University of Waterloo
Midterm Examination
Term: Winter Year: 2012

Student Family Name	
Student Given Name	
Student ID Number	
Section : Circle one	(Brecht 8:30) (Brecht 11:30)

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

Date of Exam:	March 5, 2012
Time Period	Start time: 7:00 pm End time: 9:00 pm
Duration of Exam:	120 minutes
Number of Exam Pages:	11 (including cover sheet)
	NO CALCULATORS, NO ADDITIONAL MATERIAL

Problem	Topic	Marks	Score	Marker's Initials
1	General	10		
2	Processes, fork, exec	17		
3	Threads and Synchronization	20		
4	Deadlock and Synchronization	18		
5	OS/161 and Address Translation	10		
6	TLBs and Address Translation	12		
Total		87		

Problem 1 (10 marks)

a. **(2 mark(s))** Describe the different sources of overheads that are incurred when an application makes a BLOCKING system call in OS/161.

b. **(2 mark(s))** Explain what it means for a synchronization mechanism to be "starvation free".

c. **(2 mark(s))** Did you read the paper "An Introduction to Programming with Threads", or "An Introduction to Programming with C# Threads", by Andrew Birrell? If yes did you find the paper useful? Explain why or why not. If you did not read the paper, explain why you didn't.

d. **(2 mark(s))** In whichever version of the Birrell paper you read what is the name of the function used to wait for another thread to finish?

e. **(2 mark(s))** According to the Birrell paper you read, what are a spurious wake-ups and what is one example cause of spurious wake-ups?

Problem 2 (17 marks)
For the programs shown, fill in the blanks below each program 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 each program to draw a diagram of the process hierarchy that results during execution and to explain how you arrived at your answer. **NO MARKS WILL BE GIVEN UNLESS A PROPER DIAGRAM AND EXPLANATION ARE PROVIDED.**

a. **(9 mark(s))**

```
/* forkprint1.c : source code for the program forkprint1 */

int main(int argc, char *argv[])
{
  int rc;
  int status;

  char *args[3];
  args[0] = (char *) "pgm";
  args[1] = (char *) "X";
  args[2] = (char *) 0;

  rc = fork();
  printf("A");
  if (rc != 0) {
    rc = fork();
  }
  printf("B");

  rc = fork();
  printf("C");
  if (rc == 0) {
    /* exec the program shown below */
    rc = execv("pgm", args);
    printf("D");
  } else {
    waitpid(rc, &status, 0);
  }
  printf("E");
}
/*-------------------------------------------------------------------------------------*/
/* pgm.c : source code for "pgm". Used above by forkprint1.c  */

int main(int argc, char *argv[])
{
  printf("%s", argv[1]);
}
```

Total number of printed A's _______, B's __________, C's _________, D's _________, E's __________, X's __________

b. **(8 mark(s))**

```
/* forkprint2.c : source code for the program forkprint2 */

char *str = "X";

main()
{
  int rc;

  rc = fork();
  printf("A");
  rc = fork();
  printf("B");
  if (rc == 0) {
    str = "Z";
    printf("C");
    rc = fork();
  }
  printf("%s", str);
}
```

Total number of printed A's ________, B's ___________, C's __________, X's __________, Z's ___________

Problem 3 (20 marks)
Below you will find some programs that may or may not be implemented correctly. Assume that the file `neededstuff.h` contains any definitions that are missing from the existing code (e.g., `T`, `DATA_SIZE`, etc.).

In each case below you are to look at the solution provided to the described problem and to determine whether or not the solution will always produce the correct result. If the solution will not produce the correct result add to and/or change the code (including necessary declarations and initialization code) so that it does produce the correct result. Use only those synchronization primitives available in OS/161 after assignment 1 has been completed. Do not disable and/or enable interrupts. If the solution is correct just write CORRECT on the solution. Assume that any function, library, or system calls always succeed.

a. **(8 mark(s))** This program should print the maximum value found in `array`.

```
#include "neededstuff.h"
unsigned int max = 0;
unsigned int array[DATA_SIZE];


main()
{
  int i = 0;

  for (i=0; i<T; i++) {

    thread_fork("findmax", findmax, UNUSED_PTR, i, NORETURN_VAL);

  }



  /* PRINT THE MAXIMUM VALUE FOUND IN ARRAY */
  printf("Maximum value = %u\n", max);
}

void findmax(void *unused, unsigned long threadnum)
{
  int i = 0; int size = DATA_SIZE/T; /* assume this divides evenly */
  int start = size * (int) threadnum;

  for (i=start; i<start+size; i++) {


     if (array[i] > max) {

         max = array[i];

     }


  }

}
```

b. **(12 mark(s))** This program should place values in an array (concurrently) and then compute the sum of those values (also concurrently). Once the sum has been computed it should be printed.

```
#include "neededstuff.h"
int array[DATA_SIZE];


main()
{
  int i = 0;
  int grand_total = 0;


  for (i=0; i<T; i++) {

    thread_fork("populate", populate, UNUSED_PTR, i, NORETURN_VAL);

  }



  for (i=0; i<T; i++) {

    thread_fork("dosum", dosum, UNUSED_PTR, i, NORETURN_VAL);

  }



  /* SHOULD PRINT SUM OF ALL ELEMENTS OF ARRAY */
  printf("Sum of all elements of array = %d\n", grand_total);

}

void populate(void *unused, unsigned long threadnum)
{
  int i = 0; int size = DATA_SIZE/T; /* assume this divides evenly */
  int start = size * (int) threadnum;


  for (i=start; i<start+size; i++) {


     array[i] = i;


  }


}
```

continued ...

```
void dosum(void *unused, unsigned long threadnum)
{
  int i = 0;
  int size = DATA_SIZE/T;       /* assume this divides evenly */
  int start = size * (int) threadnum;
  int sum = 0



  for (i=start; i<start+size; i++) {


     sum = sum + array[i];



  }



  grand_total = grand_total + sum;




}
```

Problem 4 (18 marks)
Several different versions of a function to transfer funds from one bank account to another are shown below. Your manager wants you to evaluate each of them to see if any one of them could be used safely and correctly with multiple threads. When evaluating any one of them it would be the only one used and the others would not be called. Assume that all function, library and system calls succeed and that appropriate missing definitions and initializations are provided and/or called elsewhere. Your manager wants you to answer the following questions and to explain your answer:

- Can the use of the procedure result in a deadlock? Circle the answer and describe why or why not.
- Will the procedure transfer the specified amount between the two accounts correctly (aside from possible deadlock problems) and maintain correct balances? Circle the answer and describe why or why not.

NOTE: several locks are declared and/or used differently in each of the different procedures so you need to carefully read the code to see which locks are being used.

```
struct lock *from_lock;
struct lock *to_lock;

struct account {
  struct lock *lock;
  volatile int balance;         /* THIS WAS NOT BUT SHOULD HAVE BEEN DECLARED VOLATILE */
  int num;
};

init()
{
   from_lock = lock_create("from_lock");
   to_lock = lock_create("to_lock");
}
```

a. **(6 mark(s))**
```
void transfer1(struct account *from_account, struct account *to_account, int amount)
{
   lock_acquire(from_lock);
     lock_acquire(to_lock);
       from_account->balance -= amount;
       to_account->balance += amount;
     lock_release(to_lock);
   lock_release(from_lock);
}
```
Deadlock: YES NO

Correct: YES NO

b. **(6 mark(s))**

```
void transfer2(stuct account *from_account, struct account *to_account, int amount)
{
   lock_acquire(from_account->lock);
     lock_acquire(to_account->lock);
       from_account->balance -= amount;
       to_account->balance += amount;
     lock_release(to_account->lock);
   lock_release(from_account->lock);
}
```

Deadlock: YES NO

Correct: YES NO

c. **(6 mark(s))**

```
void transfer3(struct account *from_account, struct account *to_account, int amount)
{
   struct lock *from = create_lock("from");  /* typo in original was missing (*) ptr */
   struct lock *to = create_lock("to");      /* typo in original was missing (*) ptr */
   lock_acquire(from);                       /* announced correction during exam     */
     lock_acquire(to);
       from_account->balance -= amount;
       to_account->balance += amount;
     lock_release(to);
   lock_release(from);
   lock_destroy(from);
   lock_destroy(to);
}
```

Deadlock: YES NO

Correct: YES NO

Problem 5 (10 marks)

The structure `addrspace` shown below describes the address space of a running process on a 32-bit MIPS processor similar to that used in the assignments. The virtual page size is 4096 (`0x1000`) bytes. Assume that the operating system gives each process a stack segment consisting of 16 pages, starting at virtual address `0x7fff0000` and ending at virtual address `0x7fffffff`.

```
struct addrspace {
    vaddr_t as_vbase1 = 0x00900000;       /* text segment: virtual base address */
    paddr_t as_pbase1 = 0x10000000;       /* text segment: physical base address */
    size_t as_npages1 = 32;               /* text segment: number of pages in decimal */
    vaddr_t as_vbase2 = 0x30000000;       /* data segment: virtual base address */
    paddr_t as_pbase2 = 0x00400000;       /* data segment: physical base address */
    size_t as_npages2 = 512;              /* data segment: number of pages in decimal */
    paddr_t as_stackpbase = 0x80000000;   /* stack segment: physical base address */
};
```

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 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 `0x0090A8C0` to a **Physical** Address.

b. **(2 mark(s))** Translate the **Virtual** Address `0x7FFFB495` to a **Physical** Address.

c. **(2 mark(s))** Translate the **Physical** Address `0x00519BCF` to a **Virtual** Address.

d. **(2 mark(s))** Translate the **Physical** Address `0x8000100A` to a **Virtual** Address.

e. **(2 mark(s))** Translate the **Virtual** Address `0x8000100A` to a **Physical** Address.

Problem 6 (12 marks)
Some possibly useful info: $2^{10} = 1$ KB, $2^{20} = 1$ MB, $2^{30} = 1$ GB

In this question all addresses, virtual page numbers and physical frame numbers are represented in octal. Recall that each octal character represents 3 bits. Consider a machine with *33-bit* virtual addresses and a page size of 32768 bytes (32 KB). During a program's execution the TLB contains the following entries (all in octal).

Virtual Page Num	Physical Frame Num	Valid	Dirty
6125	1234567	1	0
61252	123456	0	0
612	3013	1	1
612521	765432	1	1

If possible, explain how the MMU will translate the virtual addresses given below (in octal) into a *36-bit* physical address (in octal). If it is not possible, explain what will happen and why. Show and **explain how you derived your answer.** Express the final physical address using **all 36-bits**.

a. **(3 mark(s))** Load from virtual address = 61252127604.

b. **(3 mark(s))** Store to virtual address = 61252127.

c. **(3 mark(s))** Store to virtual address = 612503714.

d. **(3 mark(s))** Can a store be performed on the **physical** address = 12345671032? If yes, provide the virtual address used to access this physical address and if not explain precisely why not.