Namn: _____

Person-nummer: _____

Systems programming and Operating systems, 2005

Tentamen 2005-12-14

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!

Problem 1. (12 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 {
      pid_t pid; int status;
      if ((pid = wait(&status)) > 0) {
        printf("4");
      }
    }
  }
  else {
    if (fork() == 0) {
      printf("1");
      exit(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. 2030401 B. 1234000 C. 2300140

D. 2034012 E. 3200410 F. 3010240

Problem 2. (8 points):

This problem tests your understanding of exceptional control flow in C programs.
For problems A-C, indicate how many "hello" output lines the program would print.
Caution: Don't overlook the `printf` *function in* `main`.

Problem A

```
void doit() {
  fork();
  fork();
  printf("hello\n");
  return;
}

int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

Answer: _____ output lines.

Problem B

```
void doit() {
  if (fork() == 0) {
    fork();
    printf("hello\n");
    exit(0);
  }
  return;
}

int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

Answer: _____ output lines.

Problem C

```
void doit() {
  if (fork() == 0) {
    fork();
    printf("hello\n");
    return;
  }
  return;
}

int main() {
  doit();
  printf("hello\n");
  exit(0);
}
```

Answer: _____ output lines.

For problem D, indicate the value of the `counter` variable that the program would print.

Problem D

```
int counter = 1;

int main() {

  if (fork() == 0) {
    counter--;
    exit(0);
  }
  else {
    wait(NULL);
    counter++;
    printf("counter = %d\n", counter);
  }
  exit(0);
}
```

Answer: `counter = _____`.

Problem 3. (8 points):

This problem concerns the following four versions of the `tfgets` routine, a timeout version of the Unix `fgets` routine.

The `tfgets` routine waits for the user to type in a string and hit the return key. If the user enters the string within 5 seconds, the `tfgets` returns normally with a pointer to the string. Otherwise, the routine "times out" and returns a NULL string.

`tfgets`: Version A

```
void handler(int sig) {
  siglongjmp(env, 1);
}

char *tfgets(char *s, int size, FILE *stream) {
  pid_t pid;
  signal(SIGCHLD, handler);

  if (!sigsetjmp(env, 1)) {
    pid = fork();
    if (pid == 0) {
      return fgets(s, size, stream);
    }
    else {
      sleep(5);
      kill(pid, SIGKILL);
      wait(NULL);
      return NULL;
    }
  }
  else {
    wait(NULL);
    exit(0);
  }
}
```

`tfgets`: **Version B**

```
void handler(int sig) {
  wait(NULL);
  siglongjmp(env,1);
}

char *tfgets(char *s, int size, FILE *stream) {
  pid_t pid;

  signal(SIGUSR2, handler);
  if (sigsetjmp(env, 1) != 0)
    return NULL;
  if ((pid = fork()) == 0) {
    sleep(5);
    kill(getppid(), SIGUSR2);
    exit(0);
  }
  fgets(s, size, stream);
  kill(pid, SIGKILL);
  wait(NULL);
  return s;
}
```

`tfgets`: **Version C**

```
void handler(int sig) {
  wait(NULL);
  siglongjmp(env, 1);
}

char *
tfgets(char *s, int size, FILE *stream) {
  pid_t pid;
  str = NULL;
  signal(SIGCHLD, handler);

  if ((pid = fork()) ==  0) {
    sleep(5);
    exit(0);
  }
  else {
    if (sigsetjmp(env, 1) == 0) {
      str = fgets(s, size, stream);
      kill(pid, SIGKILL);
      pause();
    }
    return str;
  }
}
```

`tfgets`: Version D

```
void handler(int sig) {
  wait(NULL);
  siglongjmp(env, 1);
}

char *
tfgets(char *s, int size, FILE *stream) {
  pid_t pid;
  str = NULL;
  signal(SIGCHLD, handler);

  if ((pid = fork()) ==  0) {
    sleep(5);
    return NULL;
  }
  else {
    if (sigsetjmp(env, 1) == 0) {
      str = fgets(s, size, stream);
      kill(pid, SIGKILL);
      pause();
    }
    return str;
  }
}
```

Some of the preceding four versions of `tfgets` are correct, and others are flawed because the author didn't understand basic concepts of concurrency and signaling.

Circle the versions that are correct, in the sense that they return the input string if typed within 5 seconds, timeout after 5 seconds by returning NULL, and correctly reap their terminated children.

`Version A` `Version B` `Version C` `Version D`

Note: The `pause` function sleeps until a signal is received and then returns.

Problem 4. (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 14 bits wide.
- The page size is 1024 bytes.
- The TLB is 4-way set associative with 16 total entries.

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

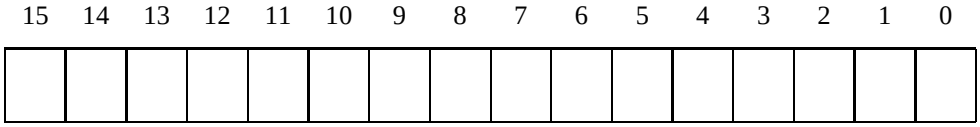
TLB			
Index	Tag	PPN	Valid
0	8	7	1
	F	6	1
	0	3	0
	1	F	1
1	1	E	1
	2	7	0
	7	3	0
	B	1	1
2	0	0	0
	C	1	0
	F	8	1
	7	6	1
3	8	4	0
	3	5	0
	0	D	1
	2	9	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	2	0	10	1	1
01	5	1	11	3	0
02	7	1	12	9	0
03	9	0	13	7	1
04	F	1	14	D	1
05	3	1	15	5	0
06	B	0	16	E	1
07	D	1	17	6	0
08	7	1	18	1	0
09	C	0	19	0	1
0A	3	0	1A	8	1
0B	1	1	1B	C	0
0C	0	1	1C	0	0
0D	D	0	1D	2	1
0E	0	0	1E	7	0
0F	1	0	1F	3	0

Part 1

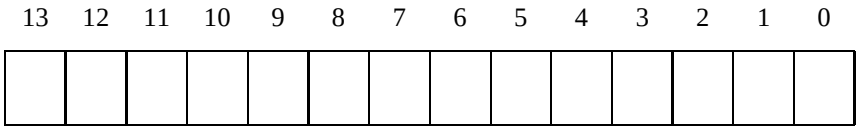
A. The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

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



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



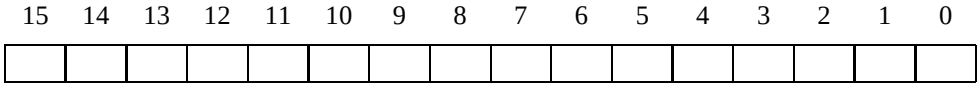
Part 2

For the given virtual addresses, indicate the TLB entry accessed and the physical address. Indicate whether the TLB misses and whether a page fault occurs.

If there is a page fault, enter "-" for "PPN" and leave part C blank.

Virtual address: `2F09`

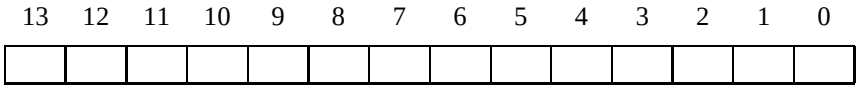
A. Virtual address format (one bit per box)



B. Address translation

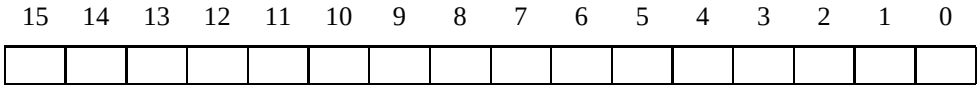
Parameter	Value
VPN	0x
TLB Index	0x
TLB Tag	0x
TLB Hit? (Y/N)	
Page Fault? (Y/N)	
PPN	0x

C. Physical address format (one bit per box)



Virtual address: `0C53`

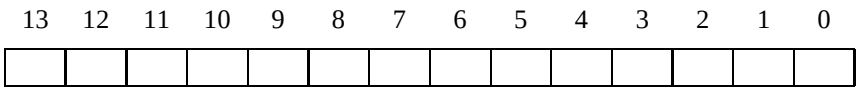
A. Virtual address format (one bit per box)



B. Address translation

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

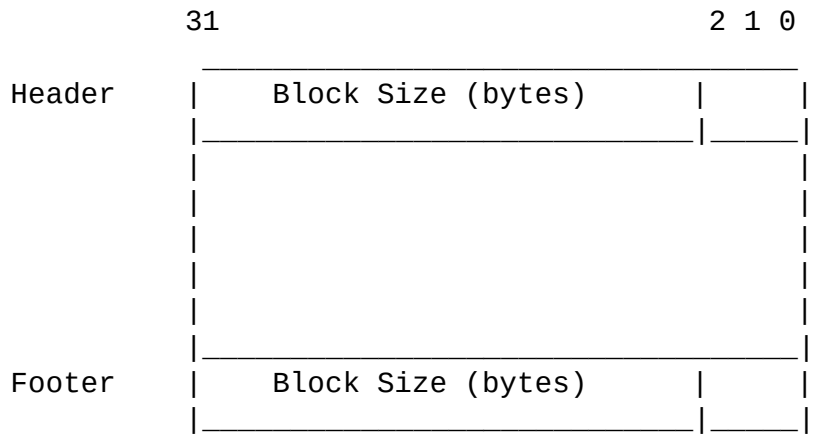
C. Physical address format (one bit per box)



Problem 5. (10 points):

The following problem concerns dynamic storage allocation.

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



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(0x400b010)` 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	0x00000013
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	0x00000013
0x400b008	0x00000013
0x400b004	0x400b601c
0x400b000	0x400b511c
0x400affc	0x00000013

Address	
0x400b028	
0x400b024	0x400b611c
0x400b020	0x400b512c
0x400b01c	
0x400b018	
0x400b014	0x400b511c
0x400b010	0x400b601c
0x400b00c	
0x400b008	
0x400b004	0x400b601c
0x400b000	0x400b511c
0x400affc	

Problem 6. (10 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)
   V(b)                   V(b)
   P(c)                   V(c)
   V(c)
   V(a)
```

Example B

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

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(a)
   P(b)               P(b)               P(c)
   V(b)               V(b)               V(a)
   P(c)               V(c)               V(c)
   V(c)
   V(a)
```

Example E

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

For each of the five examples, circle whether(Y) or not(N) it might deadlock.

A.	Y	N
B.	Y	N
C.	Y	N
D.	Y	N
E.	Y	N