ECE 391 Exam 1, Spring 2010

Feb 24, 2010, 7–9 p.m.

Name	
NetID	

- Be sure that your exam booklet has 14 pages.
- Write your netid at the top of each page.
- This is a closed book exam.
- You are allowed one 8.5×11" sheet of notes.
- Absolutely no interaction between students is allowed.
- Show all of your work.
- The last page has a reference for the synchronization API
- Don't panic, and good luck!

Problem 1	26 points	
Problem 2	8 points	
Problem 3	12 points	
Problem 4	18 points	
Total	64 points	

(You may use the rest of this page as scratch material)

Problem 1 **Short answer questions** (**26** points)

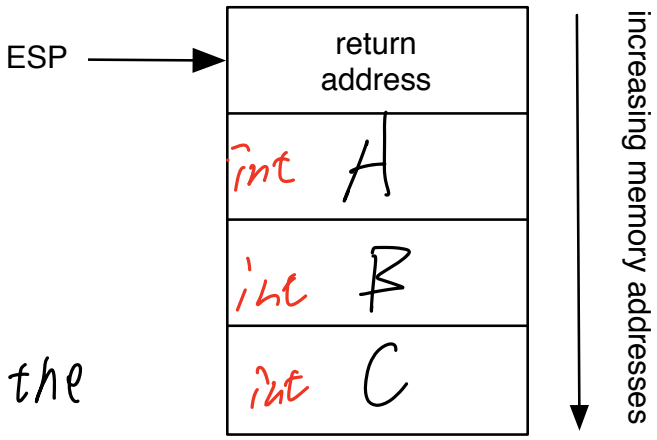
Answer the questions below and justify your answer. You should not need more than one or two sentences for explanations.

a **Calling conventions** (**10** points)

i **Parameter ordering** (**4** points)
Consider the following function prototype:

```
int test(int A, int B, int C);
```

Draw the parameters saved on the stack in the correct order, as used in the C calling convention covered in class. Why is this order used?



Because for C, the parameters of the function are push onto stack from right to left for 8(%ebp) 8(%rbp)

ii **Stack vs. registers** (**4** points)
In assembly linkage, arguments are passed in registers, rather than the stack. Describe one advantage and one disadvantage of this approach, as compared to the C calling convention.

advantage: take less stack space fast R/W

disadvantage: the number of available registers may be not enough to store all the arguments of a function

iii **System calls** (**2** points)
System calls pass arguments in registers. Explain why.

Because system calls in a interrupt should not change the stack as it will cause security leak

different stack for user kernel

b **Version control** (**4** points)

In this class, you are using the subversion version control system. Please explain two (2) benefits of using it.

easier to fix the bug one by one revert code fix

~~easier to test each function's functionality~~

save code safely

c **Concurrency problems** (**2** points)

Explain the difference between deadlock and livelock.

Deadlock will cause relevant threads blocked

Livelock will cause relevant threads do infinte meaningless loop

d **Volatile** (**4** points) What does the `volatile` keyword in C mean? And why is it not used by default on all variables?

means the compiler should assume the variable's value could be influenced by interruption or other processor. ✓

Because it will reduce performance and the compiler can optimize the code if it is not volatile reload each time

e **Data sizes** (**6** points)

An image editing program allows each pixel to have 5-bit red, green, and blue values, as well as an 17-bit *alpha* (transparency) value for layered images. A programmer uses the following declaration:

```
struct pixel {
    unsigned char red;
    unsigned char green;
    unsigned int alpha;
    unsigned char blue;
};
```

i (**2** points) The programmer computes the size of the data structure using `sizeof(struct pixel)`. To his surprise, the answer is 12 bytes. Can you explain to him why this is? (Assume you are working on a 32-bit architecture.)

low 2 byte of red and green takes 4 bytes
alpha takes 4 bytes
blue takes low 1 byte of 4 bytes
alignment
total takes 12 byte

Suggest some changes that will let the programmer get this size down to:

ii **8 bytes** (**2** points)

change the order to { red, green, blue, alpha }

iii **7 bytes (Bonus)** (**2** points)

~~change the order to { alpha, red, green, blue }~~
change alpha into 4 chars

iv **4 bytes** (**2** points)

combine all of them into one int
where [1b:0] stores alpha
[21:17] stores red
[26:22] stores blue
[31:27] stores green

Problem 2 **Synchronization** (**8** points)

The driver for your USB rocket launcher[1] has been malfunctioning. You trace the problem to the implementation of an ioctl call, listed below:

```
 1 static int *buf1, *buf2;
 2 static spinlock_t buf1_lock = SPIN_LOCK_UNLOCKED;
 3 static DECLARE_MUTEX(buf2_mutex);
 4
 5 int usb_rocket_launcher_telemetry_ioctl() {
 6     int i;
 7     spin_lock(&buf1_lock);   /* must check for valid pointer */
 8     if(buf1 == NULL)         /* after obtaining lock */
 9         return -1;
10     down(&buf2_mutex);       /* must check for valid pointer */
11     if(buf2 == NULL)         /* after obtaining mutex */
12         return -1;
13     for(i=0; i<10; i++)
14         buf2[9-i]=buf1[i];
15     up(&buf2_mutex);
16     spin_unlock(&buf1_lock);
17     return 0;
18 }
```

You know from your experience that the `buf1` variable is used by both system calls (as above) and interrupts, and therefore is protected by a spinlock, whereas `buf2` is only used in system calls, and thus is protected by a mutex. Please explain *all* the errors in this code and how you would fix them. *(Note: you should look for errors related to synchronization only, and not, e.g., syntax errors.)*

when interrupt happens at line 8 and try to access buf1, will cause deadlock

Use spin_lock_irqsave and spin_lock_irqrestore

Should release the lock before return

at release lock before return

[1] Available from ThinkGeek

Problem 3 **Where's Waldo** (**12** points)

Where's Waldo (known outside of North America as *Where's Wally?*) is a series of children's books that consist of full-page illustrations of hundreds of people in a frenzy of activity. The intent is for the reader to find a character named Waldo who is hidden in the group. Waldo is always dressed in a red/white horizontally striped shirt, a bobble red hat, and wears glasses.

Fed up with never finding Waldo as a child, your TA Chris has demanded that you write program that finds Waldo for him. To help you along, Chris has written code that will take a *Where's Waldo* image and create a singly-linked linked list where each node holds information about each person. The structure of a linked list node is:

```
typedef struct {
   uint8_t r,g,b;   /* RGB components of the hat color */
   uint8_t glasses; /* 1 if glasses are present, 0 otherwise */
   int32_t position;/* Position of the person relative to the image */
   person_t* next;  /* Pointer to the next person in the linked list */
} person_t;
```

The head of this linked list is found in the global variable `people_list_head`.

Assuming no compiler padding, write a function in x86 assembly to traverse the linked list of people, determine if a "Waldo" is found, and return the position number. If a "Waldo" is not found, return -1. To account for fashions of the future, your function should be general and take as an argument a pointer to a `person_t` structure that describes Waldo's attributes. The C function prototype is:

```
/* whereiswaldo()
 *   Description: Searches through the linked list of people to find if the
 *       waldo element is present in the list.
 *   Input: waldo - person_t struct that has the attributes of the person
 *          we want to find in the linked list.
 *   Returns: If the waldo element is found, return its position
 *             information relative to the image.  If not found, return -1.
 */
int32_t whereiswaldo(person_t* waldo);
```

You can assume that the arguments passed in are valid types (No `NULL` checking or type checking required). You may wish to write the function in C first, using the space provided. Your C code **will not** be graded, it is for your convenience only.

```
#typedef struct {
#   uint8_t r,g,b;   /* RGB components of the hat color */
#   uint8_t glasses; /* 1 if glasses are present, 0 otherwise */
#   int32_t position;/* Position of the person relative to the image */
#   person_t* next;  /* Pointer to the next person in the linked list */
#} person_t;
```

%eax

(%eax) r

1(%eax) g

2(%eax) b

3(%eax) glasses

```
.global people_list_head

whereiswaldo:
        push %ebp
        movl %esp, %ebp
        push %ebx

        movl 8(%ebp), %ebx
        movl people_list_head, %ecx
for_each_person:
        cmpl $0, %ecx
        je   NOT_FOUND
        movl (%ebx), %ax
        movl (%ecx), %dx
        cmp  %ax, %dx
        je   FOUND
        movl 8(%ecx), %ecx
        jmp  for_each_person
NOT_FOUND:
        movl $-1, %eax
        pop %ebx
        LEAVE
        RET

FOUND:
        movl 4(%ecx), %eax
        pop %ebx
        LEAVE
        RET
```

(you can write your C code here)

Problem 4 **Boy/Girl Locks** (**18** points)

You must write a lock that guards access to a shared bathroom. The bathroom may be used by several people at the same time, as long as they are of the same gender; however, mixing genders is never allowed. The data structure representing the lock has been filled in for you, please complete the code. Note: your code **must** allow for multiple people to use the bathroom, as long as they are of the same gender.

a **Implementation** (**15** points)

```
#define BOY         0
#define GIRL        1

typedef struct {
    spinlock_t lock;
    int count[2];
} boygirl_lock_t;

void boygirl_lock_init(boygirl_lock_t *lock) {
    spin_lock_init(&(lock->lock));
    lock->count[BOY] = 0;
    lock->count[GIRL]=0;
}

void boygirl_lock(boygirl_lock_t *lock, int gender) {
    if (gender != BOY && gender != GIRL) return;
    while(1) {
        if(gender == BOY){ spin_lock(&lock->lock);
            if (lock->count[GIRL] != 0) continue;
            lock->count[BOY] += 1;
            return;
        }else {
            if( lock->count[BOY] !=0) continue;
            lock->count[GIRL] +=1
            return;
        }
    }
}
```

```
void boygirl_unlock(boygirl_lock_t *lock, int gender) {
    if(gender == BOY){
        lock->count[BOY] -= 1
        spin_unlock(&(lock->lock));
        return;
    } else {
        lock->count[GIRL] -= 1
        spin_unlock(&(lock->lock));
        return;
    }
}
```

b **Starvation** (**3** points)

Does your implementation allow starvation to occur? Explain.

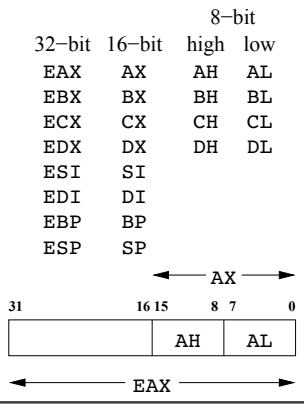
yes. As the priority of boy and girl are equal. If boys are already inside bathroom, and girls are waiting. Then new_boys will still join in instead of waiting after girls.

You may tear off this page to use as a reference

Synchronization API reference

`spinlock_t lock;`	Declare an uninitialized spinlock
`spinlock_t lock1 = SPIN_LOCK_UNLOCKED;` `spinlock_t lock2 = SPIN_LOCK_LOCKED;`	Declare a spinlock and initialize it
`void spin_lock_init(spinlock_t* lock);`	Initialize a dynamically-allocated spin lock (set to unlocked)
`void spin_lock(spinlock_t *lock);`	Obtain a spin lock; waits until available
`void spin_unlock(spinlock_t *lock);`	Release a spin lock
`void spin_lock_irqsave(spinlock_t *lock,` `unsigned long& flags);`	Save processor status in `flags`, mask interrupts and obtain spin lock (note: flags passed by name (macro))
`void spin_lock_irqrestore(spinlock_t *lock,` `unsigned long flags);`	Release a spin lock, then set processar status to `flags`
`struct semaphore sem;`	Declare an uninitialized semaphore
`static DECLARE_SEMAPHORE_GENERIC (sem, val);`	Allocate statically and initialize to `val`
`DECLARE_MUTEX (mutex);`	Allocate on stack and initialize to one
`DECLARE_MUTEX_LOCKED (mutex);`	Allocate on stack and initialize to zero
`void sema_init(struct semaphore *sem, int val);`	Initialize a dynamically allocated semaphore to `val`
`void init_MUTEX(struct semaphore *sem);`	Initialize a dynamically allocated semaphore to one.
`void init_MUTEX_LOCKED(struct semaphore *sem);`	Initialize a dynamically allocated semaphore to zero.
`void down(struct semaphore *sem);`	Wait until semaphore is available and decrement (P)
`vod up(struct semaphore *sem);`	Increment the semaphore

x86 reference



`jb`	below	CF is set
`jbe`	below or equal	CF or ZF is set
`je`	equal	ZF is set
`jl`	less	SF ≠ OF
`jle`	less or equal	(SF ≠ OF) or ZF is set
`jo`	overflow	OF is set
`jp`	parity	PF is set (even parity)
`js`	sign	SF is set (negative)

```
movb  (%ebp),%al            # AL ← M[EBP]
movb  -4(%esp),%al          # AL ← M[ESP - 4]
movb  (%ebx,%edx),%al       # AL ← M[EBX + EDX]
movb  13(%ecx,%ebp),%al     # AL ← M[ECX + EBP + 13]
movb  (,%ecx,4),%al         # AL ← M[ECX * 4]
movb  -6(,%edx,2),%al       # AL ← M[EDX * 2 - 6]
movb  (%esi,%eax,2),%al     # AL ← M[ESI + EAX * 2]
movb  24(%eax,%esi,8),%al   # AL ← M[EAX + ESI * 8 + 24]
movb  100,%al               # AL ← M[100]
movb  label,%al             # AL ← M[label]
movb  label+10,%al          # AL ← M[label+10]
movb  10(label),%al         # NOT LEGAL!

movb  label(%eax),%al       # AL ← M[EAX + label]
movb  7*6+label(%edx),%al   # AL ← M[EDX + label + 42]

movw  $label,%eax           # EAX ← label
movw  $label+10,%eax        # EAX ← label+10
movw  $label(%eax),%eax     # NOT LEGAL!

call  printf                # (push EIP), EIP ← printf
call  *%eax                 # (push EIP), EIP ← EAX
call  *(%eax)               # (push EIP), EIP ← M[EAX]
call  *fptr                 # (push EIP), EIP ← M[fptr]
call  *10(%eax,%edx,2)      # (push EIP), EIP ←
                            #          M[EAX + EDX*2 + 10]
```

Conditional branch sense is inverted by inserting an "N" after initial "J," e.g., JNB. Preferred forms in table below are those used by debugger in disassembly. Table use: after a comparison such as

```
cmp %ebx,%esi   # set flags based on (ESI - EBX)
```

choose the operator to place between ESI and EBX, based on the data type. For example, if ESI and EBX hold unsigned values, and the branch should be taken if ESI ≤ EBX, use either JBE or JNA. For branches other than JE/JNE based on instructions other than CMP, check the branch conditions above instead.

	`jnz`	`jnae`	`jna`	`jz`	`jnb`	`jnbe`	unsigned comparisons
preferred form	`jne`	`jb`	`jbe`	`je`	`jae`	`ja`	
	≠	<	≤	=	≥	>	
preferred form	`jne`	`jl`	`jle`	`je`	`jge`	`jg`	signed comparisons
	`jnz`	`jnge`	`jng`	`jz`	`jnl`	`jnle`	