NAME: _____ UID: _____

CS 4400: Computer Systems

Final Exam
Fall 2022

- The exam is open-book and open-notes.
- Make certain that you have **eleven** pages, with each page printed on the front only.
- Write your full name and UID on this page. *Do not write your name or UID on any other page.*
- The point value of each question is clearly marked, so allocate your time wisely. The exam is worth a total of 100 points. *1 point ≡ 1 minute*
- You have a total of 2 hours and 30 minutes for the exam, including time for scanning and uploading your PDF file on Gradescope.

1. **Matching (1 point each)** For each of the following descriptions, choose from the list of terms below the one that *best* fits the description. There is only one matching term for each description, there are more terms than descriptions, and each term should be used only once.

a. _____ Parts of an ELF file that offer a linking view of the program.

b. _____ A situation occurs when the correctness of a program depends on one thread reaching point X in its control before another thread reaches point Y.

c. _____ The domain name that always maps to the loopback address.

d. _____ The type of object file produced by the assembler.

e. _____ The approach to concurrent programming in which applications explicitly schedule their own logical flows.

f. _____ The system maps names to addresses.

g. _____ The version of a library for which the linker copies only the object modules referenced by the application when building the executable.

h. _____ The function that binds socket to connection, as a client.

i. _____ The automatic memory management technique that keeps track of whether an object X has other users, but cannot handle cyclic references.

j. _____ Condition code that indicates whether the most recent operation yielded an arithmetic carry or borrow.

k. _____ The byte order that stores the least-significant byte at memory address m, and the next least-significant byte at address $m + 1$.

l. _____ The reliable protocol for establishing connections between internet applications, which establishes a substantial layer on top of IP.

m. _____ A logical flow that runs in the context of a process.

n. _____ The type of symbol used in module (e.g., file) x but defined by some other module.

o. _____ The tool whose primary task is to translate C code to assembly code.

p. _____ The part of a URL that comes immediately after the ? character.

q. _____ A variable that protects code or a shared object, as well as enables "signaling" among threads.

r. _____ The C keyword that indicates that a value may change between different accesses, even if it does not appear to be modified.

s. _____ Optimization blocker in which two pointers may designate the same memory location.

t. _____ A thread that cannot be reaped or killed by other threads.

`accept`	aliasing	assembler	big endian
`bind`	CF	compiler	`connect`
critical region	detached	DNS	domain
dynamic	ELF	executable	`extern`
external	file descriptor	fragment	garbage collection
global	HTTP	IPv6 address	I/O multiplexing
joinable	linker	`listen`	little endian
local	`localhost`	lock	multi-threading
mutex	OF	path	peer
PIC	port	pointer arithmetic	preprocessor
processes	query	race	reference counting
relocatable	sections	segments	semaphore
SF	shared	synchronization	signal
socket	static	`static`	TCP
thread	UDP	`volatile`	ZF

2. **(8 points)** Let the address of `int arr[]` be in `%rdx` and `i` be in `%rcx`. Fill in the table below, using the first row as an example. For the assembly code, put the result of the expression in the leftmost column in a register. Use `%rax` for a pointer result and `%eax` for a `int` result.

	data type	**assembly code**
`arr + 1`	`int*`	`leaq 4(%rdx), %rax`
`*(arr + i + 1)`		
`&arr[i-3]`		
`arr[4 * i]`		
`arr + 3 * i - 5`		

3. **(8 points)** Consider the following three threads executing concurrently in the same process. Semaphore `t` has been initialized to 1, while semaphore `s` has been initialized to 0.

Thread 1	*Thread 2*	*Thread 3*
`P(&t);`	`P(&t);`	`P(&s);`
`n *= 2;`	`n += 5;`	`P(&t);`
`V(&t);`	`V(&s);`	`n -= 3;`
	`V(&t);`	`V(&t);`

Enumerate the possible values of global variable `n` after all three threads have terminated. The value of `n` is initialized to 2.

4. **(6 points)** Consider the C program below. For space reasons, we are not checking error return codes. Assume that all functions return normally, that each printed string is flushed to stdout immediately, and that the proper header files have been included.

   Process/thread graphs must be shown to get any credit.

```
int main() {
  int fds[2];
  char buffer[1];

  pipe(fds);
  if(fork() == 0) {
    if(fork() == 0) {
      read(fds[0], buffer, 1);
      printf("b");
      return 0;
    }
    printf("a");
    write(fds[1], "a", 1);
    return 0;
  }
  wait(NULL);
  dup2(fds[1], 1);
  printf("c");
  write(fds[1], "a", 1);
  return 0;
}
```

   Enumerate all possible outputs of this program.

5. **(6 points)** In C, is the following Boolean expression always true? Assume that `x` was declared to have type `uint64_t`. Explain your answer.

```
x == (uint64_t)(double)x
```

6. **(10 points)** What are all of the possible outputs of the following program? Assume that the program runs on Linux, where the child process created by fork does not include all the threads of the parent process. The child process includes only the thread that called fork.

   Process/thread graphs must be shown to get any credit.

```
#include "csapp.h"

static sem_t done;

static void *go(void *p) {
  pid_t pid = Fork();
  if (pid == 0) {
    Write(1, "a", 1);
    V(&done);
    exit(0);
  } else {
    Write(1, "b", 1);
    Waitpid(pid, NULL, 0);
    V(&done);
    return NULL;
  }
}

int main() {
  pid_t pid;
  pthread_t th1, th2;
  int fds[2];

  Sem_init(&done, 0, 0);

  Write(fds[1], "-", 1);
  Pthread_create(&th1, NULL, go, NULL);
  Pthread_create(&th2, NULL, go, NULL);

  P(&done);
  P(&done);
  Write(1, "+", 1);

  return 0;
}
```

7. **(12 points)** Consider the following two modules of a program.

```
/* Module 1: main.c */

#include <stdio.h>

extern int n;
int x = 3;

int square();

int main() {
  x = n;
  printf("%i squared is %i.", x, square(x));

  return 0;
}
```

```
/* Module 2: square.c */

int square(int m) {
  return m * m;
}
```

Fill in the following table about the symbol references in `main.c`. Pay close attention to the choices for each entry.

- *Entry* in `main.o .symtab`? ["yes" or "no"]
- *Type* ["external", "global", or "local"]
- *Module* where defined ["main.o" or "square.o"]
- *Section* where defined ["`.text`", "`.data`", or "`.bss`"]

If you answer that there is no entry in the `.symtab` segment of relocatable object file `main.o` for a certain symbol, fill in the remaining entries for that symbol with "n/a".

Symbol	*Entry*	*Type*	*Module*	*Section*
`main`				
`n`				
`x`				
`square`				

8. **(8 points)** Fill in the body of `decode` with valid C code so that it is equivalent to the x86 assembly code below.

```
int decode(long* p1, long* p2, int a, short b){









}
```

```
decode:
  sub       %rdi, %rsi
  movswl    %cx, %ecx
  sar       $0x3, %rsi
  imul      %esi, %ecx
  lea       (%rcx, %rdx, 1), %eax
  retq
```

9. **(8 points)** Consider the C program below. For space reasons, we are not checking error return codes. Assume that all functions return normally, that each printed string is flushed to stdout immediately, and that the proper header files have been included.

   Process/thread graphs must be shown to get any credit.

```
int main() {
   int status;
   int x;

   x = !!fork();
   printf("start\n");

   if(!x) printf("%d\n", x);

   if(wait(&status) != -1)
      printf("%d\n", WEXITSTATUS(status));

   if(x) printf("end\n");
   exit(3);
}
```

   Recall the following:

   - Function `wait` returns `-1` when there is an error; e.g., when there is no child.
   - Macro `WEXITSTATUS` extracts the exit status of the terminating process.

   Enumerate all possible outputs of this program.

10. **(14 points)** Consider the cache behavior of the following fragment of C code:

```
typedef struct {
  short s[8];
} pair;
pair arr[100][100];

int sum(int limit) {
  int i, j, k;
  long result = 0;

  for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
      for (k = 0; k < limit; k++)
        result += arr[i][j].s[k];

  return result;
}
```

Assume a 1024-byte direct-mapped cache that uses 4-byte blocks. Also, assume that the cache is initially empty, local variables are in registers, and the starting address of `arr` is 0x0.

(a) Suppose that we call `sum(4)`. For each of first five memory accesses to `arr`, what is the accessed element, the accessed address, and is the access a cache hit or miss? (The first row is given as an example. You may give addresses in hex or decimal.)

Access expression	**Access address**	**Hit or miss?**
`arr[0][0].s[0]`	`0`	miss

What is the overall cache miss rate when we call `sum(4)`?

(b) Suppose that we call `sum(2)`. For each of first five memory accesses to `arr`, what is the accessed element, the accessed address, and is the access a cache hit or miss? (The first row is given as an example. You may give addresses in hex or decimal.)

Access expression	**Access address**	**Hit or miss?**
`arr[0][0].s[0]`	`0`	miss

What is the overall cache miss rate when we call `sum(2)`?

(c) Suppose that we call `sum(1)`. For each of first five memory accesses to `arr`, what is the accessed element, the accessed address, and is the access a cache hit or miss? (The first row is given as an example. You may give addresses in hex or decimal.)

Access expression	**Access address**	**Hit or miss?**
`arr[0][0].s[0]`	`0`	miss

What is the overall cache miss rate when we call `sum(1)`?