University of Waterloo
Midterm Examination
Term: Winter Year: 2013

Student Family Name _____

Student Given Name _____

Student ID Number _____

Section : Circle one (Brecht 11:30) (Brecht 1:00)

Course Abbreviation and Number:	CS 350
Course Title:	Operating Systems
Section(s):	2
Instructors:	Tim Brecht

Date of Exam:	March 6, 2013
Time Period	Start time: 7:00 pm End time: 9:00 pm
Duration of Exam:	120 minutes
Number of Exam Pages:	9 (including cover sheet)
	NO CALCULATORS, NO ADDITIONAL MATERIAL

Problem	Topic	Marks	Score	Marker's Initials
1	General	8		
2	Processes and fork	12		
3	Stack State and Interrupts	12		
4	Threads and Synchronization	10		
5	Synchronization and Deadlock	10		
6	OS/161 and Address Translation	10		
7	Segmentation and Paging	14		
Total		76		

Problem 1 (8 marks)

a. **(2 mark(s))** You are working on an operating system for a new machine. The processor in this system uses 36 bits for virtual and physical addresses and it has three options for different page sizes:

(a) 4096 bytes (4 KB)

(b) 8192 bytes (8 KB)

(c) 65536 bytes (64 KB)

Each of these choices leaves a different number of bits available for the virtual page number and some team members are arguing about which choice allows the largest amount of virtual memory to be addressed. Explain which option, if any, provides the largest amount of virtual memory to be addressed and why.

b. **(2 mark(s))** Can a single thread have more than one address space? **Explain your answer.**

c. **(2 mark(s))** In a system that implements paging, the processor uses 34-bit virtual addresses, 40-bit physical addresses and a page size of 8 KB. How many bits are needed to represent the physical frame? **Explain your answer.**

d. **(2 mark(s))** Explain why it is not a good idea to wake up more than one thread when implementing `lock_release`.

Problem 2 (12 marks)

For the program shown below, fill in the blanks at the bottom of the page to indicate how many characters of each letter will be printed in total when the program finishes running. If a range of values is possible, give the range. If it is not possible to determine the number or a range, state so and explain why. Assume that all function, library and system calls are successful. Use the space to the right of the program to draw a diagram of the process hierarchy that results during execution. Use that diagram to explain how you arrived at your answer. **NO MARKS WILL BE GIVEN UNLESS A PROPER DIAGRAM AND EXPLANATION ARE PROVIDED.**

```
#include <stdio.h>
#include <unistd.h>

main()
{
  int rc1, rc2, rc3;

  rc1 = fork();
  rc2 = fork();


  if (rc1 == 0) {
    printf("A");
    rc2 = 0;
    rc3 = fork();
  }

  if (rc2 == 0) {
    printf("B");
  } else {
    printf("C");
  }
  printf("D");
}
```

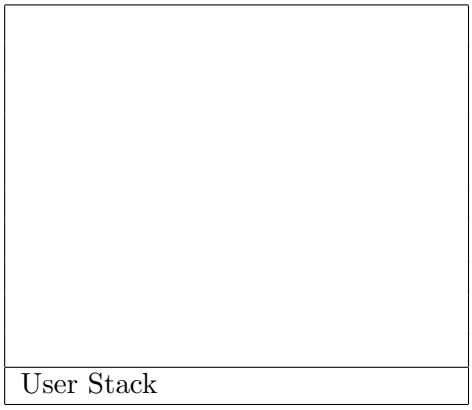
Total number of printed A's _____ B's _____ C's _____ D's _____

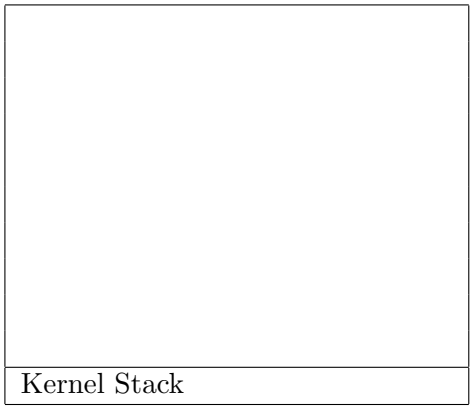
Problem 3 (12 marks)
Assume one **user-level process** (named P1) executes the code shown below on OS161.

```
main()          Q()             R()                     S()
{               {               {                       {
  Q();             S();            S();                    int i, x;
  R();            }               printf("Hello\n");       for (i=0; i<N; i++) {
}                                 }                           x = x + i;
                                                          }
                                                        }
```

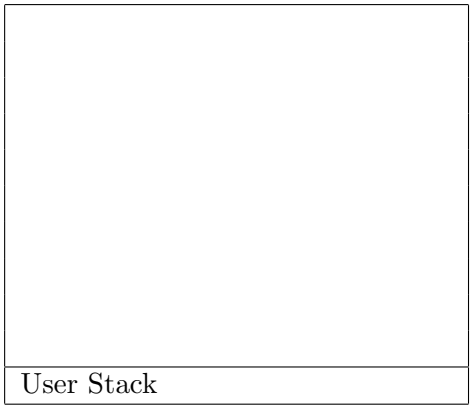
In the rectangles shown for each part of this question below, fill in and label any information about the state of the **user-level stack and the kernel stack** for the executing process (P1) as they would appear at the point in time stated in the question. Do not draw anything that has been popped from the stacks (is no longer active) and use the same level of detail used in class and the course notes. Be sure to show any stack frames, trap frames, and thread contexts, if they are present. Draw the stack so that the high addresses are at the top of the diagram and low addresses are at the bottom. Recall that the stack grows from high addresses to low.

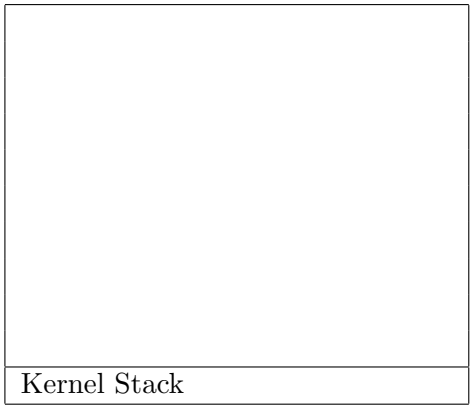
a. **(8 mark(s))** The process P1 calls `main`, `Q`, and `S` at which point it is interrupted (while still executing `S`) and a context switch to another process (P2) occurs. Show the state of the stacks for P1 after the context switch to P2 has completed.





b. **(4 mark(s))** Now assume that the thread for P1 is later dispatched and it resumes execution. Show the state of the stacks for P1 as they would appear after running to the point in the code just after returning from `S()` but before the call to `printf("Hello")`.





Problem 4 (10 marks)
Consider executing the code below when answering the questions on this page.

```
struct semaphore *sem1, *sem2;
main()
{
   sem1 = sem_create("sem1", 2);
   sem2 = sem_create("sem2", 4);

   for(i=0; i<10; i++) {
      thread_fork("A",NULL,i,A,NULL);
   }
   for(i=0; i<7; i++) {
      thread_fork("B",NULL,i,B,NULL);
   }
}
void A(void *x, unsigned long y)
{
  P(sem1);
  printf("A");
  P(sem2);
  printf("C");
}
```

```
 void B(void *x, unsigned long y)
{
   printf("B");
   P(sem2);
   printf("D");
   V(sem1);
}
```

For each of the substrings of output below indicate, by circling the appropriate response, whether or not the output IS POSSIBLE, IS NOT POSSIBLE, or CAN NOT BE DETERMINED. Assume that the first character shown is the first character printed when the threads start running and that not all of the output is shown (i.e., all threads have not finished executing). If you choose IS NOT POSSIBLE, or CAN NOT BE DETERMINED **briefly explain why**.

ACBDBABBBDCAC [IS POSSIBLE] [IS NOT POSSIBLE] [CAN NOT DETERMINED]

BABABCBBCBB [IS POSSIBLE] [IS NOT POSSIBLE] [CAN NOT DETERMINED]

BABABCBCBCB [IS POSSIBLE] [IS NOT POSSIBLE] [CAN NOT DETERMINED]

ABAABBBBDD [IS POSSIBLE] [IS NOT POSSIBLE] [CAN NOT DETERMINED]

ABABBCCDBDAA [IS POSSIBLE] [IS NOT POSSIBLE] [CAN NOT DETERMINED]

Problem 5 (10 marks)

Consider the code below when answering the questions on this page. Assume that the locks are all initialized properly before being used (as shown in the function `init()`) and that `funcA()` and `funcB()` do not do anything that could produce a deadlock. For each of the scenarios in the questions below state whether or not deadlock CAN or CAN NOT occur **and explain why**. Each scenario/question is separate (i.e., the locks and threads are reinitialized for each part of the question).

```
struct lock *A[N],
struct lock *B[N];
void init()
{
  for (i=0; i<N; i++) {
    A[i] = lock_create("NoName");
    B[i] = lock_create("NoName");
  }
}
void ProcB(int i, int j)
{
    assert(i > j);
    assert(i >= 0 && i < N);
    assert(j >= 0 && j < N);
    lock_acquire(A[i]);
      lock_acquire(B[j]);
        funcB(i,j);
      lock_release(B[j]);
    lock_release(A[i]);
}
```

```
void ProcA()
{
  for (i=0; i<N; i++) {
    lock_acquire(A[i]);
      lock_acquire(B[i]);
        funcA();
      lock_release(A[i]);
    lock_release(B[i]);
  }
}
```

a. **(6 mark(s))** A bunch of threads are created and they only call `ProcA()`.

b. **(4 mark(s))** A bunch of threads are created and they only call `ProcB()`. When they call `ProcB()` the value of `i` is always greater than `j` and both `i` and `j` are always between `0` and `N-1` (inclusive). In other words, the assertions are never triggered.

Problem 6 (10 marks)

The structure `addrspace` shown below describes the address space of a running process on a 32-bit MIPS processor similar to the `dumbvm` provided in OS161. The virtual page size is 4096 (`0x1000`) bytes. In this implementation, the compiler, linker and operating system use different locations for text, data and stack segments than those used by the version of OS161 and the toolchains you are using this term. Fortunately, this new version of the OS161 kernel now explicitly represents the stack as segment 3 (note the stack size).

```
struct addrspace {
    vaddr_t as_vbase1 = 0x10000000;       /* text segment: virtual base address */
    paddr_t as_pbase1 = 0x00010000;       /* text segment: physical base address */
    size_t as_npages1 = 0x200;            /* text segment: number of pages */
    vaddr_t as_vbase2 = 0x20000000;       /* data segment: virtual base address */
    paddr_t as_pbase2 = 0x80000000;       /* data segment: physical base address */
    size_t as_npages2 = 0x137;            /* data segment: number of pages */
    vaddr_t as_vbase3 = 0x70000000;       /* stack segment: virtual base address */
    paddr_t as_pbase3 = 0x10000000;       /* stack segment: physical base address */
    size_t as_npages3 = 0x18;             /* stack segment: number of pages */
};
```

For an application executing in user space that uses the address space defined above, assume that it is accessing the specified addresses below. When possible you are to translate the provided address. If the translation is not possible, explain why it is not possible and what would happen during translation. If the translation is possible indicate which segment the address belongs to. Use 32-bit hexadecimal notation for all addresses.
Some possibly useful values:

```
  1 * 4096 =   0x1000     2 * 4096 =   0x2000    10 * 4096 =   0xA000
 16 * 4096 =  0x10000    32 * 4096 =  0x20000   100 * 4096 =  0x64000
128 * 4096 =  0x80000   256 * 4096 = 0x100000   512 * 4096 = 0x200000
```

a. **(2 mark(s))** Translate the **Virtual** Address `0x70016429` to a **Physical** Address.

b. **(2 mark(s))** Translate the **Virtual** Address `0x7FFF1289` to a **Physical** Address.

c. **(2 mark(s))** Translate the **Physical** Address `0x80000080` to a **Virtual** Address.

d. **(2 mark(s))** Translate the **Physical** Address `0x10013F39` to a **Virtual** Address.

e. **(2 mark(s))** Translate the **Virtual** Address `0x80000080` to a **Physical** Address.

Problem 7 (14 marks)

Consider a processor that uses segmentation and paging (i.e., this is not a MIPS processor). Below is the segment table being used for the currently executing process and below that are pages tables for several processes in the system. Note that some processes may not use all of the available segments. Recall that VPN is the virtual page number, PFN is the physical frame number, V is the valid bit and D is the dirty bit (i.e., the page can be dirtied/modified).

Segment	PT base addr	Max VPN Value
4	70700000	3
3	70200000	3
2	70500000	3
1	70300000	3
0	70100000	3

VPN	PFN	V	D
3	5177	1	0
2	20	0	0
1	77	1	0
0	4251	0	0

Base addr: 70000000

VPN	PFN	V	D
3	1311	1	0
2	12	1	0
1	711	0	0
0	23	1	0

Base addr: 70700000

VPN	PFN	V	D
3	52	1	1
2	41	1	1
1	30	1	1
0	5177	0	1

Base addr: 70400000

VPN	PFN	V	D
3	65	1	1
2	77	1	1
1	567	1	1
0	672	1	1

Base addr: 70300000

VPN	PFN	V	D
3	641	0	1
2	753	1	1
1	2577	1	1
0	517	1	1

Base addr: 70200000

VPN	PFN	V	D
3	5532	0	1
2	5177	1	1
1	336	0	1
0	77	1	1

Base addr: 70600000

VPN	PFN	V	D
3	5177	1	1
2	34	1	1
1	563	1	1
0	1641	1	1

Base addr: 70500000

VPN	PFN	V	D
3	516	1	1
2	37	0	1
1	7731	1	1
0	6341	1	1

Base addr: 70100000

For the first parts of this question (parts a – e) assume that the processor is using 32-bits for virtual and physical addresses, that the page size is 64 KB, that all addresses (virtual and physical) and values shown in the segment table and page tables are expressed in hexadecimal, and that the system uses 4 bits for segments.

a. **(2 mark(s))** Explain how many **bits** of the virtual address will be used to represent the offset?

b. **(2 mark(s))** What is the **maximum possible size** of a segment in this system in **bytes** (expressed as an equation).

c. **(2 mark(s))** Convert the **virtual address** 0x20043751 into a 32-bit physical addresses (also expressed in hexadecimal). Show your work and if the address can not be translated, explain why.

d. **(2 mark(s))** Convert the **virtual address** 0x30022267 into a 32-bit physical addresses (also expressed in hexadecimal). Show your work and if the address can not be translated, explain why.

e. **(2 mark(s))** Convert the **physical address** 0x51773721 into a 32-bit virtual addresses (also expressed in hexadecimal). Show your work and if the address can not be translated, explain why.

For the remaining part of this question assume that the processor is using 24-bits for virtual and physical addresses, that the page size is 512 bytes, that all values shown in the segment table and the page tables are to be interpreted as octal values (i.e., each character represents 3 bits) and that the system uses 3 bits for segments. Note that some processes may not use all of the available segments.

f. **(4 mark(s))** Convert the **physical address** 00077356 (expressed in octal) into a 24-bit virtual address (also expressed in octal). Show your work and if the address can not be translated, explain why.