Öğrenci Adı – Soyadı: _____
Öğrenci Numarası: _____

S1	*S2*	*S3*	*S4*	*S5*	*Toplam*

Hacettepe Üniversitesi
Bilgisayar Mühendisliği Bölümü

BİL 220
Sistem Programlamaya Giriş
2. Ara Sınav

Tarih: 3 Mayıs 2012
Süre: 110 dak.

Sınava başlamadan önce aşağıda yazılanları mutlaka okuyunuz!

- Bu sınav **kapalı kaynak** bir sınavdır. Yani sınav süresince ilgili ders kitapları veya ders notlarınızdan faydalanmanız yasaktır.
- Size yardımcı olması açısından sonraki 2 sayfada bazı Intel IA32/x86-64 Assembly komutlarının söz dizimleri ve diğer bazı ilgili tanımlar verilmiştir.
- **Sınavda kopya çekmek yasaktır.** Kopya çekmeye teşebbüs edenler hakkında ilgili idare işlemler **kesinlikle** başlatılacaktır.
- Her bir sorunun toplam ağırlığı soru numarasının ardında parantez içinde belirtilmiştir.
- Sınav toplam 110 puan üzerinden değerlendirilecektir.

Sınav bu kapak sayfası dahil toplam 13 sayfadan oluşmaktadır. Lütfen kontrol ediniz!

BAŞARILAR!

Sıçrama İşlemleri

Sıçrama	Koşul
`jmp`	`1`
`je`	`ZF`
`jne`	`~ZF`
`js`	`SF`
`jns`	`~SF`
`jg`	`~(SF^OF)&~ZF`
`jge`	`~(SF^OF)`
`jl`	`(SF^OF)`
`jle`	`(SF^OF)|ZF`
`ja`	`~CF&~ZF`
`jb`	`CF`

Aritmetik İşlemler

Format		İşlem
`addl`	*Src,Dest*	Dest = Dest + Src
`subl`	*Src,Dest*	Dest = Dest - Src
`imull`	*Src,Dest*	Dest = Dest * Src
`sall`	*Src,Dest*	Dest = Dest << Src
`sarl`	*Src,Dest*	Dest = Dest >> Src
`shrl`	*Src,Dest*	Dest = Dest >> Src
`xorl`	*Src,Dest*	Dest = Dest ^ Src
`andl`	*Src,Dest*	Dest = Dest & Src
`orl`	*Src,Dest*	Dest = Dest | Src
`incl`	*Src*	Dest = Dest + 1
`decl`	*Src*	Dest = Dest - 1
`negl`	*Src*	Dest = - Dest
`notl`	*Src*	Dest = ~ Dest

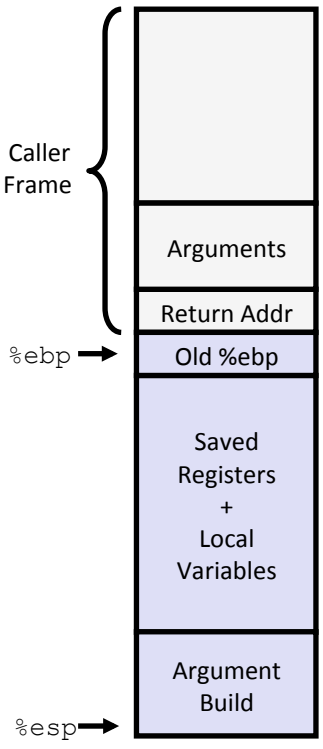
Bellek İşlemleri

Format	İşlem
(Rb, Ri)	Mem[Reg[Rb]+Reg[Ri]]
D(Rb, Ri)	Mem[Reb[Rb]+Reg[Ri]+D]
(Rb, Ri, S)	Mem[Reg[Rb]+S*Reg[Ri]]

Yazmaçlar (Registers)

63	31	15	8	7	0	
`%rax`	`%eax`	`%ax`	`%ah`	`%al`		Return value
`%rbx`	`%ebx`	`%bx`	`%bh`	`%bl`		Callee saved
`%rcx`	`%ecx`	`%cx`	`%ch`	`%cl`		Argument #4
`%rdx`	`%edx`	`%dx`	`%dh`	`%dl`		Argument #3
`%rsi`	`%esi`	`%si`		`%sil`		Argument #2
`%rdi`	`%edi`	`%di`		`%dil`		Argument #1
`%rbp`	`%ebp`	`%bp`		`%bpl`		Callee saved
`%rsp`	`%esp`	`%sp`		`%spl`		Stack Pointer
`%r8`	`%r8d`	`%r8w`		`%r8b`		Argument #5
`%r9`	`%r9d`	`%r9w`		`%r9b`		Argument #6
`%r10`	`%r10d`	`%r10w`		`%r10b`		Reserved
`%r11`	`%r11d`	`%r11w`		`%r11b`		Used for linking
`%r12`	`%r12d`	`%r12w`		`%r12b`		Callee saved
`%r13`	`%r13d`	`%r13w`		`%r13b`		Callee saved
`%r14`	`%r14d`	`%r14w`		`%r14b`		Callee saved
`%r15`	`%r15d`	`%r15w`		`%r15b`		Callee saved

Linux Yığıt (Stack) Yapısı



Özel Hizalama Durumları (Intel IA32)

1 byte: `char`, ...
sınırlandırma yok

2 bytes: `short`, ...
en düşük bit adresi 0_2

4 bytes: int, float, char *, ...
e*n düşük 2 bit adresi* 00_2

8 bytes: `double`, ...
Windows: e*n düşük 3 bit adresi* 000_2
Linux: e*n düşük 2 bit adresi* 00_2

12 bytes: `long double`
Windows & Linux:
e*n düşük 2 bit adresi* 00_2

C Veri Tipi	IA-32	X86-64
`char`	1	1
`short`	2	2
`int`	4	4
`long`	4	8
`long long`	8	8
`float`	4	4
`double`	8	8
`long double`	10/12	10/16
`pointer`	4	8

Özel Hizalama Durumları (Intel x86-64)

1 byte: `char`, ...
sınırlandırma yok

2 bytes: `short`, ...
en düşük bit adresi 0_2

4 bytes: `int`, `float`, ...
e*n düşük 2 bit adresi* 00_2

8 bytes: `double`, `char *`, ...
Windows & Linux:
e*n düşük 3 bit adresi* 000_2

16 bytes: `long double`
Linux: e*n düşük 3 bit adresi* 000_2

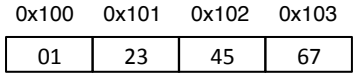
Kayan noktalı sayı (floating point)

Bias = $2^{k-1} - 1$

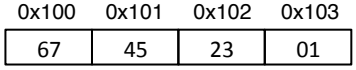
Bayt Sıralama (Byte Ordering)

`0x100` adresinde 4-bayt'lık değişken `0x01234567`

Big Endian
En anlamsız bayt en yüksek adreste



Little Endian
En anlamsız bayt en düşük adreste



Soru 1. (20 puan) *Arabellek taşması (Buffer overflow).*

Buflab ödevinizde `gets` fonksiyonunun açığından yararlanarak çeşitli arabellek taşma saldırıları nasıl gerçekleştirilir onu deneyimlediniz. Ancak pratikte `gcc` derleyicisi, `gets` fonksiyonunun kullanımını tespit ettiğinde programcıyı bu fonksiyonun güvenli olmadığı konusunda uyarmaktadır.

Kamil, kodlama yaparken `gets` gibi güvenli olmayan fonksiyonlardan uzak durduğu sürece arabellek taşma saldırılarından korunduğunu düşünmektedir. Aşağıda Kamil'in yazdığı bir kod verilmiştir.

NOT: Bu kodun 32-bit Little-Endian bir makinede bir uyarı mesajı almadan hatasız derlendiğini varsayınız.

`test.c`

```
#include <stdio.h>

void sifre_gir() {
    char sifre[8];
    scanf("%s",sifre);
}

void hodrimeydan() {
    printf("Acigimi buldunuz!!!\n");
}

int main(void)
{
    printf("Sifre: ");
    sifre_gir();
    return 0;
}
```

Bu soruda Kamil'in bu düşüncesinin yanlış olduğunu `hodrimeydan` fonksiyonuna bir çağrı gerçekleştirerek kanıtlamanız istenmektedir.

Kamil'in `test` programını yarattığı klasörün altında çalıştığınızı ve komut satırında aşağıdaki komut zincirini çalıştırdığınızı düşünün.

```
./hex2raw < exploit | ./test
```

burada `exploit` *saldırıda kullandığınız kodun onaltılı sayı düzenindeki karşılığını içeren bir metin dosyasıdır.*

Bu güvenlik açığını Kamil'e kanıtlamak için `exploit` dizisinin içeriği ne olmalıdır?

NOT: Cevabınızda peşi sıra gelen `n` *rastgele bayt için* `[n]` *gösterimini kullanabilirsiniz.*

`test` programının `objdump` ile elde edilen Assembly çıktısı (sadece ilgili bölümler):

```
08048464 <hodrimeydan>:
 8048464:       55                      push   %ebp
 8048465:       89 e5                   mov    %esp,%ebp
 8048467:       83 ec 18                sub    $0x18,%esp
 804846a:       c7 44 24 04 90 85 04    movl   $0x8048590,0x4(%esp)
 8048471:       08
 8048472:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8048479:       e8 f6 fe ff ff          call   8048374 <__printf_chk@plt>
 804847e:       c9                      leave
 804847f:       c3                      ret

08048480 <sifre_gir>:
 8048480:       55                      push   %ebp
 8048481:       89 e5                   mov    %esp,%ebp
 8048483:       83 ec 28                sub    $0x28,%esp
 8048486:       8d 45 f0                lea    -0x10(%ebp),%eax
 8048489:       89 44 24 04             mov    %eax,0x4(%esp)
 804848d:       c7 04 24 a5 85 04 08    movl   $0x80485a5,(%esp)
 8048494:       e8 fb fe ff ff          call   8048394 <__isoc99_scanf@plt>
 8048499:       c9                      leave
 804849a:       c3                      ret

0804849b <main>:
 804849b:       55                      push   %ebp
 804849c:       89 e5                   mov    %esp,%ebp
 804849e:       83 e4 f0                and    $0xfffffff0,%esp
 80484a1:       83 ec 10                sub    $0x10,%esp
 80484a4:       c7 44 24 04 a8 85 04    movl   $0x80485a8,0x4(%esp)
 80484ab:       08
 80484ac:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 80484b3:       e8 bc fe ff ff          call   8048374 <__printf_chk@plt>
 80484b8:       e8 c3 ff ff ff          call   8048480 <sifre_gir>
 80484bd:       b8 00 00 00 00          mov    $0x0,%eax
 80484c2:       c9                      leave
 80484c3:       c3                      ret
```

Soru 2. (24 puan) *Bellek sıradüzeni (The memory hierarchy).*
Bu soruyu yanıtlarken üzerinde çalıştığınız sistemin 32-bit bir Linux makinesi olduğunu ve 32 baytlık bloklara sahip 4 kümeden oluşan direkt eşlenen (direct-mapped) bir önbellek kullandığını varsayınız.

(a) (12 puan) `u` dizisinin `0x0` adresinden başladığını ve önbelleğin başlangıçta soğuk (boş) olduğunu kabul ederek;

```
int i,j;
float u[8][8],ux;
for (i = 1; i < 7; i++)
    for (j = 1; j < 7; j++)
        ux = 0.5*(u[i][j+1]-u[i][j-1]);
```

programının çalıştırılması sırasında `u` dizisinin her bir elemanına yapılan *ilk erişimin* bir rastlama (hit) (*R*) veya ıskalama (miss) (*I*) ile sonuçlanıp sonuçlanmadığını aşağıdaki tablo üzerinde işaretleyerek gösteriniz. Ayrıca `u` dizisine yapılan tüm erişimleri göz önüne alarak ıslama oranını hesaplayınız.

	j=0	1	2	3	4	5	6	7
i=0								
1								
2								
3								
4								
5								
6								
7								

Iskalama oranı =

(b) (12 puan) u dizisinin yine 0x0 adresinden başladığını ve önbelleğin başlangıçta soğuk olduğunu kabul ederek;

```
int i,j;
float u[8][8],uy;
for (i = 1; i < 7; i++)
    for (j = 1; j < 7; j++)
        uy = 0.5*(u[i+1][j]-u[i-1][j]);
```

programının çalıştırılması sırasında u dizisinin her bir elemanına yapılan *ilk erişimin* bir rastlama (hit) (*R*) veya ıskalama (miss) (*I*) ile sonuçlanıp sonuçlanmadığını aşağıdaki tablo üzerinde işaretleyerek gösteriniz. Ayrıca u dizisine yapılan tüm erişimleri göz önüne alarak ıslama oranını hesaplayınız.

	j=0	1	2	3	4	5	6	7
i=0								
1								
2								
3								
4								
5								
6								
7								

Iskalama oranı =

Soru 3. (25 puan) *Bağlama (Linking).*
Aşağıda `main.c` ve `bubbleSort.c` dosyalarının içerikleri belirtilmiştir:

`main.c`

```
#define SIZE 10

void bubbleSort( float *array, int size, int order );
int main(int argc, char** argv) {
    int i,order,N;
    float a[SIZE];
    sscanf(argv[1], "%d", &order);
    sscanf(argv[2], "%d", &N);

    for(i=0;i<N;i++)
        scanf("%f", &a[i]);
    bubbleSort(a,N,order);

    return 0;
}
```

`bubbleSort.c`

```
extern int formatted;

static void printArray(float* array, int SIZE) {
    int i;
    for (i = 0; i<SIZE; i++)
        if (formatted)
           printf("%.2f ", array[i]);
        else printf("%f ", array[i]);
    printf("\n");
}

void bubbleSort(float *array, int size, int order) {
    int pass,j;

    printArray(array, size);

    for (pass=0; pass<size-1; pass++)
        for (j=0; j<size-1; j++)
            if (order==0) {
               if (array[j]>array[j+1])
                  swap(&array[j], &array[j+1]);
          }
            else {
               if (array[j]<array[j+1])
                  swap(&array[j], &array[j+1]);
            }
    printArray(array, size);
}
```

Yukarıdaki tanımlara ek olarak `helper.c` adlı dosyada `swap` adlı fonksiyonun ve `formatted` adlı değişkeninin tanımlandığını

```
int formatted=1;
void swap( float *element1Ptr, float *element2Ptr);
```

varsayarak aşağıdaki soruları yanıtlayınız.

(a) (14 puan) Aşağıdaki sembol tablolarında verilen her ismin karşısına o sembolün yerel veya global ve kuvvetli veya zayıf olup olmadığını belirtiniz.

Bu özelliklerin verilen bir isim için geçerli olmadığını düşünüyorsanız ilgili kutuya çarpı atabilirsiniz. Örneğin; verilen sembolün sembol tablosunda yer almaması gerektiğini düşünüyorsanız ilgili satırdaki ilk kutuya çarpı atmanız yeterlidir, veya sembolün global olmadığını düşüyorsanız (ve dolayısıyla zayıf veya güçlü olamıyorsa) ilgili satırdaki ikinci kutuya çarpı atabilirsiniz.

NOT: C programlama dilinde harici fonksiyonları beyan etme zorunluluğu yoktur!

`main.c`	yerel / global	zayıf / güçlü
`bubbleSort`		
`main`		
`order`		

`bubbleSort.c`	yerel / global	zayıf / güçlü
`formatted`		
`printArray`		
`bubbleSort`		
`swap`		

(b) (5 puan) `helper.c` dosyasının derlenip `helper.a` adlı statik bir kütüphanede arşivlendiğini varsayınız. Bu durumda altta sıralanan `gcc` komutlarının çıktısı aşağıdaki olasılıklardan hangisi olmaktadır?

(i) Derleme ve bağlama aşamaları düzgün bir şekilde gerçekleşir.
(ii) Tanımsız referanstan (undefined reference) dolayı bağlama aşamasında hata alınır.
(iii) Çoklu tanımdan (multiple definitions) dolayı bağlama aşamasında hata alınır.

Komut	**Sonuç**
`gcc -o sort main.c bubbleSort.c helper.a`	
`gcc -o sort helper.a main.c bubbleSort.c`	
`gcc -o sort helper.a bubbleSort.c main.c`	
`gcc -o sort bubbleSort.c helper.a main.c`	
`gcc -o sort bubbleSort.c main.c helper.a`	

(c) (6 puan) Aşağıda `bubbleSort.o` nesne dosyasının `objdump` ile elde edilen Assembly çıktısının `printArray` ile ilgili bölümü verilmiştir.

Bu kod üzerinde yer değiştirebilen (relocatable) kısımları işaretleyiniz:

```
00000000 <printArray>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 28                sub    $0x28,%esp
   6:   c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
   d:   eb 45                   jmp    54 <printArray+0x54>
   f:   a1 00 00 00 00          mov    0x0,%eax
  14:   85 c0                   test   %eax,%eax
  16:   74 1d                   je     35 <printArray+0x35>
  18:   8b 45 fc                mov    0xfffffffc(%ebp),%eax
  1b:   c1 e0 02                shl    $0x2,%eax
  1e:   03 45 08                add    0x8(%ebp),%eax
  21:   d9 00                   flds   (%eax)
  23:   dd 5c 24 04             fstpl  0x4(%esp)
  27:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  2e:   e8 fc ff ff ff          call   2f <printArray+0x2f>
  33:   eb 1b                   jmp    50 <printArray+0x50>
  35:   8b 45 fc                mov    0xfffffffc(%ebp),%eax
  38:   c1 e0 02                shl    $0x2,%eax
  3b:   03 45 08                add    0x8(%ebp),%eax
  3e:   d9 00                   flds   (%eax)
  40:   dd 5c 24 04             fstpl  0x4(%esp)
  44:   c7 04 24 06 00 00 00    movl   $0x6,(%esp)
  4b:   e8 fc ff ff ff          call   4c <printArray+0x4c>
  50:   83 45 fc 01             addl   $0x1,0xfffffffc(%ebp)
  54:   8b 45 fc                mov    0xfffffffc(%ebp),%eax
  57:   3b 45 0c                cmp    0xc(%ebp),%eax
  5a:   7c b3                   jl     f <printArray+0xf>
  5c:   c7 04 24 0a 00 00 00    movl   $0xa,(%esp)
  63:   e8 fc ff ff ff          call   64 <printArray+0x64>
  68:   c9                      leave
  69:   c3                      ret
```

Soru 4. (30 puan) *Kural dışı durumlarda komut akışı (Exceptional control flow).*

NOT: Bu soruyu yanıtlarken tüm fonksiyonların normal bir şekilde sonlandığını ve `printf` *fonksiyonunun arabellek kullanmadığını (unbuffered olduğunu) varsayınız.*

(a) (15 puan) Aşağıdaki verilen C programının üretebileceği olası çıktıları nelerdir?

```
int main() {
    pid_t pid1, pid2;
    printf("0\n");
    if ((pid1 = fork()) > 0) {
        printf("1\n");
        if ((pid2 = wait(NULL)) > 0) {
            printf("2\n");
        }
    }
    else {
        printf("3\n");
        if ((pid2 = fork()) == 0) {
            printf("4\n");
            exit(0);
        }
        else {
            printf("5\n");
        }
    }
    printf("6\n");
    return 0;
}
```

Aşağıdaki tabloda verilen bir çıktının olası olduğunu düşünüyorsanız ilgili sütunu *Evet*, olmadığını düşünüyorsanız *Hayır* olarak işaretleyiniz.

0 3 4 5 6 1 2 6	0 1 2 6 3 5 6	0 1 3 4 5 6 2 6	0 3 1 4 2 5 6 6	0 3 1 5 6 2 6

(b) (15 puan) Aşağıda verilen C programının üreteceği olası çıktılar nelerdir?

NOT: `kill` *fonksiyonun ilk argümanı 0 ise 2. argümanda belirtilen sinyal o anki işlemin (current process) dahil olduğu grup içindeki bütün diğer işlemlere gönderilir.*

```
void handler(int sig)
{
    printf("%d dede\n", pid);
    wait(NULL);
}

int main() {
    int pid;
    setpgid(0,0);
    signal(SIGCHLD, handler);
    printf("leyla\n");
    if((pid = fork()) > 0) {
        printf("mecnun\n");
    }
    else {
        kill(0, SIGCHLD);
        printf("ismail\n");
        exit(0);
    }
    printf("erdal\n");
    return 0;
}
```

Aşağıdaki tabloda verilen bir çıktının olası olduğunu düşünüyorsanız ilgili sütunu *Evet*, olmadığını düşünüyorsanız *Hayır* olarak işaretleyiniz.

leyla dede ismail dede mecnun erdal	leyla dede dede ismail mecnun erdal	leyla erdal mecnun dede ismail	leyla mecnun erdal dede ismail	leyla mecnun dede erdal dede ismail

Soru 5. (11 puan) *Okuma ödevleri.*

(a) (3 puan) Yer değiştirebilen kod (relocatable code) kavramının faydası, çeşitli altbirimlerden oluşan bir programın belleğin herhangi bir yerine yüklenip buradan çalıştırılabilmesidir. Bu özellik sayesinde bir altbirime ait kodun sadece belli bir adresten başlatılabilme gereksinimi ortadan kalkmaktadır. Bu yer değiştirebilen kod kavramı ne kadar eskiye dayanmaktadır?
NOT: İlgili bilgisayar sisteminin veya bilimadamının adını belirtmeniz yeterlidir.

(b) (8 puan) İlk İnternet solucanı olan Morris solucanı, arabellek taşımından kaynaklı bir güvenlik açığından yararlanarak yüklendiği sistemi kötücül amaçları doğrultusunda kullanmıştır. Bu solucan, aynı zamanda belirli aralıklarla kendisi üzerinde bir `fork` sistem çağrısı gerçekleştirmekte ve bunun ardından üst komut işlemini (parent process) öldürmektedir. Sizce solucanın bu tarz bir yol izlemesinin altında yatan neden(ler) nedir? Kısaca açıklayınız.