Introduction to Computer Systems
2016 Spring Middle Examination

Name_____ Student No._____ Score_____

Problem 1: (17 points)

1.

2.

3.

Problem 2: (14 points)

1.

2.

3.

4.

Problem 3: (16 points)

1.

2.

3.

Problem 4: (20 points)

1 **[1]** **[2]** **[3]**

2

[1]	**[2]**	**[3]**
[4]	**[5]**	**[6]**
[7]	**[8]**	**[9]**
[10]	**[11]**	**[12]**
[13]	**[14]**	**[15]**

Problem 5: (13 points)

1

2

3

4

Problem 6: (20 points)

1

2

Problem 1: Process (17 points)

```
sigjmp_buf buf;

int counter = 0;

void handler_alrm(int sig){
}

void handler_usr1(int sig) {
  counter +=1;
  siglongjmp(buf,1);
  counter +=2;
}
```

```
int main(void) {
  signal(SIGUSR1, handler_usr1);
  signal(SIGALRM, handler_alrm);
  if (!sigsetjmp(buf,1)) {
    sleep(10);
    printf("A\n");
    counter +=3;
  } else {
    printf("B\n");
  }
  exit(0);
}
```

Assume:

The **pid** of above program is 22033

Signals are received after the **sigsetjmp** function

`printf()` function will not be interrupted by signals.

1. Suppose another program sends **ONE** signal **SIGALRM** to the running process 22033. Please write down the output on screen (2') and the value of "**`counter`**" before exiting from main function. (2')
2. Suppose another program sends **ONE** signal **SIGUSR1** to the running process 22033. Please write down all possible output on screen (3') and the values of "**`counter`**" before exiting from main function. (3')
3. Suppose another program sends **TWO** signals **SIGUSR1 sequentially** to the running process 22033. Please write down all possible outputs on screen (5'). You also need explain your results (2')

Problem 2: IO (14 points)

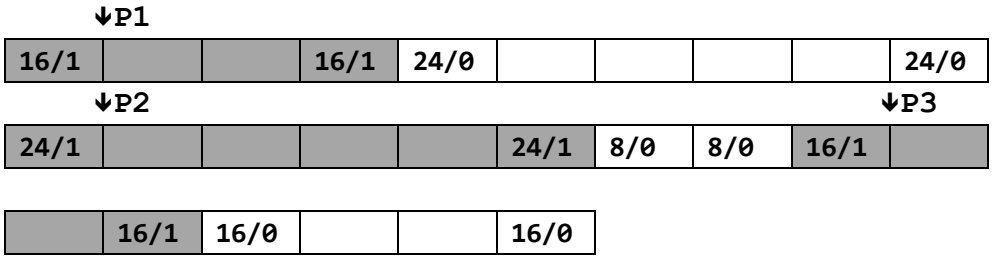
```
1.  pid_t pid;
2.  int main(void) {
3.      int fd1, fd2;
4.      char c;
5.      fd1 = open("ics.txt", O_RDWR);
6.
7.      if ((pid = fork()) == 0) {
8.          read(fd1, &c, 1);
9.          printf("Child fd1 %c\n", c);
10.         fd2 = open("ics.txt", O_RDWR);
11.         // dup2(fd1,fd2);
12.         read(fd2, &c, 1);
13.         // write(fd2, 'H', 1);
14.         printf("Child fd2 %c\n", c);
15.     } else {
16.         waitpid(-1,NULL,0);
17.         read(fd1, &c, 1);
18.         printf("Parent fd1 %c\n", c);
19.         fd2 = open("ics.txt", O_RDWR);
20.         read(fd2, &c, 1);
21.         printf("Parent fd2 %c\n", c);
22.     }
23.     close(fd1);
24.     close(fd2);
25.     return 0;
26. }
```

Note: suppose the initial content of **ics.txt** is "**SJTU120**", and the following problems are **independent**.

1. Please write down the output of the **child** process on the screen when the program runs normally. (2')
2. Please write down the output of the **child** process on the screen when the line 11 "**`// dup2(fd1,fd2);`**" is uncommented. (2') Please also draw the kernel data structures of open files for the **child** process before and after the execution of instructions in line 11 (like the Figure 10.11~10.13). You should give the "**refcnt**" on each open file table. (4')
3. Please write down the output of the **parent** process on the screen when the line 11 ""**`// dup2(fd1,fd2);`**" is uncommented. (2')
4. Please write down the output of the **parent** process on the screen when the line 13 "**`// write(fd2, 'H', 1);`**" is uncommented. (2') Please also write down the content of ics.txt after the program execution? (2')

Problem 3: Memory allocation (16 points)

Now we organize the heap as a sequence of **contiguous** allocated and free blocks, as shown below. **Allocated** blocks are shaded, and **free** blocks are blank (each block represents 1 word = 4 bytes). **Headers** and **footers** are labeled with the number of bytes and allocated bit. The allocator maintains **double-word** alignment. You are given the execution sequence of memory allocation operations (**`malloc()`** or **`free()`**) from 1 to 6.




```
1. P4 = malloc(8)
2. free(P1)
3. P5 = malloc(6)
4. free(P2)
5. P6 = malloc(3)
6. P7 = malloc(2)
```

Please answer the questions below. Assume that **immediate coalescing** strategy and **splitting free blocks** are employed. (*NOTE*: **DON'T** need consider P1, P2 and P3 when calculating **internal** fragments)

1. Assume **best-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed (3'). Please also calculate the total bytes of the **internal** fragments (2').
2. Assume **first-fit** algorithm is used to find free blocks. Please draw the **final** status of memory and mark with block size in headers and footers after the operation sequence is executed (3'). Please also calculate the total bytes of the **internal** fragments (2').
3. Suppose that we use the entire free block instead of splitting when doing allocation and coalesce immediately after freezing. The total size of heap we can use is shown in the picture. Please identify that whether the operation sequence will succeed or not for both **first-fit** and **best-fit** respectively. If yes, please give the total bytes of the **internal** fragments. If no, please point out **which step** causes the failure (out of memory). (6')

Problem 4: Cache (20 points)

Consider a processor with 16-bit physical address. It has a **2-way** set-associative data cache. There are **4 sets** in the cache. The cache line size is **4 bytes**.

1. How would a **16-bit** physical memory address be split into tag/set-index /offset fields in this processor? (3')

   tag **_[1]_** bits, set-index **_[2]_** bits, and offset **_[3]_** bits.

2. The following table shows the content of the data cache at time T. **`ByteX`** is the byte value stored at offset **`X`**.

Set	Tag	Valid	Byte0	Byte1	Byte2	Byte3	Tag	Valid	Byte0	Byte1	Byte2	Byte3
0	0xffc	1	0x12	0x44	0x66	0x55	0x298	1	0x89	0x24	0x34	0x74
1	0xffc	1	0xca	0x98	0x24	0xff	0x923	1	0x19	0x91	0x06	0x21
2	--	0	--	--	--	--	--	0	--	--	--	--
3	0x435	1	0xde	0xad	0xbe	0xef	0xffc	1	0xc0	0xc0	0xc0	0xc0

1) What is the capacity of this cache? (2')

2) Assume the cache line replacement policy is **LRU**. A short program will read memory in the following sequences starting from time T. Each access will read one byte. Please fill the following blanks. If the value cannot be determined with given information, fill the blank with '--'. (15')

Order	Address	Set	Hit/Miss	Byte Returned
1	0xffc3	0	Hit	0x55
2	0xffc4	[1]	[2]	[3]
3	0xffc7	[4]	[5]	[6]
4	0x9230	[7]	[8]	[9]
5	0x435a	[10]	[11]	[12]
6	0x2982	[13]	[14]	[15]

Problem 5: Locality (13 points)

Bob tests the following programs against a **4-way** set associative data cache with **4 sets** and **16 bytes** cache line size. The cache line replacement policy is **LRU**. The size of **`int`** value is **4 bytes**. NOTE that **`i`**, **`j`** and **`sum`** are stored in **registers**. The cache is **empty** before each execution. Each function will be separately executed once. Please only consider **data cache** access.

1. Bob first tries to iterate over the columns of matrix in the inner loop.

```
int a[4][16]; int b[4][16];
int matrix_sum1(void) {
   int i, j, sum = 0;
   for (i = 0; i < 4; i++)
      for (j = 0; j < 16; j++)
         sum += a[i][j] + b[i][j];
   return sum;
}
```

   Assume the address of **a[0][0]** is **0x100**. The address of **b[0][0]** is **0x0**. What is the cache miss rate of executing **matrix_sum1()**? (3')

2. Bob then tries to iterate over the rows in the inner loop.

```
int a[4][16]; int b[4][16];
int matrix_sum2(void) {
   int i, j, sum = 0;
   for (i = 0; i < 16; i++)
      for (j = 0; j < 4; j++)
         sum += a[j][i] + b[j][i]; // beware of order
   return sum;
}
```

   Assume the address of **a[0][0]** is **0x100**. The address of **b[0][0]** is **0x0**. What is the cache miss rate of executing **matrix_sum2()**? (3')

3. Bob finally comes up with a slightly different implementation. Beware of the declaration of **int p[4]** in the first line.

```
int a[4][16]; int p[4]; int b[4][16];
int matrix_sum3(void) {
   int i, j, sum = 0;
   for (i = 0; i < 16; i++)
      for (j = 0; j < 4; j++)
         sum += a[j][i] + b[j][i]; // beware of order
   return sum;
}
```

   Assume the address of **a[0][0]** is **0x110**. The address of **p[0]** is **0x100**. The address of **b[0][0]** is **0x0**. What is the cache miss rate of executing **matrix_sum3()**? (3')

4. Bob executes **matrix_sum1()** on another three different data caches: 1) **C1** is an 8-way set associative cache with 4 sets and 16 bytes cache line size; 2) **C2** is a 4-way set associative cache with 8 sets and 16 bytes cache line size; 3) **C3** is a 4-way set associative cache with 4 sets and 32 bytes cache line size. Which cache(s) would reduce cache miss rate? (2') Please also explain the reason. (2')

Problem 6: Optimization (20 points)

Suppose we have the following codes that run with little efficiency

```
typedef struct {
  int argb[4];  /* Alpha Red Green Blue */
} pixel_t;

typedef struct {
  int len;
  pixel_t *data;
} pixels_t;

static pixels_t pixels;

int get_length(pixels_t *p) { return p->len; }

pixel_t *get_pixel_at(int index) {
  return pixels.data + index;
}

void dosth_and_find(pixels_t *p, pixel_t *minAlpha) {
  int i, j;
  minAlpha = get_pixel_at(0);
  for (i = 1; i < 4; i++) {
    for (j = 0; j < get_length(p); j++) {
      get_pixel_at(j)->argb[i] = get_pixel_at(j)->argb[i]*2%255;
      if (get_pixel_at(j)->argb[0] < minAlpha->argb[0])
        minAlpha = get_pixel_at(j);
    }
  }
}
```

Suppose **`pixels`** has been initialized and **`len`** is greater than 0. Function **`dosth_and_find`** will do some computation on pixels and find the pixel with **least Alpha**.

1. Please refine the function **`dosth_and_find`** with **machine-independent** optimization techniques. (You should add comments to explain your changes and use at least 4 techniques. The outer loop should not be modified) (12’)
2. Please further optimize the function to get a **better locality** and **combine loops** (8’)