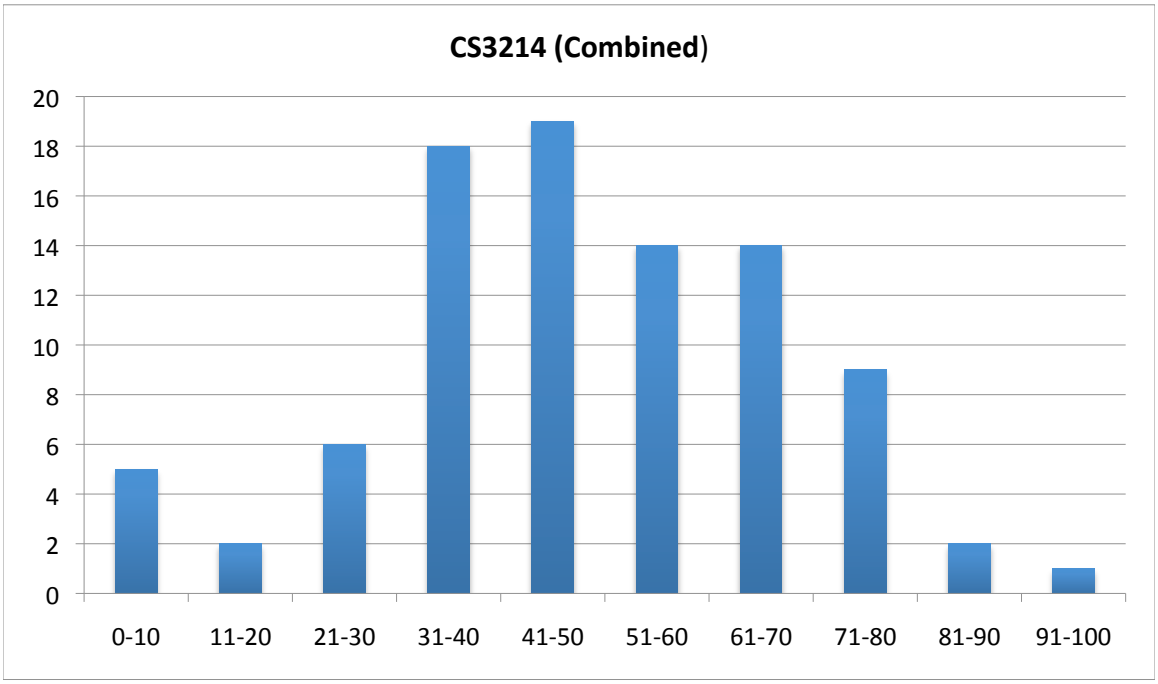
CS 3214 Midterm Solution

A total of 86 students took the midterm in the two sections. The table below shows a statistical summary of each problem and who graded the problem. If you have questions, contact the person who graded the respective problem first. Students who scored below 25 are at risk of failing the class even if they otherwise meet minimum requirements; they will need to show improvement taking the final exam. The histogram shows the distribution of scores across both the sections.

	1	2	3	4	Total
Median	6.5	15.0	13.5	16.5	47.5
Average	6.8	14.6	12.2	16.1	49.7
StDev	4.7	5.1	6.7	6.6	17.2
Min	0	3	1	4	9
Max	18	25	25	29	91
Possible	20	25	25	30	100
Grader(s)	Hari	(a)-(b) Ian (c) Pat	Ruslan	(a)-(e) Ali (f)-(g) Dennis	



Solutions are shown in this style.

Grading Comments are shown in this style.

1. Stack Organization (20 points)

The assembly language code in the IA32 architecture for a C function is shown below.

```
        movl    $0, -4(%ebp)
        jmp     L2
L3:
        addl    $1, -4(%ebp)
L2:
        movl    -4(%ebp), %eax
        addl    8(%ebp), %eax
        movzbl  (%eax), %eax
        testb   %al, %al
        jne     L3
        movl    12(%ebp), %eax
        movl    -4(%ebp), %edx
        leal    (%edx,%eax), %eax
        addl    16(%ebp), %eax
```

(a) (6 points) Deduce the return value and the argument(s) from this assembly language code and write the C declaration for the function. In the assembly language code the entry and exit code is not shown.

`int function(char* p1, int p2, int p3)`

Most points were given for correctly determining the number and type of arguments. Smaller credit was given for correctly determining the return type.

(b) (6 points) Suppose a function g is partially defined as follows:

```
void g(int x, double y) {

   int z[3];
   struct {
        int a;
        char b[8];
        double c;
   } s;
  ….
}
```

Ignoring other factors, show the assembly language entry code of this function for the IA32 architecture.

```
pushl %ebp          // save base pointer
movl  %sp,%ebp      // create new stack frame
subl  $0x20, %sp    // allocate space in new frame
```

About half of the points were given for correctly setting the new stack frame. The remaining points were given for analyzing the amount of space needed in the new stack frame and adjusting the stack pointer accordingly.

Suppose a function h is partially defined as follows:

```
char* h(int x) {

     int a[10];
        …
     }
```

(c) (3 points) Draw a diagram showing the runtime stack when function h is being executed. Your diagram must include the parameter x and the local variable a.

Credit was given for showing the return address and saved base pointer, the correct ordering of these elements, and for showing all the elements required.

(d) (3 points) What assembly language code in the IA32 architecture would be used so that the function returns the pointer to the next instruction to be executed in the caller when the function h returns?

```
movl   4(%ebp),%eax
```

Credit was given for using the correct register for the return value, for using the correct offset, and for using the base pointer register as a base address.

(e) (2 points) For the specific function h defined in part (c) show the C language statement that would achieve the same result (i.e., returns the pointer to the next instruction to be executed in the caller).

```
return a[11];
```

Credit was given for using out of bounds array indexing to reach the return address; partial credit was given for using negative indexing (implying a wrong idea about the layout of the array in memory); padding that might have been part of the answer for part (c) was not considered as wrong.

2. Linking and Loading (25 pts)

As shown below, the file f.c defines a function f whose code refers to the variable x that is not a local variable or a parameter. The file g.c defines a function g whose code refers to the variable x that is not a local variable or a parameter.

//file f.c
// declaration for x – column a below
 	
void f(int a){
  …
  x = …
}
	//file g.c
	// declaration for x – in column b below
	
	void g(char* p){
	  …
	  x = …
	}

(a) (9 points) The following table below shows how the variable x is defined in each file. For each row in the table indicate in column (c) whether the two functions at run-time refer to the SAME identifier (memory location) or to DIFFERENT identifiers (memory locations) or indicate in column (d) if there is a linking ERROR.

(a) declaration in f.c	(b) declaration in g.c	(c) SAME or DIFFERENT	(d) ERROR?
`int x;`	`extern int x;`	**SAME**	
`int x = 0;`	`extern int x;`	**SAME**	
`int x = 0;`	`int x = 0`		**X**
`static int x;`	`static int x`	**DIFFERENT**	
`static int x;`	`static double x;`	**DIFFERENT**	
`static int x;`	`extern int x;`		**X**
`int x = 0;`	`int x = 1;`		**X**
`int x;`	`double x;`	**SAME**	
`int x = 0;`	`double x = 0.0;`		**X**

Approximately one point of credit was given for each correct answer.

(b) (10 points) The next page shows C code and the corresponding object module. Determine what instructions in the .text segment and what data in the .data and .bss segments will need to be modified by the linker when the module is relocated. Draw a box around the specific bytes that will be modified. An example of this is shown in line 6 the object module listing. For each box,

describe in words what should appear in that box at run-time (e.g., the box shown would be described as “the address of x”). Write the descriptions at the end of the next page.

C code

```
extern int x;
extern int f1(int* yp );
int y = 0;
int* xp = &x;

int f2(void) {

  x = x + 1;
  f1(xp);
  return x + y;
}
```

Object File

```
Disassembly of section .text:

00000000 <_f2>:
   0:     55                          push   %ebp
   1:     89 e5                       mov    %esp,%ebp
   3:     83 ec 18                    sub    $0x18,%esp
   6:     83 05 00 00 00 00 01        addl   $0x1,0x0
   d:     a1 00 00 00 00  (1)         mov    0x0,%eax
  12:     89 04 24                    mov    %eax,(%esp)
  15:     e8 00 00 00 00  (2)         call   1a <_f2+0x1a>
  1a:     a1 00 00 00 00  (3)         mov    0x0,%eax
  1f:     03 05 00 00 00 00  (4)      add    0x0,%eax
  25:     c9                          leave
  26:     c3                          ret
  27:     90                          nop
Disassembly of section .data:

00000000 <_xp>:
   0:     00 00  (5)
     ...
Disassembly of section .bss:

00000000 <_y>:
   0:     00 00
     ...
```

(1) address of xp
(2) entry point of f1
(3) address of x
(4) address of y

(5) address of x

Equal credit was given for each correct answer. The entry in the .bss section was not graded.

(c) (6 points) There are three points in time when linking can be done. Briefly describe the form of linking that takes place at each of these three points. For each form of linking describe one advantage of using that form of linking.

Linking Time	Advantage
static	no run-time overhead
dynamic linking at load time	shared across processes/address spaces less disk/memory space used easier updates to libraries (no re-linking needed)
dynamic linking at run time	program controlled most flexible form of choosing libraries pluggable software architectures

Equal credit was given for each answer. Multiple advantages are shown above to indicate that there were a variety of correct answers, but only one advantage was required in each case.

3. Optimization and Locality (25 pts)

The code snippet shown below computes a measure of how clustered the data in an array is by squaring the difference between each data point and the average of the data. The data is in an array named a. The size of the array and the average of the data is computed by two utility functions, length(a) and average(a), respectively.

```
int fit = 0;

for(i=0; i<length(a); i++) {

   fit = fit + pow(a[i] - average(a), 2);

}
```

(a) (4 points) Name a machine-independent optimization that can be applied to the above code and show the resulting code.

Code hoisting – moving expensive, invariant operations out of the loop body.

```
int ln = length(a);
double av = average(a);

for(i=0; i<ln; i++) {
     fit = fit + pow(a[i] - av, 2);
}
```

Credit was assigned for correctly naming (or describing) a machine-independent optimization. Equal credit was assigned for showing correctly revised code.

(b)(8 points) Name a second optimization that can be applied to your answer in part (a) and show the resulting code.

Loop unrolling

```
for(i=0; i<ln; i=i+2) {
  fit = fit
       + pow(a[i] - av, 2)
       + pow(a[i+1] - av, 2);}
if (i == ln)
  fit = fit + pow(a[ln]-av,2)
```

Multiple accumulators

```
for(i=0; i<ln; i=i+2) {
 fit1 = fit1 + pow(a[i]-av,2);
 fit2 = fit2 + pow(a[i+1]-av,2);}
if (i == ln)
 fit1 = fit1 + pow(a[ln]-av,2);
fit = fit1 + fit2;
```

Two solutions are shown above but only one is required. Either (or an equivalent) method was acceptable for credit. Credit was given for correctly naming (or describing) the method. In the cases shown above, some credit was also assigned for correctly handling last case after loop termination.

For the following code:

```
int x[2][128];
int i;
int sum = 0;

for (i = 0; i < 128; i++) {
     sum += x[0][i] * x[1][i];
}
```

Assume this code is executed under the following conditions:

- sizeof(int) = 4.
- Array x begins at memory address 0x0 and is stored in row-major order.
- The cache is initially empty.
- The only memory accesses are to the entries of the array x. All other variables are stored in registers.

(c) (8 points) What is the miss rate for a cache of size 512 bytes that is direct mapped and has 16-byte cache blocks. A direct mapped cache is one in which there is only one line in each set. In this problem the single line contains a 16-byte block.

Miss rate is 64/256 = 25%.

Significant credit was given for a clear indication that the code will generate 1 miss for each group of 4 ints in row-major order. Additional credit was given for a clear indication that there will be 64 such blocks of 4 ints leading to 64 misses in total. Credit was also given for determining the miss rate.

(d) (2 points) What is the miss rate for the same cache as in (a) but of size 1024.

Same answer as in (c) – increase in cache size has no effect in this case.

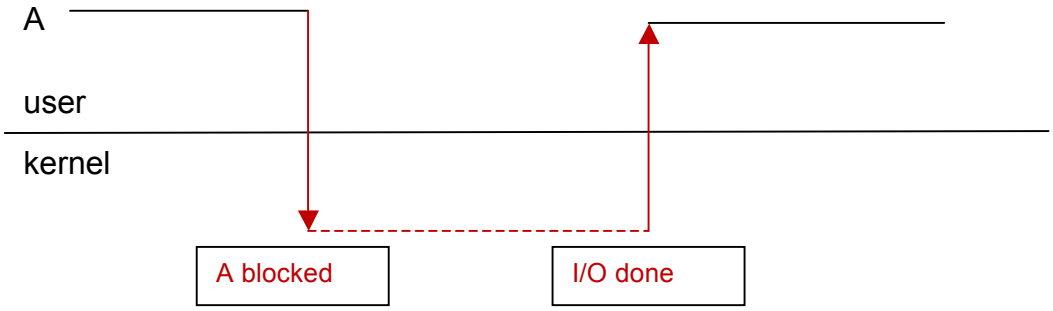
(e) (3 points) What is the miss for the same cache as in (a) but with 32-byte cache blocks.

Miss rate is 32/256 = 12.5%

Significant credit was given for a clear indication that the code will generate 1 miss for each group of 8 ints in row-major order. Additional credit was given for a clear indication that there will be 32 such blocks of 8 ints leading to 32 misses in total. Credit was also given for determining the miss rate.

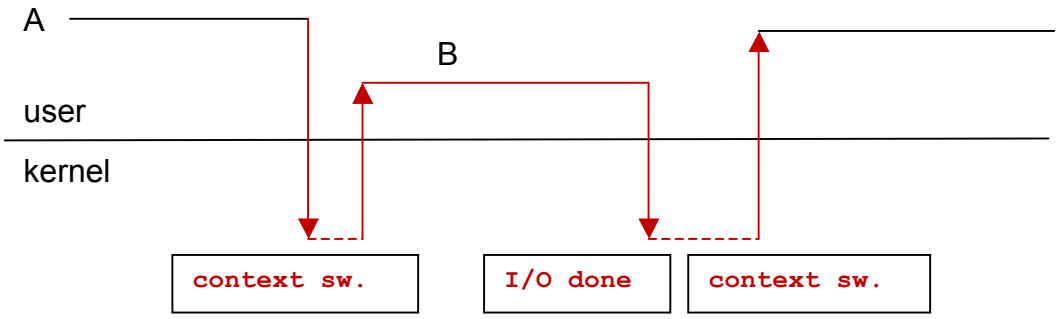
4. Processes and Job Control (30 pts)

(a) (3 points) Suppose there is only one process, A, executing in a system. At a given point A performs a blocking I/O operation. Complete the diagram below (like the one used in class) to show the user mode and kernel mode transitions. Time moves from left to right in the diagram.



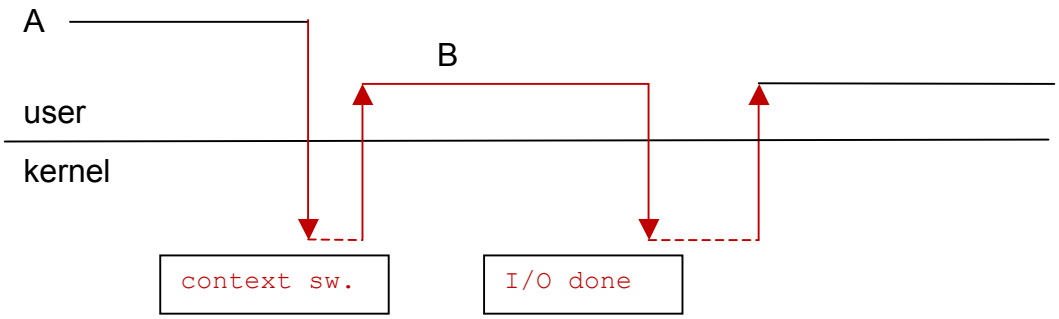
Credit was giving for showing the transition from user to kernel mode and some meaningful labeling of the added parts of the diagram.

(b) (2 points) Redraw the diagram from part (a) below to show one possible scenario for the case where a second process, B, was in the READY state when A performed its I/O operation.



Credit was given for showing the blocking of A, the context switch to B, and the resumption later (in this case) of A when the I/O is completed. Other valid scenarios were also given credit.

(c) (2 points) Redraw the diagram from part **(b)** to show another possible scenario.



Credit was given for showing the blocking of A, the context switch to B, and the resumption later (in this case) of B when the I/O is completed. Other valid scenarios were also given credit. The question mistakenly pointed to (a) instead of (b), however most students correctly interpreted the question. Credit was also given if students tried to show some scenario for (a).

(d) (4 points) Fill in the table below to name a system call that has the described effect. If no such system call exists write NONE.

always causes a context switch	**exit()**
always causes a mode switch	**read, write, (all)**
may not necessarily causes a context switch	**read, write (many others)**
may not necessarily causes a mode switch	**NONE**

Equal credit was given to each of the four answers. In the middle two cases many answers are possible. Any valid answer was given credit in these two cases.

(e) Suppose two processes are communicating by a Unix pipe. Describe under what circumstances the following situations occur.

(i) (3 points) The process performing a read from a pipe causes a context switch.

The pipe is empty. In this case the reading process is forced to wait (block) for data to become available in the pipe through a writing process.

Credit was given for correctly describing that the context switch was caused by the pipe having no available data to complete the read operation.

(ii) (3 points) The processes performing a write to a pipe causes a context switch.

The pipe is full. In this case the writing process is forced to wait (block) until a sufficient amount of the data has been removed from the pipe by a reading process.

Credit was given for correctly describing that the context switch was caused by the pipe having reached its capacity and, therefore, could not complete the write operation until a read operation was performed by another process to consume sufficient data.

In the code below, a parent process creates n processes running the same search program, each process searches a different file (i.e. the ith child process will search the ith file). For ease of job control the parent wants all of the child processes to be in the same process group. Only the critical parts of the code are shown. **This code is incorrect.**

```
main() {

int group = 0;

for(i=0; i<n; i++) {
   pid = fork();
   if (pid == 0) {      /* each child process … */
     pid = getpid();    /* get's its process pid */
     if (group==0) {    /* first child defines the group */
        group = pid;
        setpgrp(pid, group);
     }
     else setpgrp(pid, group);  /* other's use group id */

     execv("search", … );  /* child adopts search code */
   }
}
/* parent continues here after loop */
```

(f) (8 points) Explain briefly why this code is incorrect.

The code is incorrect because:

(1) each child process has a copy of the parent's address space in which the value of group is always zero (0). – 5 points

(2) Thus, each child will put itself in its own group rather than them being in the same process group – 3 points

Substantial credit was given for given the core reason for the code's incorrect operation: namely, the incorrect assumption about the value of group being communicated to the children. Credit was also given for recognizing the effect of this error on the creation of process groups. Because only critical parts of the code are shown, identification of missing declarations or initializations were not considered as acceptable errors.

(g) (5 points) Show a corrected version of the code that work properly (use the back of this page if necessary).

```
main() {

int group = 0;

//launch first child

pid = fork();
if (pid==0){
     pid = getpid();   /* first child defines the group */
     setpgrp(pid, pid);
     execv("search", … );  /* child adopts search code */
}

// parent continue here after launching first child

group = pid;  /* define group for other children  */

for(i=2; i<n; i++) {
   pid = fork();
   if (pid == 0) {         /* each other child process … */
     pid = getpid();       /* get's its process pid */
     setpgrp(pid, group); /* … and puts itself in group */
     execv("search", … ); /* child adopts search code */
   }
}

/* parent continues here after loop */
```

Credit was given for any of several alternative strategies for correctly putting the child processes in their own separate (from the parent process) group. Coding details were not considered in the grading.