

Öğrenci Adı – Soyadı: _____
Öğrenci Numarası: _____

<i>S1</i>	<i>S2</i>	<i>S3</i>	<i>S4</i>	<i>S5</i>	<i>Toplam</i>



Hacettepe Üniversitesi
Bilgisayar Mühendisliği Bölümü

BİL 220
Sistem
Programlamaya Giriş
2. Ara Sınav

Tarih: 3 Mayıs 2012

Süre: 110 dak.

Sınavı 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 <i>Src, Dest</i>	Dest = Dest + Src
subl <i>Src, Dest</i>	Dest = Dest - Src
imull <i>Src, Dest</i>	Dest = Dest * Src
sall <i>Src, Dest</i>	Dest = Dest << Src
sarl <i>Src, Dest</i>	Dest = Dest >> Src
shrl <i>Src, Dest</i>	Dest = Dest >> Src
xorl <i>Src, Dest</i>	Dest = Dest ^ Src
andl <i>Src, Dest</i>	Dest = Dest & Src
orl <i>Src, Dest</i>	Dest = Dest Src
incl <i>Src</i>	Dest = Dest + 1
decl <i>Src</i>	Dest = Dest - 1
negl <i>Src</i>	Dest = - Dest
notl <i>Src</i>	Dest = ~ Dest

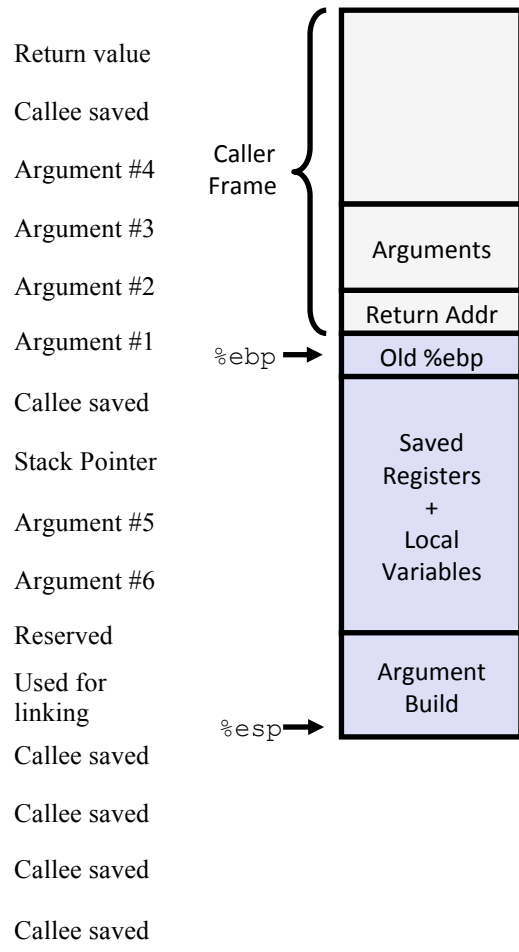
Bellek İşlemleri

Format	İşlem
(Rb, Ri)	Mem[Reg[Rb]+Reg[Ri]]
D(Rb, Ri)	Mem[Reg[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		
%rbx	%ebx %bx	%bh	%bl		
%rcx	%ecx %cx	%ch	%cl		
%rdx	%edx %dx	%dh	%dl		
%rsi	%esi %si		%sil		
%rdi	%edi %di		%dil		
%rbp	%ebp %bp		%bpl		
%rsp	%esp %sp		%spl		
%r8	%r8d %r8w		%r8b		
%r9	%r9d %r9w		%r9b		
%r10	%r10d %r10w		%r10b		
%r11	%r11d %r11w		%r11b		
%r12	%r12d %r12w		%r12b		
%r13	%r13d %r13w		%r13b		
%r14	%r14d %r14w		%r14b		
%r15	%r15d %r15w		%r15b		

Linux Yığıt (Stack) Yapısı



Özel Hizalama Durumları (Intel IA32)

1 byte: char, ...

sınırlandırma yok

2 bytes: short, ...

en düşük bit adresi 0₂

4 bytes: int, float, char *, ...

en düşük 2 bit adresi 00₂

8 bytes: double, ...

Windows: *en düşük 3 bit adresi 000₂*

Linux: *en düşük 2 bit adresi 00₂*

12 bytes: long double

Windows & Linux:

en düşük 2 bit adresi 00₂

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 0₂

4 bytes: int, float, ...

en düşük 2 bit adresi 00₂

8 bytes: double, char *, ...

Windows & Linux:

en düşük 3 bit adresi 000₂

16 bytes: long double

Linux: *en düşük 3 bit adresi 000₂*

Bayt Sıralama (Byte Ordering)

0x100 adresinde 4-bayt'lık değişken

0x01234567

Big Endian

En anlamsız bayt en yüksek adreste

0x100 0x101 0x102 0x103

01	23	45	67
----	----	----	----

Little Endian

En anlamsız bayt en düşük adreste

0x100 0x101 0x102 0x103

67	45	23	01
----	----	----	----

Kayan noktalı sayı (floating point)

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

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

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

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.

[20] 64 84 04 08

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    $0xffffffff0,%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 0×0 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 ıskalama oranını hesaplayınız.

	0	1	2	3	4	5	6	7
0								
1	R	R	I	R	R	R	R	R
2	R	R	I	R	R	R	R	R
3	R	R	I	R	R	R	R	R
4	R	R	I	R	R	R	R	R
5	R	R	I	R	R	R	R	R
6	R	R	I	R	R	R	R	R
7								

(5 puan)

Iskalama oranı = $6 / 72 = 1/12$ (7 puan)

(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 ıskalama oranını hesaplayınız.

	0	1	2	3	4	5	6	7
0		I	R	R	R	R	R	
1		I	R	R	R	R	R	
2		I	R	R	R	R	R	
3		I	R	R	R	R	R	
4		I	R	R	R	R	R	
5		I	R	R	R	R	R	
6		I	R	R	R	R	R	
7		I	R	R	R	R	R	

(5 puan)

İskalama oranı = $8 / 72 = 1 / 9$ (7 puan)

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 düşünüyorsanız (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	global	zayıf
main	global	güçlü
order	X	X

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

- (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?

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

Komut	Sonuç
gcc -o sort main.c bubbleSort.c helper.a	i
gcc -o sort helper.a main.c bubbleSort.c	ii
gcc -o sort helper.a bubbleSort.c main.c	ii
gcc -o sort bubbleSort.c helper.a main.c	i
gcc -o sort bubbleSort.c main.c helper.a	i

(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  $0x0,0xffffffffc(%ebp)
 d: eb 45             jmp   54 <printArray+0x54>
 f: a1 00 00 00 00   mov   0x0,%eax
                                10: R_386_32   formatted
14: 85 c0             test  %eax,%eax
16: 74 1d             je    35 <printArray+0x35>
18: 8b 45 fc         mov   0xffffffffc(%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)
                                2a: R_386_32   .rodata
2e: e8 fc ff ff ff   call  2f <printArray+0x2f>
                                2f: R_386_PC32 printf
33: eb 1b             jmp   50 <printArray+0x50>
35: 8b 45 fc         mov   0xffffffffc(%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)
                                47: R_386_32   .rodata
4b: e8 fc ff ff ff   call  4c <printArray+0x4c>
                                4c: R_386_PC32 printf
50: 83 45 fc 01     addl  $0x1,0xffffffffc(%ebp)
54: 8b 45 fc         mov   0xffffffffc(%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>
                                64: R_386_PC32 putchar
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.

Evet	Hayır	Evet	Hayır	Evet
0	0	0	0	0
3	1	1	3	3
4	2	3	1	1
5	6	4	4	5
6	3	5	2	6
1	5	6	5	2
2	6	2	6	6
6		6	6	

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

NOT: kill fonksiyonunun 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.

Evet	Evet	Hayır	Evet	Hayır
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 deęiřtiren kod (relocatable code) kavramının faydası, çeřitli altbirimlerden oluřan bir programın belleęin herhangi bir yerine yüklenip buradan çalıştırılabilmesidir. Bu özellik sayesinde bir altbirime ait kodun sadece belli bir adresten başlatılabilme gereksinimi ortadan kalkmaktadır. Bu yer deęiřtiren kod kavramı ne kadar eskiye dayanmaktadır?

NOT: İlgili bilgisayar sisteminin veya bilimadamının adını belirtmeniz yeterlidir.

John von Neumann

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

İki sebebi vardır. Öncelikle, solucan fork çağrısı yaparak komut dizisi numarasını sürekli deęiřtirmektedir ki böylece işlemci zamanı tüketen tek bir komut dizisi gözükmemektedir. İkinci olarak, çok uzun zaman çalışan komut dizilerinin öncelikleri çizelgeyici (scheduler) tarafından düşürülmektedir. fork çağrısı yaparak yeni yaratılan komut dizisi her zaman normal çizelge önceliğinde kalmaktadır.