

# 《计算机系统基础II》期末测试

姓名：

学号：

考试要求：

- 开卷考试，开卷范围限定文字材料、受监控的考试电脑上存储的本地电子版资料（包括电子书、ppt、笔记、作业等），同时可以在受监控电脑上使用计算器，或python程序编写的简单计算程序；
- 不允许使用手机、Pad，不允许搜索题目相关信息，不允许使用手机、计算机软件等进行相互沟通、询问、核对答案等作弊行为（受监控电脑上在考试期间严禁开启QQ、微信、钉钉等通信工具）；
- 保持后方第二机位的手机在考试期间全程正常开启，摄像头、音频正常工作和接入网络，监考会一直查看考生行为、屏幕和语音；如果考试中出现黑屏或掉线，监考会通过语音进行沟通，联系不上会拨打电话；如果始终无法联系上，则至少会扣分，甚至严重情况会取消考试成绩。注意第二机位（一般为手机）的声音打开，监考老师可能会做一些通知。
- 请在电子版答题纸上填写回答，保存为pdf，文件名为学号+姓名，在obe上按时提交。

我知晓并遵守以上考试规则。个人签名：

大题	一	二	三	四	五	六	七	总分
满分	6	12	13	14	20	25	10	<u>100</u>
得分								

## 1. IPv4地址翻译（6分）

请完成下列IPv4地址形式的翻译：

Hex Address	Dotted-decimal Address
【1】	202.112.113.64
【2】	10.77.50.39
0xC0A80A77	【3】

## 2. 磁盘调度算法（12分）

假设一个磁盘有100个柱面，编号为0~99，在刚刚完成了磁道25处的请求后，磁头当前正在磁道43处进行服务。然后，用户请求的磁盘磁道按35、6、41、2、80、20、22、10的次序很快到达磁盘驱动器，寻道时每移动一个磁道需要0.1ms。请写出以下每种算法的访问请求服务顺序（标记出磁道号），并计算以下算法的总寻道时间（从当前时刻开始计算）：

- (1) 先来先服务算法；(4')
- (2) 最短寻道时间优先算法；(4')
- (3) 电梯调度算法（F-SCAN，磁头向任何一个方向移动的过程中，都可以对请求进行服务）。(4')

### 3. I/O Redirection（13分）

下面的代码中访问的文件text.txt的内容是ASCII码abcdefghijk。你可以假设read()函数是原子性的。

dup函数原型为：int dup(int fd);

dup函数的功能和dup2非常类似，返回值是当前可用的文件描述符中最小的一个，返回值和参数fd都指向同一个打开文件的结构体。

read\_and\_print\_one函数的定义为：

```
static inline void read_and_print_one(int fd) {  
  
    char c;  
  
    read(fd, &c, 1);  
  
    printf("%c", c);  
  
    fflush(stdout);  
  
}
```

- 1) 请写出下列代码的屏幕输出，建议写一些中间过程解释为什么。(6')

```
int main() {  
    int file1 = open("text.txt", O_RDONLY);  
    int file2;  
    int file3 = open("text.txt", O_RDONLY);  
  
    file2 = dup(file3);  
    printf("%d", file1);  
    printf("%d", file2);  
    printf("%d", file3)  
  
    read_and_print_one(file1);  
    read_and_print_one(file2);  
    read_and_print_one(file3);  
}
```

```

read_and_print_one(file1);
read_and_print_one(file2);
read_and_print_one(file3);

return 0;
}

```

2) 请写出下列代码的屏幕输出，建议写一些中间过程解释为什么。(7')

```

int main() {
    int file1, file2, file3, pid;
    file1 = open("text.txt", O_RDONLY);
    file3 = open("text.txt", O_RDONLY);

    file2 = dup(file3);
    read_and_print_one(file1);
    read_and_print_one(file2);

    pid = fork();
    if(!pid) {
        printf("%d", file2);
        read_and_print_one(file3);
        close(file3);

        file3 = open("text.txt", O_RDONLY);
        read_and_print_one(file3);
        read_and_print_one(file2);
    }else {
        wait(NULL);
        read_and_print_one(file3);
        read_and_print_one(file2);
        read_and_print_one(file1);
    }

    return 0;
}

```

## 4. 并行代码 (14分)

1) 假设N=3，请填写代码中的9个空，使程序总是能够打印正确的等式（例如\n7=+2+3+2, 或\n7=+3+2+2或\n7=+2+2+3）。注意[1]-[6]空中只能填写信号量的P和V操作。(9')

2) 如果N=6, 9, 12等3的倍数，这个代码具有什么bug? 在什么情况下会出什么错误? (5')

```

#define N 3

```

```

#define M 100    // M >> N

sem_t a, b, c;

int main(void) {
    void *(*threads[3])(void *) = {doA, doB, doC};
    pthread_t tid[N];
    int i;

    // sem_init第三个参数是信号量的初始值，即最多允许几个线程同时进入临界区
    sem_init(&a, 0, ____[7]__);
    sem_init(&b, 0, ____[8]__);
    sem_init(&c, 0, ____[9]__);

    for(i=0;i<N;i++) {
        pthread_join(tid[i], NULL);
    }

    // destroy sem_t
    ...

    return 0;
}

// 至少使用信号量a
void *doA(void *arg) {
    int i;
    for(i=0;i<M;i++) {
        ____[1]____
        printf("+3");
        ____[2]____
    }
}

// 至少使用信号量b
void *doB(void *arg) {
    int i;
    for(i=0;i<2*M;i++) {
        ____[3]____
        printf("+2");
        ____[4]____
    }
}

// 至少使用信号量c
void *doC(void *arg) {
    int i;
    for(i=0;i<M;i++) {
        ____[5]____
    }
}

```

```

printf("\n7=");
_____ [6] _____
}
}

```

## 5. Flash hybrid mapping (20分)

SSD内部FTL算法采用hybrid mapping的方式进行虚拟地址到物理地址的映射，block table中记录逻辑**block id**与物理地址的映射关系，log table记录逻辑**page id**与物理地址的映射关系（page id = 4\*block id）。要求同时最多只能有3个log block存在，每个原始block对应一个log block（即一个log block中的数据不能来自于不同的原始block），选择log block进行合并操作时优先选择代价小的进行，选择空白block写入时采用first-fit原则。

1) 假设系统的初始状态如下图所示，到达的请求序列为a, b, c, d, e, e, h, m, n, time1, f, r, s, t, time2, i, l, tim3，均为写请求。请画出time1 ~ time3这三个时间点上系统的状态，包括log table, block table, 每个block中的数据块标号（对数据块的修改加'或"表示，例如a', a"等）。（12'）

每个时间点的状态信息写法示例：（其中下划线表示空白，修改后加'（如a'），再次修改加"（如a"），以此类推）

Log Table: 1000->0,

Block Table: 250->0,500->4,750->20,1000->24,1250->28

Block 0: a, b, c, d

Block 1: e, f, g, h

Block 2: a', a", b, \_

....

Log Table:

Block Table: 250->0, 500->4, 750->20, 1000->24, 1250->28

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	d	e	f	g	h								
Block 0				Block 1				Block 2				Block 3			
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
				i	j	k	l	m	n	o	p	q	r	s	t
Block 4				Block 5				Block 6				Block 7			

2) 计算截止到time3时, 发生了几次擦除操作? 写放大多少个page? 做了几次合并, 合并的类型是什么? (switch merge, partial merge还是full merge) (8')

## 6. 文件系统 (25分)

假设文件系统采用类似Very Simple File System的方式进行数据分布, 包括superblock, inode bitmap, data bitmap, inodes (每个inode包含12个direct pointers, 2个indirect point, 1个double indirect pointer, 所有指针均为4B) 和data block, 每个文件或目录每次至少分配一个block。Superblock, inode bitmap和data bitmap各占一个block, 每个block的大小为4KB。该文件系统采用metadata log的方式。

假设以下两问中的操作是独立进行的, 初试状态缓存中没有任何数据, 访问一次后会一直缓存。假设对文件的访问或修改需要触发该文件的访问/修改时间等; 创建/删除文件还会引起该文件所在文件夹记录访问/修改时间的变化(再上层目录不受影响)。所有修改会立刻落盘(对一个数据块的多次更新算消耗一次I/O)。请计算下列操作需要进行多少次I/O(包括读、写)? 建议写出具体过程, 每个部分的I/O次数, 以便出错时能够得一部分分数。

- 1) 假设文件系统中已有/home/ics2/目录, 在其中创建一个goodluck文件, 并写入大小为1MB的数据。(10')
- 2) 假设文件系统中已有/home/ics2/goodluck文件, 大小为1MB, 向其中追加写入100MB数据。(10')
- 3) 假设文件系统中已有/home/ics2/test文件, 执行: rm /home/ics2/test。(5')

## 7. 无锁化 (10分)

下面是有锁的多线程counter的代码, 假设下列CAS函数可用, 且提供原子性的保障。请将多线程counter的代码修改为无锁的版本(给出完整代码)。

```
/* return 1 when swapping happens (*dst = newVal); return 0 when not */
```

```
int CAS(unsigned long *dst, unsigned long oldVal, unsigned long newVal);
```

```
typedef struct __conter_t {
    int value;
    pthread_mutex_t lock;
} counter_t

void init(counter *c) {
    c->value=0;
    pthread_mutex_init(&c->lock, NULL);
}

void increment(counter_t *c) {
    Pthread_mutex_lock(&c->lock);
    c->value++;
}
```

```
    Pthread_mutex_unlock(&c->lock);  
}  
  
void decrement(counter_t *c) {  
    Pthread_mutex_lock(&c->lock);  
    c->value--;  
    Pthread_mutex_unlock(&c->lock);  
}  
  
int get(counter_t *c) {  
    Pthread_mutex_lock(&c->lock);  
    rc = c->value;  
    Pthread_mutex_unlock(&c->lock);  
    return rc;  
}
```