Parent: my ID = 25937, i = 0
Parent: terminating...

┌──(toros㉿toros)-[~/Desktop/lab2]
└─$
```

Programın çalışması sırasında, çocuk süreci (child) genellikle ebeveyn sürecinden (parent) önce tamamlanacaktır. Bunun nedeni, çocuk sürecin 10 saniyelik bir uyku (sleep(10)) süresine sahip olmasıdır. Ebeveyn süreç ise sadece 3 saniye uyuduktan sonra kendi işlemlerini tamamlayacaktır.

Çocuk sürecin farklı parent süreçlere sahip olmasının nedeni, execl("simple", "simple", 0) çağrısı nedeniyledir. Bu çağrı, çocuk süreci tarafından kullanıldığında, çocuk sürecin kendisini başka bir programa (simple) değiştirmesini sağlar. Bu nedenle, çocuk sürecin yeni bir programın süreç kimliğine sahip olması beklenir.

Çocuk sürecin programın sonlandığını görmemesi nedeniyle program sonlanmadan UNIX prompt görünür. Çocuk sürec, execl("simple", "simple", 0) çağrısı ile kendisini başka bir programa değiştirdiğinde, ebeveyn süreç bu değişikliği fark etmez ve kendi işlemlerini sürdürür. Bu nedenle, çocuk sürecin yeni programın sonlandığını görmemesi ve bu süreç tamamlanana kadar UNIX prompt hemen görünmez.

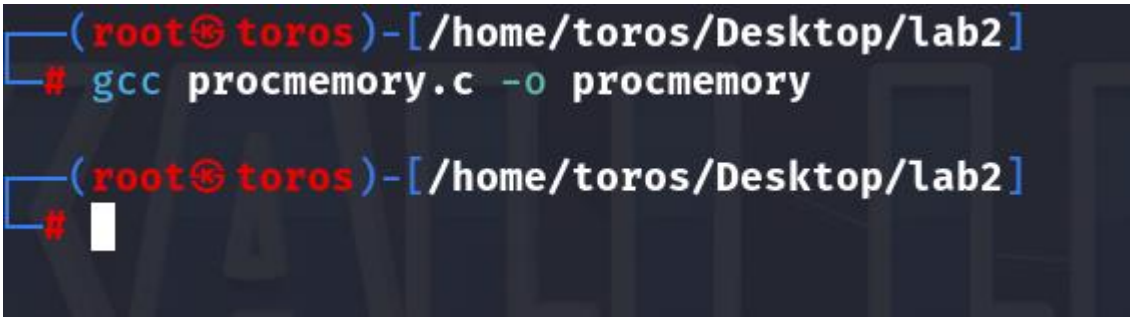
2.3)



Çocuk süreci (simple.c'yi çalıştıran süreç) artık 3 saniye boyunca uyuyacak, bu nedenle ebeveyn sürecinden (procident.c) önce tamamlanacaktır.

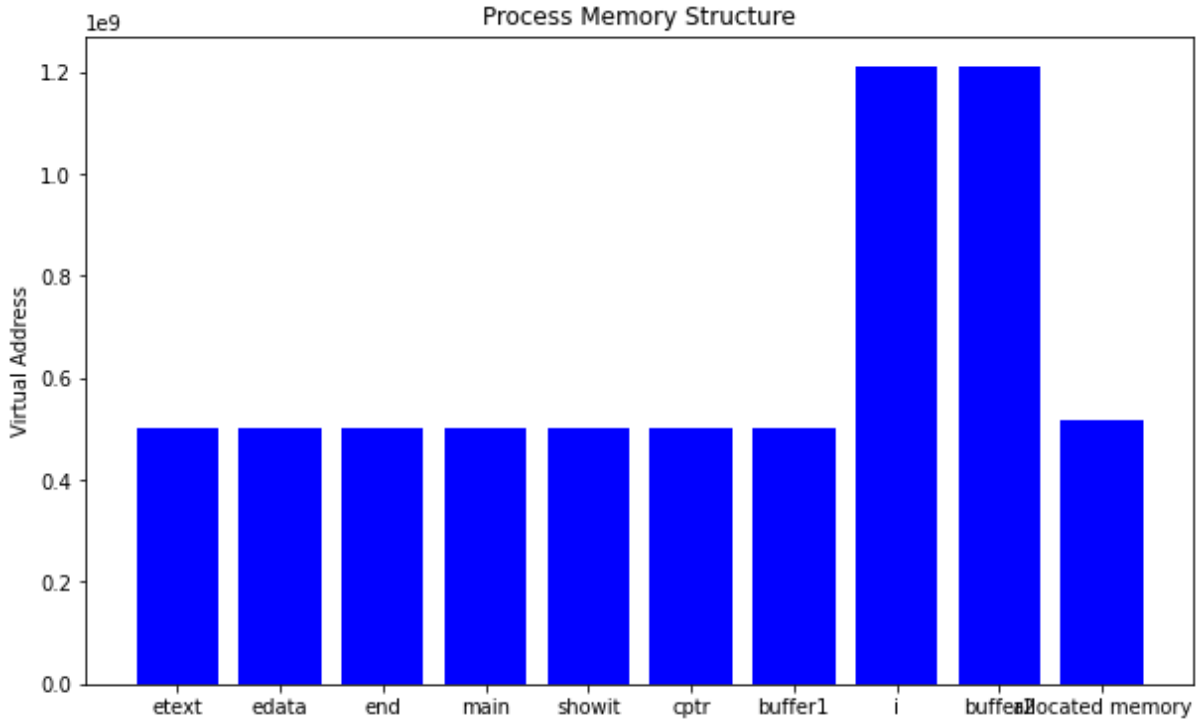
Ebeveyn süreci, çocuk sürecinden önce sonlanacak ve bu durum ebeveyn sürecin PID değerini etkilemeyecektir. PID değeri, sürecin kimliğini belirtir ve süreç sona erdiğinde bu kimlik serbest bırakılmaz veya değiştirilmez.

3.1)



3.2)

```
┌──(root㉿toros)-[/home/toros/Desktop/lab2]
└─# ./procmemory

Address etext: 1DDFF419
Address edata: 1DE02058
Address end  : 1DE02090
ID main          is at virtual address: 1DDFF1A9
ID showit        is at virtual address: 1DDFF345
ID cptr          is at virtual address: 1DE02050
ID buffer1       is at virtual address: 1DE02070
ID i     is at virtual address: 48189EBC
A demonstration
ID buffer2       is at virtual address: 48189E98
Alocated memory at 1EBD96B0
This message is output by the function showit()
```



3.3)

```
┌──(root㉿toros)-[/home/toros/Desktop/lab2]
└─# gcc procmemory.c -o procmemory

┌──(root㉿toros)-[/home/toros/Desktop/lab2]
└─# ./procmemory

ddress etext: 5E7D742D
ddress edata: 5E7DA060
ddress end  : 5E7DA090
D main          is at virtual address: 5E7D71B9
D showit        is at virtual address: 5E7D735A
D cptr          is at virtual address: 5E7DA058
D buffer1       is at virtual address: 5E7DA070
D i     is at virtual address: 6464D2FC
 demonstration
D buffer2       is at virtual address: 6464D2D8
located memory at 5ECD56B0
his message is output by the function showit()


ddress etext: 5E7D742D
ddress edata: 5E7DA060
ddress end  : 5E7DA090
D main          is at virtual address: 5E7D71B9
D showit        is at virtual address: 5E7D735A
D cptr          is at virtual address: 5E7DA058
D buffer1       is at virtual address: 5E7DA070
D i     is at virtual address: 6464D2FC
 demonstration
D buffer2       is at virtual address: 6464D2D8
located memory at 5ECD56B0
his message is output by the function showit()
┌──(root㉿toros)-[/home/toros/Desktop/lab2]
└─#
```

fork() fonksiyonu çağrıldığında, yeni bir süreç oluşturulur ve bu yeni süreç, çağrıldığı sürecin tam bir kopyasıdır. her iki süreç de aynı sanal adres uzayına sahiptir, ancak kendi bağımsız bellek kopyalarına sahiptirler. her iki süreçte de i değişkeni aynı sanal adresi paylaşacak ancak kendi değerini tutacaktır. Bu durum, eğer bir süreç i'yi değiştirirse, bu değişiklik diğer süreci etkilemeyecek ve her iki süreç bağımsız olarak devam edecektir.