Namn:____

Person-nummer:______________________________

Systems programming and Operating systems, 2007

Tentamen 2007-12-18

Instructions:

- Make sure that your exam is not missing any sheets, then write your name and person-nummer on the front. If you need extra pages be sure to write on those too.
- Write your answers in the space provided below the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 60 points.
- The problems are of varying difficulty. The point value of each problem is indicated. Pile up the easy points quickly and then come back to the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. Good luck!
- Remember that there is a SURVEY on the homepage for the course. See “Senaste nytt”.

Problem 1. (10 points):

Consider the C program below. (For space reasons, we are not checking error return codes, so assume that all functions return normally.)

```
main() {

  if (fork() == 0) {
    if (fork() == 0) {
      printf("3");
    }
    else {
      printf("4");
    }
  }
  else {
     if (fork() == 0) {
      printf("1");
      exit(0);
    }
    if ((wait(NULL)) > 0) {
      printf("2");
    }
  }

  printf("0");

  return 0;
}
```

Out of the 6 outputs listed below, circle only the valid outputs of this program. Assume that all processes run to normal completion.

A. 1234000 B. 1020340 C. 1203040

D. 4030201 E. 4321000 F. 3204010

Problem 2. (16 points):

This problem tests your understanding of exceptional control flow in C programs. Assume we are running code on a Unix machine. The following problems all concern the value of the variable `counter`.

Part I (6 points)

```
int counter = 0;

int  main()
{
    int i;

    for (i = 0; i < 3; i ++){
        counter ++;
        fork();
        printf("counter = %d\n", counter);
    }

    printf("counter = %d\n", counter);
    return 0;
}
```

A. How many times would the value of `counter` be printed: ________________

B. What is the value of `counter` printed in the first line? ________________

C. What is the value of `counter` printed in the last line? ________________

Part II (6 points)

```
pid_t pid;
int counter = 0;

void handler1(int sig)
{
    counter ++;
    printf("counter = %d\n", counter);
    fflush(stdout);   /* Flushes the printed string to stdout */
    kill(pid, SIGUSR1);
    exit(0);
}

void handler2(int sig)
{
    counter += 3;
    printf("counter = %d\n", counter);
    exit(0);
}

main() {
    pid_t p;
    pid = getpid();
    signal(SIGUSR1, handler1);
    if ((p = fork()) > 0) {
        signal(SIGUSR1, handler2);
        kill(p, SIGUSR1);
        while(1) {};
    }
    else {
        while(1) {};
    }
}
```

What is the output of this program?

Part III (4 points)

```
int counter = 0;

void handler(int sig)
{
    counter ++;
}


int  main()
{
    int i;
    pid_t p;

    signal(SIGCHLD, handler);

    for (i = 0; i < 5; i ++){
        if ((p = fork()) == 0){
            exit(0);
        }
    }

    /* wait for last created child to die */
    waitpid(p, NULL, 0);

    printf("counter = %d\n", counter);
    return 0;
}
```

A. Does the program output the same value of `counter` every time we run it?   `Yes`   `No`

B. If the answer to A is `Yes`, indicate the value of the `counter` variable. Otherwise, list all possible values of the `counter` variable.

Answer: `counter` = ______________________

Problem 3. (12 points):

The following problem concerns the way virtual addresses are translated into physical addresses.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Virtual addresses are 16 bits wide.
- Physical addresses are 12 bits wide.
- The page size is 1024 bytes.
- The TLB is 4-way set associative with 16 total entries.
- The cache is 2-way set associative, with a 4 byte line size and 8 total lines.

In the following tables, **all numbers are given in hexadecimal**. The contents of the TLB, the page table for the first 32 pages, and the cache are as follows:

TLB			
Index	Tag	PPN	Valid
0	4	0	1
	2	2	1
	0	1	1
	5	3	0
1	4	1	0
	7	3	0
	5	2	0
	3	1	0
2	0	3	0
	3	1	0
	2	0	0
	7	1	0
3	6	1	0
	3	1	0
	7	3	0
	2	2	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	0	1	10	0	1
01	1	0	11	2	0
02	3	0	12	2	1
03	1	1	13	1	0
04	2	0	14	0	0
05	2	1	15	2	0
06	1	0	16	1	0
07	3	1	17	0	0
08	2	1	18	1	1
09	1	0	19	2	0
0A	3	0	1A	1	0
0B	2	0	1B	3	0
0C	0	0	1C	3	0
0D	1	0	1D	2	0
0E	1	1	1E	3	0
0F	0	0	1F	1	0

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	19	1	99	11	23	11	00	0	99	11	23	11
1	15	0	4F	22	EC	11	2F	1	55	59	0B	41
2	1B	1	00	02	04	08	0B	1	01	03	05	07
3	06	0	84	06	B2	9C	FF	0	84	06	B2	9C

Part 1

A. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

B. The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPO	The physical page offset
PPN	The physical page number
CO	The block offset within the cache line
CI	The cache index
CT	The cache tag

11	10	9	8	7	6	5	4	3	2	1	0

Part 2

For the given virtual address, indicate the TLB entry accessed, the physical address, and the cache byte value returned **in hex**. Indicate whether the TLB misses, whether a page fault occurs, and whether a cache miss occurs.

If there is a cache miss, enter “-” for “Cache Byte returned”. If there is a page fault, enter “-” for “PPN” and leave parts C and D blank.

Virtual address: `1FFD`

A. Virtual address format (one bit per box)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

B. Address translation

Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

C. Physical address format (one bit per box)

11	10	9	8	7	6	5	4	3	2	1	0

D. Physical memory reference

Parameter	Value
Byte offset	0x
Cache Index	0x
Cache Tag	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Problem 4. (10 points):

The following problem concerns dynamic storage allocation.

Consider an allocator that uses an implicit free list. The layout of each free memory block is as follows:

```
               31                             2 1 0
                ___________________________________
     Header    |     Block Size (bytes)      |     |
               |_____________________________|_____|
               |                                   |
               |                                   |
               |                                   |
               |                                   |
               |                                   |
               |___________________________________|
     Footer    |     Block Size (bytes)      |     |
               |_____________________________|_____|
```

The layout for an allocated memory block is the same except for that there is no footer. Each memory block, either allocated or free, has a size that is a multiple of eight bytes. Thus, only the 29 higher order bits in the header and footer are needed to record block size, which includes the header and footer. The usage of the remaining 3 lower order bits is as follows:

- `bit 0` indicates the use of the current block: 1 for allocated, 0 for free.
- `bit 1` indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
- `bit 2` is unused and is always set to be 0.

Given the contents of the heap shown on the left, show the new contents of the heap (in the right table) after a call to `free(0x400b008)` is executed. Your answers should be given as hex values. Note that the address grows from bottom up. Assume that the allocator uses immediate coalescing, that is, adjacent free blocks are merged immediately each time a block is freed.

Address	
0x400b028	0x00000012
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	0x00000012
0x400b018	0x400b412a
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	0x400b531c
0x400b008	0x400b52cc
0x400b004	0x0000001B
0x400b000	0x400b511c
0x400affc	0x0000000B

Address	
0x400b028	
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	
0x400b018	
0x400b014	0x400b511c
0x400b010	
0x400b00c	0x400b531c
0x400b008	
0x400b004	
0x400b000	
0x400affc	

Problem 5. (12 points):

This problem concerns deadlocking threads.
In some of the following five examples of parallell executing threads, there is a risk for deadlock.
In all five examples initially: `a = 1`, `b = 1`, `c = 1`

Example A

```
Thread 1:              Thread 2:
   P(a)                   P(c)
   P(b)                   P(b)
   P(c)                   V(b)
   V(a)                   V(c)
   V(c)
   V(b)
```

Example B

```
Thread 1:              Thread 2:
   P(a)                   P(c)
   P(b)                   P(a)
   V(a)                   V(a)
   P(c)                   V(c)
   V(c)
   V(b)
```

Example C

```
Thread 1:              Thread 2:
   P(a)                   P(c)
   P(c)                   P(b)
   V(c)                   V(b)
   V(a)                   V(c)
```

Example D

```
Thread 1:            Thread 2:            Thread 3:
   P(a)                 P(c)                 P(b)
   P(c)                 P(b)                 P(a)
   V(a)                 V(b)                 V(b)
   P(b)                 V(c)                 V(a)
   V(c)
   V(b)
```

Example E

```
Thread 1:            Thread 2:            Thread 3:
   P(a)                 P(b)                 P(c)
   V(a)                 P(c)                 P(a)
   P(b)                 V(b)                 V(a)
   P(c)                 V(c)                 V(c)
   V(c)
   V(b)
```

For each of the five examples, circle whether(Y) or not(N) it might deadlock. For the once that might deadlock, suggest a change to the locking order so that there is no risc for deadlock.

A. Y N How to avoid deadlock:

B. Y N How to avoid deadlock:

C. Y N How to avoid deadlock:

D. Y N How to avoid deadlock:

E. Y N How to avoid deadlock: