Andrew ID:
Full Name:

Hint: This is an old school handwritten exam. There is no authenticated login. If we can't read your AndrewID, we won't easily know who should get credit for this exam. If we can't read either your AndrewID or Full Name, we're in real bind. Please write neatly :-)

18-213/18-613, Spring 2023 Final Exam

Monday, May 1, 2023

Instructions:

- Make sure that your exam is not missing any sheets (check page numbers at bottom)
- Write your Andrew ID and full name on this page (and we suggest on each and every page)
- This exam is closed book and closed notes (except for 2 double-sided note sheets).
- You may not use any electronic devices or anything other than what we provide, your notes sheets, and writing implements, such as pens and pencils.
- Write your answers in the space provided for the problem.
- If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The point value of each problem is indicated.
- **Good luck!**

Problem #	Scope	Max Points	Score
1	Data Representation: "Simple" Scalars: Ints and Floats	10	
2	Data Representation: Arrays, Structs, Unions, and Alignment	10	
3	Assembly, Stack Discipline, Calling Convention, and x86-64 ISA	15	
4	Caching, Locality, Memory Hierarchy, Effective Access Time	15	
5	Malloc(), Free(), and User-Level Memory Allocation	10	
6	Virtual Memory, Paging, and the TLB	15	
7	Process Representation and Lifecycle + Signals and Files	10	
8	Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW	15	
TOTAL	***Total points across all problems***	***100***	

Question 1: Representation: "Simple" Scalars (10 points)

Part A: Integers (5 points, 1 point per blank)

Assume we are running code on a machine using two's complement arithmetic for signed integers:

- 7-bit integers
- 2s complement signed representation

Fill in the five empty boxes in the table below when possible and indicate "UNABLE" when impossible. An "Everyday" number or expression has the value it would be understood to have in middle school arithmetic. A "C expression" has the value it would have if evaluated in a C Language program.

Goal	Machine 1: 7-bit w/2s complement signed	True or False
"Everyday number" -67		
"Everyday number" -9		
"C Expression" (-64 - 3)		
C Expression: (-7 > 11U)		
Tmax (Most positive number)		

Part B: Floats (5 points, 1 point per blank)

For this problem, please consider a floating point number representation based upon an IEEE-like floating point format as described below.

- Format:
  - There are 8 bits
  - There is 1 sign bit s.
  - There are n = 4 exponent bits.

Fill in the empty (non grayed-out) boxes as instructed.

	Answer
Total Number of Bits (Decimal)	**8**
Number of Sign Bits (Decimal)	**1**
Number of Exponent Bits (Decimal)	**4**
Number of Fraction Bits (Decimal)	
Bias (Decimal)	
The absolute difference, in decimal or as a power of 2, between any two adjacent denormalized numbers	
1100 1010 (Decimal value, unrounded)	
Bit representation of value below, or closest possible. *Hint:* Round even. **9/1024**	

Question 2: Representation: Arrays, Structs, Unions, Alignment, etc. (10 points)

Please consider a "Shark" machine for all parts of this question: 1-byte chars, 2-byte shorts, 4-byte ints, 8-byte longs, 8 byte doubles, 4 byte floats, and 8 byte doubles

Part A (2 points): Consider the following struct. How much memory is required? Answer in bytes. Please write only the number, without units, in the provided box:

```
struct {
  float f;
  double d;
  short s;
  char c;
} examStruct1;
```

Part B (2 points): In the box below, please rewrite the struct to require as little memory as possible:

Part C (2 point): How much memory, in bytes, is saved by the reorganization of the struct? Please write only the number, without units, in the provided box:

Continued on next page.

Part D (2 points): Consider the following array. How far apart are array[3][2] and array[2][1]? Answer in bytes.(Use your optimized version of examStruct1 from PartB)

```
struct examStruct1 array[4][3];
```

Part E (2 points): Considering the original, unoptimized struct and the following definition, what is the offset in bytes of `es1p->c` within the referenced struct. Please write only the number, without units, in the provided box:

Question 3: Assembly, Stack Discipline, Calling Convention, and x86-64 (15 points)

Assume that all subparts pertain to the “Shark Machine” environment.

Part A: Calling Convention (4 points)

Consider the following code which adds numbers.

```
(gdb) disassemble fn
Dump of assembler code for function fn:
   0x00000000004005d0 <+0>:     add    0x200a79(%rip),%rdi
   0x00000000004005d7 <+7>:     add    0x200a6a(%rip),%rdi
   0x00000000004005de <+14>:    add    %rsi,%rdi
   0x00000000004005e1 <+17>:    add    %rdi,%rdx
   0x00000000004005e4 <+20>:    add    %rdx,%rcx
   0x00000000004005e7 <+23>:    add    %rcx,%r8
   0x00000000004005ea <+26>:    lea    (%r8,%r9,1),%rax
   0x00000000004005ee <+30>:    add    0x8(%rsp),%rax
   0x00000000004005f3 <+35>:    add    0x10(%rsp),%rax
   0x00000000004005f8 <+40>:    add    0x18(%rsp),%rax
   0x00000000004005fd <+45>:    add    0x20(%rsp),%rax
   0x0000000000400602 <+50>:    retq
End of assembler dump.
```

Continued on next page.

3(A)(1) (1 points): How many numbers are being added? How do you know? Please answer in the box below.

3(A)(2) (1 points): How many of the numbers being added are global variables? How do you know? Please answer in the box below.

3(A)(3) (1 points): How many of the numbers being added are local variables? How do you know? Please answer in the box below.

.

3(A)(4) (1 points): How many bytes on the stack are being used to store local variables? How do you know? Please answer in the box below.

Continued on next page.

Question 3: Assembly, Stack Discipline, Calling Convention, and x86-64, *cont.* (15 points)

Part B: Conditionals and Loops (5 points)

Consider the following code:

```
Dump of assembler code for function loop:
   0x0000000000001169 <+0>:     endbr64
   0x000000000000116d <+4>:     push   %rbp
   0x000000000000116e <+5>:     mov    %rsp,%rbp
   0x0000000000001171 <+8>:     sub    $0x20,%rsp
   0x0000000000001175 <+12>:    mov    %edi,-0x14(%rbp)
   0x0000000000001178 <+15>:    mov    %esi,-0x18(%rbp)
   0x000000000000117b <+18>:    mov    %edx,-0x1c(%rbp)
   0x000000000000117e <+21>:    mov    %ecx,-0x20(%rbp)
   0x0000000000001181 <+24>:    movl   $0xffffffff,-0x8(%rbp)
   0x0000000000001188 <+31>:    mov    -0x18(%rbp),%eax
   0x000000000000118b <+34>:    mov    %eax,%edx
   0x000000000000118d <+36>:    neg    %edx
   0x000000000000118f <+38>:    cmovns %edx,%eax
   0x0000000000001192 <+41>:    mov    %eax,-0x18(%rbp)
   0x0000000000001195 <+44>:    mov    -0x20(%rbp),%eax
   0x0000000000001198 <+47>:    mov    %eax,%edx
   0x000000000000119a <+49>:    neg    %edx
   0x000000000000119c <+51>:    cmovns %edx,%eax
   0x000000000000119f <+54>:    mov    %eax,-0x20(%rbp)
   0x00000000000011a2 <+57>:    mov    -0x14(%rbp),%eax
   0x00000000000011a5 <+60>:    mov    %eax,-0x10(%rbp)
   0x00000000000011a8 <+63>:    jmp    0x11f4 <loop+139>
   0x00000000000011aa <+65>:    mov    -0x1c(%rbp),%eax
   0x00000000000011ad <+68>:    mov    %eax,-0xc(%rbp)
   0x00000000000011b0 <+71>:    jmp    0x11e8 <loop+127>
   0x00000000000011b2 <+73>:    mov    -0x10(%rbp),%eax
   0x00000000000011b5 <+76>:    imul   -0xc(%rbp),%eax
   0x00000000000011b9 <+80>:    mov    %eax,-0x4(%rbp)
   0x00000000000011bc <+83>:    cmpl   $0x0,-0x4(%rbp)
   0x00000000000011c0 <+87>:    je     0x11e3 <loop+122>
   0x00000000000011c2 <+89>:    mov    -0x4(%rbp),%ecx
   0x00000000000011c5 <+92>:    mov    -0xc(%rbp),%edx
   0x00000000000011c8 <+95>:    mov    -0x10(%rbp),%eax
   0x00000000000011cb <+98>:    mov    %eax,%esi
   0x00000000000011cd <+100>:   lea    0xe30(%rip),%rax        # 0x2004
   0x00000000000011d4 <+107>:   mov    %rax,%rdi
   0x00000000000011d7 <+110>:   mov    $0x0,%eax
   0x00000000000011dc <+115>:   call   0x1060 <printf@plt>
   0x00000000000011e1 <+120>:   jmp    0x11e4 <loop+123>
   0x00000000000011e3 <+122>:   nop
   0x00000000000011e4 <+123>:   addl   $0x1,-0xc(%rbp)
   0x00000000000011e8 <+127>:   mov    -0xc(%rbp),%eax
   0x00000000000011eb <+130>:   cmp    -0x20(%rbp),%eax
   0x00000000000011ee <+133>:   jl     0x11b2 <loop+73>
   0x00000000000011f0 <+135>:   addl   $0x1,-0x10(%rbp)
   0x00000000000011f4 <+139>:   mov    -0x10(%rbp),%eax
   0x00000000000011f7 <+142>:   cmp    -0x18(%rbp),%eax
   0x00000000000011fa <+145>:   jl     0x11aa <loop+65>
   0x00000000000011fc <+147>:   nop
   0x00000000000011fd <+148>:   nop
   0x00000000000011fe <+149>:   leave
   0x00000000000011ff <+150>:   ret
End of assembler dump.
```

***Hint*: Please be careful to understand the code. Answering these questions isn't as simple as counting forward or backward jumps.**

***Continued on next page*.**

3(B)(1) (2 points): How many loops are there? How do you know? If there are nested loops, count each separately. Please answer in the box below. Do not include units.

3(B)(2) (1 points): How many “if statements” are there? How do you know? Please answer in the box below. Do not include units.

3(B)(3) (1 points): How many ?-operators (ternary operators) are there? Explain your answer. Please respond in the box below:

3(B)(4) (1 points): Does any loop end other than by the condition tested in the loop’s predicate/test? How do you know? Please respond in the box below.

Continued on next page.

Part C: Switch statement (6 points)

Consider the following compiled from C Language code containing a switch statement and no if statements.

```
(gdb) disassemble foo
Dump of assembler code for function foo:
   0x0000000000400550 <+0>:     cmp    $0x6,%esi
   0x0000000000400553 <+3>:     ja     0x4005a0 <foo+80>
   0x0000000000400555 <+5>:     mov    %esi,%esi
   0x0000000000400557 <+7>:     jmpq   *0x400650(,%rsi,8)
   0x000000000040055e <+14>:    xchg   %ax,%ax
   0x0000000000400560 <+16>:    lea    0x0(,%rdi,8),%eax
   0x0000000000400567 <+23>:    sub    %edi,%eax
   0x0000000000400569 <+25>:    mov    %eax,%edi
   0x000000000040056b <+27>:    lea    0x2(%rdi),%edx
   0x000000000040056e <+30>:    mov    %edx,%eax
   0x0000000000400570 <+32>:    retq
   0x0000000000400571 <+33>:    nopl   0x0(%rax)
   0x0000000000400578 <+40>:    lea    0x9(%rdi),%edx
   0x000000000040057b <+43>:    mov    %edx,%eax
   0x000000000040057d <+45>:    retq
   0x000000000040057e <+46>:    xchg   %ax,%ax
   0x0000000000400580 <+48>:    mov    %edi,%edx
   0x0000000000400582 <+50>:    shr    $0x1f,%edx
   0x0000000000400585 <+53>:    add    %edi,%edx
   0x0000000000400587 <+55>:    sar    %edx
   0x0000000000400589 <+57>:    mov    %edx,%eax
   0x000000000040058b <+59>:    retq
   0x000000000040058c <+60>:    nopl   0x0(%rax)
   0x0000000000400590 <+64>:    lea    (%rdi,%rdi,2),%edx
   0x0000000000400593 <+67>:    mov    %edx,%eax
   0x0000000000400595 <+69>:    retq
   0x0000000000400596 <+70>:    nopw   %cs:0x0(%rax,%rax,1)
   0x00000000004005a0 <+80>:    mov    %edi,%eax
   0x00000000004005a2 <+82>:    mov    $0x55555556,%edx
   0x00000000004005a7 <+87>:    sar    $0x1f,%edi
   0x00000000004005aa <+90>:    imul   %edx
   0x00000000004005ac <+92>:    sub    %edi,%edx
   0x00000000004005ae <+94>:    mov    %edx,%eax
   0x00000000004005b0 <+96>:    retq
End of assembler dump.
```

Consider also the following memory dump, with the address obscured. Assume that it begins with the 0th entry of the switch statement's jump table.

```
(gdb) x/16gx 0xXXXXXX

        :       0x0000000000400590      0x0000000000400560
        :       0x000000000040056b      0x0000000000400578
        :       0x00000000004005a0      0x0000000000400580
        :       0x0000000000400580      0x0000003c3b031b01
        :       0x00000000004005a0      0x0000000000400580
        :       0x0000000000400590      0x0000000000400560
        :       0x0000000000400590      0x0000000000400560
        :       0x000000000040056b      0x0000000000400578
```

Continued on next page.

Part C: Switch statement, *cont.* (6 points)

(3)(C)(1) (2 point): At what address does the jump table shown above begin? How do you know? Please respond in the box below:

(3)(B)(3) (2 points): Is there a default case? If so, at what address does it begin? How do you know? Please respond in the box below:

(3)(C)(2) (2 points): Which case(s), if any, fall through to the next case **after executing some of their own code?** How do you know? Please respond in the box below:

Hint: Give the value for the case not the address.

Continued on next page.

Question 4: Caching, Locality, Memory Hierarchy, Effective Access Time (15 points)

Part A: Caching (9 points)

Given a model described as follows:

- Associativity: 2-way set associative
- Total size: 64 bytes (not counting meta data)
- Block size: 8 bytes/block
- Replacement policy: Set-wise LRU
- 8-bit addresses

4(A)(1) (1 point) How many bits for the block offset? Please answer, without units, in the box below:

4(A)(2) (1 point) How many bits for the set index? Please answer, without units, in the box below:

4(A)(3) (1 point) How many bits for the tag? Please answer, without units, in the box below:

Continued on next page.

4(A)(4) (6 points, ½ point per row): For each of the following addresses, please indicate by circling if it hits, or misses, and if it misses, the type of miss:

Address	**Circle one (per row):**		**Circle one (per row):**			
0xA0	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x25	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0xA6	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x0D	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x11	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x60	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x0F	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0xBA	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0xA5	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x42	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0x67	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A
0xC7	Hit	Miss	Capacity	Compulsory/Cold	Conflict	N/A

Part B: Locality (4 points)

Consider a 64B data cache that is 2-way associative and can hold four (4) 4-bytes ints in each line.

For the code below assume a cold cache, a cache-aligned array of 16 ints, and that all other variables are in registers.

The code is parameterized by positive integers m and n that satisfy m*n = 16 such that if you know either one, you know both.

```
int A[16], t = 0;
for(int i = 0; i < m; i++)
   for(int j = 0; j < n; j++)
      t += A[j*m + i];
```

4(B)(1) (1 points) For m=1, what is the miss rate? Please answer in the box below with a number and the percent sign, e.g. 62%.

Continued on next page.

4(B)(2) (1 points) For m=2, what is the miss rate? Please answer in the box below with a number and the percent sign, e.g. 62%.

4(B)(3) (1 points) For m=4, what is the miss rate? Please answer in the box below with a number and the percent sign, e.g. 62%.

4(B)(4) (1 points) For m=8, what is the miss rate? Please answer in the box below with a number and the percent sign, e.g. 62%.

Part C: Memory Hierarchy and Effective Access Time (2 points)

Imagine a computer system as follows:

- 2-level memory hierarchy (L1 cache, Main memory)
- L1: 5% miss rate
- Main memory: 100nS access time, 0% miss rate
- The effective memory access time is 10nS
- Memory accesses at different levels of the hierarchy **do not** overlap

FOR SIMPLICITY, AVOID COMPLEX CALCULATION AND LEAVE YOUR ANSWER AS A SIMPLE FRACTION

What is the access time for the L1 cache? Please answer, in nS, without units, in the box below:

Continued on next page.

Question 5: Malloc(), Free(), and User-Level Memory Allocation (10 points)

Considering a malloc implementation as described below:

- Explicit list, ordered smallest-to-largest, allocation via first-fit (*not* next-fit)
- Headers of size 8 bytes, Footer size of 8-bytes, Allocated blocks have footers
- Blocks must be a multiple of 8-bytes (In order to keep payloads aligned to 8 bytes).
- Minimum block size of 48B
- If there is no unallocated block of a large enough size to service the request, sbrk is called to grow the heap enough to get a new block of the smallest size that can service the request.
- The heap is unallocated until it grows in response to the first malloc.
- Constant-time coalescing is employed.
- The heap never shrinks

5(A) (10 points, 1 point per line) Please complete the following table with the values *after* the requested operation completes. The following definitions may be helpful reminders:

- *Total Heap Size* is the number of bytes between the base of the heap and the brk point, i.e. top of the heap.
- *Aggregate Request Size* is the total number of bytes requested via malloc() and not yet free()d.
- *Allocated Internal Fragmentation*: The difference between the size of each allocated block and the size of the request for which it was allocated
- *Total Malloc Overhead* is the difference, in bytes, between the amount of heap space and the aggregate request size

	Operation	Total Heap Size	Aggregate Request Size	Allocated Internal Fragmentation	Total Malloc Overhead
5(A)(1)(1 point)	`ptr1 = malloc (40);`				
5(A)(2)(1 point)	`ptr2 = malloc (40);`				
5(A)(3)(1 point)	`free(ptr1);`				
5(A)(4)(1 point)	`free (ptr2);`				
5(A)(5)(1 point)	`ptr1 = malloc(96);`				
5(A)(6)(1 point)	`ptr2 = malloc(28);`				
5(A)(7)(1 point)	`ptr3 = malloc(20);`				
5(A)(8)(1 point)	`free (ptr1);`				
5(A)(9)(1 point)	`free (ptr3);`				
5(A)(10)(1 point)	`malloc(2);`				

Continued on next page.

6. Virtual Memory, Paging, and the TLB (15 points)

This problem concerns the way virtual addresses are translated into physical addresses. Imagine a system has the following parameters:

- Virtual addresses are 12 bits wide.
- Physical addresses are 10 bits wide.
- The page size is 32 bytes.
- The TLB is 2-way set associative with 4 total entries.
- The TLB may cache invalid entries
- TLB REPLACES THE ENTRY WITH THE LOWEST TAG (NOT LRU)
- A single level page table is used

Part A: Interpreting addresses (3 points)

6(A)(1)(1 points): Please label the diagram below showing which bit positions are interpreted as each of the PPO and PPN. Leave any unused entries blank.

Bit	9	8	7	6	5	4	3	2	1	0
PPN/ PPO										

6(A)(2)(1 points): Please label the diagram below showing which bit positions are interpreted as each of the TLBI and TLBT . Leave any unused entries blank.

Bit	11	10	9	8	7	6	5	4	3	2	1	0
TLBI/ TLBT												

6(A)(3)(1 points): How many entries exist within each page table? Answer with a simple number, without units, in the box below:

6(A)(4) (2 points): How many sets are in the TLB?Answer with a simple number, without units, in the box below:

Continued on next page.

Part B: Hits and Misses (12 points)

Shown below are the **initial** states of the TLB and **partial** page table.

TLB (X=INvalid, V=VALID, R=READ, W=WRITE):

Set	Tag	PPN	BITS	Scratch space for you
0	000001	1	V-RW	
0	000010	9	V-R	
1	000011	3	V-RW	
1	000100	2	V-R	

Page Table (X=INvalid, V=VALID, R=READ, W=WRITE):

Index/VPN	PPN	BITS	Scratch space for you
0	5	V-RW	
1	13	V-RW	
2	1	V-RW	
3	11	V-R	
4	9	V-R	
5	15	X-R	
6	27	V-RW	
7	3	V-RW	
8	16	V-RW	
9	2	V-R	
10	12	V-RW	
11	23	X-RW	

Continued on next page.

Part B: Hits and Misses, *cont.* (12 points, 2 points per line)

Consider the following memory access trace e.g. sequence of memory operations listed in order of execution, as shown in the first two columns (operation, virtual address). It begins with the TLB and page table in the state shown above.

Please complete the remaining columns

Operati on	**Virtual Address**	**TLB Hit or Miss?**	**Page Table Hit or Miss?**	**Page Fault? Yes or No?**	**PPN If Knowable**
Write	0x0E5	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	
Read	0x058	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	
Write	0x0D8	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	
Write	0x15B	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	
Write	0x17A	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	
Read	0x0C5	Hit Miss Not knowable	Hit Miss N/A	Yes No Not knowable	

Continued on next page.

Question 7: Process Representation and Lifecycle + Signals and Files (10 points)

Part A (3 points):

Please consider the following code:

```
void main(){
   printf ("A"); fflush(stdout);

   if (!fork()) {
     printf ("B"); fflush(stdout);

     if (fork()) {
        printf ("C"); fflush(stdout);
     } else {
        printf ("D"); fflush(stdout);
     }
   }
   printf ("E"); fflush(stdout);
}
```

7(A)(1) (1 points): How many times is "E" printed? If it can vary, give the range of possibilities. Answer with a simple number and without units in the box below:

7(A)(2) (1 points): Give one output string that has the correct output characters (and number of each character), but in an impossible order. Answer in the box below:

7(A)(3) (1 points): Why can't the output you provided in 7(A)(2) be produced? Specifically, what constraint(s) from the code does it violate? Answer in the box below:

Continued on next page.

Question 7: Process Representation and Lifecycle + Signals and Files, *cont.* (10 points)
Part B (4 points):

Please consider the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/wait.h>

int main(int argc, char* argv[]) {
 char buffer[7] = "abcdef";
 char buffer2[7];

  // Assume "file.txt" and "file2.txt" are initially non-existant or empty.
  int fd0 = open("file.txt", O_RDWR | O_CREAT, 0666);
  int fd1 = -1;

  write(fd0, buffer, 3);

  if (!fork()) {
    write(fd0, buffer+3, 3);

    fd1 = open("file.txt", O_RDWR | O_CREAT, 0666);

    write(fd1, "X", 1);

    // int dup2(int oldfd, int newfd); copies oldfd over newfd
    dup2 (fd0,fd1);

    write(fd0, "A", 1);
 } else {
    wait(NULL);
    write(fd0, "P", 1);
 }
return 0;
}
```

7(B)(1) (2 points): What is the content of the output file after this code completes? Please answer in the box below:

7(B)(2) (1 points): If the child process was just about to “return 0”, how many entries are there in the system-wide open file table related to this code (ignore stdin, stdout, stderr)? Assume open file table garbage collection is done only when the program terminates. Please answer with a simple number and without units in the box below:

Continued on next page.

7(B)(3) (1 points): If the child process was just about to “return 0”, how many file descriptors are open in the parent and child (ignore stdin, stdout, stderr)? Please answer with a simple number and without units in the box below:

Question 7: Process Representation and Lifecycle + Signals and Files, *cont.* (10 points)

Part C (4 points):

Consider the C code below. Assume that no errors prevent any processes from running to completion.

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_CMDLINE 1024
#define MAX_CHILDREN 100
#define MAX_ARGS ( (MAX_CMDLINE/2) + 1)

pid_t children[MAX_CHILDREN];

/*
 *  SIGCHLD handler
 */
void sigchld_handler(int signum) {
  int pid, index;

  /* Note: WNOHANG = Don't block, return if nothing waitable */
  if ( (pid=waitpid (0, NULL, WNOHANG)) <= 0) return;
  for (index=0; index < MAX_CHILDREN; index++) {
    if (children[index] == pid) {
      children[index] = 0;
      return;
    }
  }
}


/*
 * Parses a cmdline populating a provided argv array
 *
 * Returns the number of arguments in the argv[] array,
 * not counting argv[0]
 *
 *  argv[] is allocated by caller
 */

int getargs(char *cmdline, char *argv[]) {
  int argc = 0;
  char *argp;

  while (argp = strtok(cmdline, " ")) {
    argv[argc++] =  argp;
    cmdline = NULL;
  }
```

```
  if (!argc) return 0;
  return argc;
}


void main(int argc, char *argv[]) {
  char cmdline[MAX_CMDLINE];
  char *args[MAX_ARGS+1];

  int child_index;
  pid_t cpid;

  signal(SIGCHLD, sigchld_handler);


  for (child_index = 0; child_index < MAX_CHILDREN; child_index++) {
    memset (args, 0, (MAX_ARGS+1) * sizeof(char *));
    memset (cmdline, 0, MAX_CMDLINE*sizeof(char));

    printf ("Please enter the command line to execute or an empty line to stop.\n");
    printf ("cmdprompt> ");
    fflush(stdout);
    fgets (cmdline, 1024, stdin);
    *(strchr(cmdline, '\n')) = '\0';

    if (!strlen(cmdline)) break;
    if (!getargs(cmdline, args)) break;
    if (!(cpid = fork()) ) { /* Child process */

      execv(args[0], args);

      perror ("Exec failed :");
      continue;
    } /* if fork */

    /* parent */
    children[child_index] = cpid;

  } /* for child_index */

  printf ("The following children have not been reaped:\n");
  for (int child_index=0; child_index < MAX_CHILDREN; child_index++) {
    if (children[child_index] > 0) printf ("%d\n", children[child_index]);
  }
  fflush(stdout);

} /* main() */
```

7(C)(1)(1 points) Assume that three commands are exec'd. What is the *minimum* number of times SIGCHLD might be received by the handler before the program terminates? Please answer with a simple number and without units in the box below:

Continued on next page.

7(C)(2)(1 points) Assume that three commands are exec'd. What is the *maximum* number of times SIGCHLD might be received by the handler before the program terminates? Please answer with a simple number and without units in the box below:

7(C)(3)(2 points) If the SIGCHLD handler is correct for its intended purpose, write "Correct". Otherwise, correct it in the box below:

Continued on next page

Question #8: Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW (15 points)
Part A (5 points): Deadlock

Consider the following C code:

8(A)(2)	8(A)(3)	Code
		1. /* Initialize semaphores */
		2. mutex1 = 1;
		3. mutex2 = 1;
		4. mutex3 = 1;
		5. mutex4 = 1;
		6
		7. void thread1() {
		8. P(mutex2);
		9. P(mutex3);
		10. P(mutex4);
		11
		12. /* Access Data */
		13. V(mutex4);
		14. V(mutex2);
		15. V(mutex3);
		16. }
		17
		18. void thread2() {
		19. P(mutex1);
		20. P(mutex2);
		21. P(mutex3);
		22
		23. /* Access Data */
		24
		25. V(mutex1);
		26. V(mutex2);
		27. V(mutex3);
		28. }

8(A)(1) (2 points) Is it possible for the code above to deadlock? Circle one: Yes No

8(A)(2) (3 points) Consider your answer to (A) above. If you answered "No", explain why deadlock is impossible. If you answered "Yes", then please provide a schedule that results in deadlock. Do this by numbering, i.e. 1, 2, 3, etc, the semaphore operations (Ps and Vs, only) in the code above with an execution order that results in deadlock. Use the 8(A)(2) column to record your answer. Please answer in the box below:

Continued on next page.

Question #8: Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW, *cont.* (15 points)

Part B (7 points): Concurrency Control

This question asks you to synchronize **exactly two** threads, one of which prints “A” and the other of which prints “B” such that the output is ABBBABBBABBB…and so on. In other words, one A is printed, and then three Bs are printed, and then that pattern repeats indefinitely.

Please modify, as instructed by the comments in the code, the code provided below by declaring and initializing any semaphores or mutexes that are needed, and then using them within the PrintA() and PrintB() threads.

You may assume that the threads are created and joined elsewhere in the code. Please do not concern yourself with this code.

```
// Declare any needed shared, global variables here.
// Hint: Semaphores may be declared as sem_t, e.g. "sem_t someSemaphore;"
```

```
// Do any initialization here. This runs before any thread.
// Hint: Semaphores may be initialized with sem_init, e.g. sem_init(sem, num)
```

Continued on next page.

Question #8: Concurrency Control: Maladies, Semaphores, Mutexes, BB, RW, *cont.* (15 points)
Part B (7 points), *cont.*: Concurrency Control

```
void *PrintA(void *) {

   while (true){

      // Add code here, as needed






      write(STDOUT_FILENO, 'A', 1);

      // Add code here as needed






  }
}
```

```
void *PrintB(void *) {

    while (true){

       // Add code here, as needed






       write(STDOUT_FILENO, 'B', 1);

       // Add code here as needed






  }
}
```

Continued on next page.

Part C

8(C)(1) (1.5 points) The question above asked you to consider the case where there is exactly one instance of each of TheadA() and ThreadB(). This question asks you to consider the applicability of your solution to a different scenario: Assume that there can be **exactly one instance of ThreadA()**, at most, **how many instance of ThreadB() can exist** without breaking correctness? Why?

8(C)(2) (1.5 points) The question above asked you to consider the case where there is exactly one instance of each of TheadA() and ThreadB(). This question asks you to consider the applicability of your solution to a different scenario: Assume that there can be **exactly one instance of ThreadB()**, at most, **how many instance of ThreadA() can exist** without breaking correctness? Why?

The End (of the whole exam!)! You made it!