CS 3214 Midterm

This is a closed-book, closed-internet, closed-cell phone and closed-computer exam. However, you may refer to your sheet of prepared notes. Your exam should have 9 pages with 4 topics totaling 100 points. You have 75 minutes. Please write your answers in the space provided on the exam paper. If you unstaple your exam, please put your initials on all pages. You may use the back of pages if necessary, but please indicate if you do so we know where to look for your solution. In three places you are asked to write your answers on the back of the pages. You may ask us for additional pages of scratch paper. You must submit all sheets you use with your exam. However, we will not grade what you scribble on your scratch paper unless you indicate you want us to do so. Answers will be graded on correctness and clarity. The space in which to write answers to the questions is kept purposefully tight, requiring you to be concise. You will lose points if your solution is more complicated than necessary or if you provide extraneous, but incorrect information along with a correct solution.

Name (printed) _____

I accept the letter and the spirit of the Virginia Tech undergraduate honor code – I will not give and have not received aid on this exam.

(signed) _____

#	Problem	Points	Score
1	Program Representation	20	
2	Linking and Loading	25	
3	Processes and Job Control	35	
4	Memory Management	20	
	Total	100	

1. Program Representation (20 points)

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

```
pushl   %ebp                //line 1
movl    %esp, %ebp          //line 2
subl    $16, %esp           //line 3
movl    8(%ebp), %eax       //line 4
movl    %eax, -8(%ebp)      //line 5
movl    12(%ebp), %eax      //line 6
movl    %eax, -4(%ebp)      //line 7
movl    -4(%ebp), %eax      //line 8
movl    -8(%ebp), %edx      //line 9
leal    (%edx,%eax), %eax   //line 10
movl    %eax, s             //line 11
movl    z, %eax             //line 12
movl    %eax, 16(%ebp)      //line 13
movl    16(%ebp), %eax      //line 14
leave                       //line 15
ret                         //line 16
```

(a) (7 points) Fill in the table below to indicate how many parameters, local variables, and global variables are accessed in this code and which lines of the code reference each type.

Type	Number	Lines where referenced
Parameters	**3**	**4,6,13,14**
Local variables	**2**	**5,7,8,9**
Global variables	**2**	**11,12**

Shown below are two versions of the assembly code produced by gcc for function “b(int)”. The left column is the result when compiling at the first level of optimization (-O1). The right column is the result when compiling at the second level of optimization (-O2).

IA32 code compiled with –O1

```
b:
      pushl %ebp
      movl  %esp, %ebp
      subl  $24, %esp
      movl  8(%ebp), %ebx
      movl  $0, %eax
      testl %ebx, %ebx
      je    .L3
      movb  $1, %al
      cmpl  $1, %ebx
      je    .L3
      leal  -1(%ebx), %eax
      movl  %eax, (%esp)
      call  b
      movl  %eax, %esi
      subl  $2, %ebx
      movl  %ebx, (%esp)
      call  b
      addl  %esi, %eax
.L3:
      movl  %ebp, %esp
      popl  %ebp
      ret
```

IA32 code compiled with –O2

```
b:
      pushl %ebp
      movl  %esp, %ebp
      xorl  %esi, %esi
      subl  $16, %esp
      movl  8(%ebp), %ebx
      testl %ebx, %ebx
      je    .L3
      cmpl  $1, %ebx
      jne   .L8
      jmp   .L13
.L6:
      cmpl  $1, %ebx
      je    .L12
.L8:
      leal  -1(%ebx), %eax
      movl  %eax, (%esp)
      call  b
      addl  %eax, %esi
      subl  $2, %ebx
      jne   .L6
.L3:
      addl  $16, %esp
      movl  %esi, %eax
      popl  %ebp
      ret
.L12:
      addl  $1, %esi
      addl  $16, %esp
      movl  %esi, %eax
      popl  %ebp
      ret
.L13:
      movw  $1, %si
      jmp   .L3
```

(b) (8 points) Provide a C version of function b(int). [Hint: it is a well known mathematical algorithm.] **Write your answer on the back of this page**.

```
int b(int n) {
          if (n==0) return 0;
          if (n==1) return 1;
          return ( b(n-1) + b(n-2)  )
        }
```

(c) (5 points) Describe the optimization that the complier applied in the –O2 case that it did not apply in the –O1 case. **Write your answer on the back of this page.**

The recursive call in b has been replaced by iteration

2. Linking and Loading (25 pts)

Two .c files shown below are compiled to produce an executable file that is loaded into memory and ready to run. Use these files to answer the questions below.

```
extern int x;
int *p;
static int f=0;

main(){
  int f,g;

  p = malloc(100);

  f = 6;      //line 6
  sub(f,g);   //line 7
  exit();     //line 8
}
```

```
int x;
int y = 0;
extern int *p;

void sub(int a, int b){
  int c;

  c = a;     //line 1
  a = x;     //line 2
  b = y;     //line 3
  x = y;     //line 4
  *p = b;    //line 5
}
```

(a) (15 points) The following table references the code above using line numbers shown as comments. In the assignment statement in line 1, the variable “a” is the source and variable “c” is the target. For each line that is an assignment statement identify in column (b) the region in the executable file where the SOURCE value is located, identify in column (c) the region in the executable file where the TARGET value is located, and show in column (d) whether the source region is above (at a higher address) or below (at a lower address) the target region. Write UNKNOWN if the above/below ordering is not known. For lines 8 and 9 the source is the region containing the program counter address immediately before the call and the target is the region containing the program counter address immediately after the call. Use correct ELF names wherever possible to name the regions.

(a) CODE	(b) SOURCE	(c) TARGET	(d) SOURCE ABOVE/BELOW TARGET
line 1	stack	stack	above
line 2	bss	stack	below
line 3	data	stack	below
line 4	data	bss	above
line 5	stack	heap	above
line 6	code	data	above
line 7	code	code	unknown
line 8	code	kernel	below

(b) (6 points) The table below refers to symbols in the file above that contains the code for the sub function. Fill in each entry in the table with linker information. For each symbol definition. Write “NO” if the symbol is not one known to the linker or if it is not a definition.

Symbol	Weak/Strong	Local/Global
x	weak	global
y	strong	global
sub	strong	global
p	no	no
a	no	no
c	no	no

(c) (4 points) The table below refers to symbols in the file above that contains the definition of the main function. Fill in each entry in the table with linker information. In some cases you are asked to show whether a symbol is strong/weak and local/global. In other cases (where the symbol name is not given) you must identify a symbol with the given properties. Write “NO” if the symbol is not one known to the linker

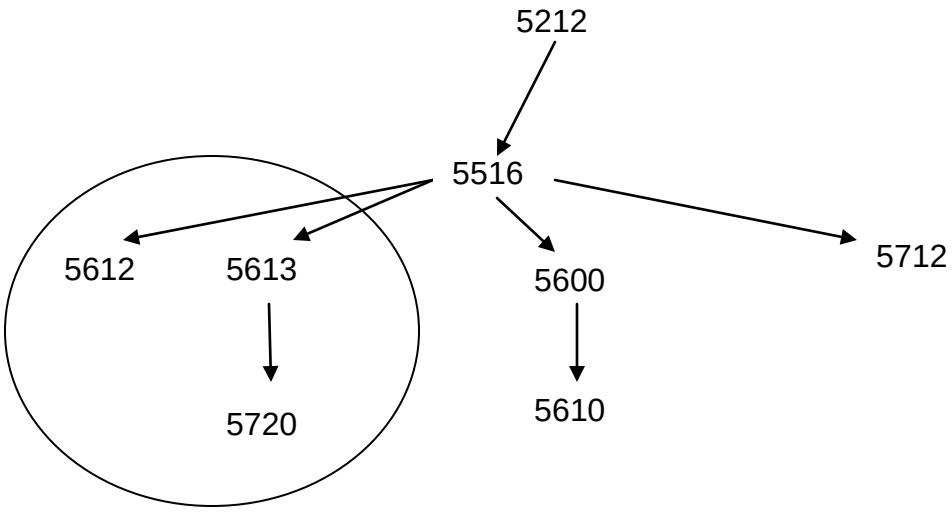
Symbol	Weak/Strong	Local/Global
x	no	no
p	weak	global
main	strong	global
f	strong	local

3. Processes and Job Control (35 pts)

The table below gives information about a collection of processes. Use this information to answer the questions below.

PID	Parent PID	Group PID
5516	5212	5516
5612	5516	5612
5613	5516	5612
5600	5516	5600
5610	5600	5610
5712	5516	5712
5720	5613	5612

(a) (5 points) Draw a tree diagram showing the parent-child relationships labeling each node with its PID. Draw a circle around those processes which could share control of the terminal at one time.



(b) (5 points) Suppose that the process whose process id is 5712 has just output a line to the terminal and is blocked waiting for the user to provide input. At this point the user enters a <cntrl>-C. For each signal that is sent give the name of the signal and what is the process id of the process that receives this signal.

Signal Name	PID of receiver
SIGINT	5712
SIGCHILD	5516

(c) (5 points) Suppose that the process whose process id is 5613 has just output a line to the terminal and is blocked waiting for the user to provide input. At this point the user enters a <cntrl>-C. For each signal that is sent give the name of the signal and what is the process id of the process that receives this signal.

Signal Name	PID of receiver
SIGINT	5613
SIGINT	5612
SIGINT	5720
SIGCHILD	5613
SIGCHILD	5516
SIGCHILD	5516

You have been assigned the job of writing a function that can be used to launch a new process running a given executable and return two file descriptors allowing the parent to write to and read from a pipe that is connected to the child process. The child process communicates with the parent through the pipe using its STDIN and STDOUT file descriptors. The signature for this function is:

```
int  run_piped(char* ex, int write_fd, int read_fd);
```

(d) (10 points) Identify the five most important system calls that you would need and very briefly state for each what role it would play.

1. **pipe – to create the pipes for communication**
2. **fork – to create a child process**
3. **exec – to load/execute the child code**
4. **dup2 – to rearrange the child's file descriptors**
5. **close – to close unneeded file descriptors in the parent and child**

NOTE: IT IS RECOMMENDED THAT DEFER ANSWERING THIS QUESTION UNTIL AFTER YOU HAVE FINISHED WITH OTHER PARTS OF THE EXAM.

(e) (10 points). Write below the code for the `run_piped` function.

```
int  run_piped(char* ex, int *write_fd, int *read_fd){

  int to_child[2], from_child[2]; // create pipes
  pipe(to_child);
  pipe(from_child);

  if (fork()==0) {  //child code

     dup2(to_child[1], STDIN);   // set up pipes for child
     dup2(from_child[0], STDOUT);
     close(to_child[0];
     close(from_child[1];

     exec(ex);                   // load child code
  }
  //parent code

  *read_fd  = from_child[0];  // return pipe ends to parent
  *write_fd = to_child[1];
  close[from_child[1];       // close unneeded pipe ends
  close(to_child[0]);

}
```

4. Memory Management (20 pts)

(a)(5 points) For a memory allocator using implicit lists, write an algorithm that determines the size of the largest payload in the free memory. Assume that `void* pstart` and void* pend are pointers to the beginning and the end of the heap memory.

```
int max = 0;
p = pstart;               // from start of heap
while (p<pend) {          // while more blocks
  if (p&01)               // check use bit
    p=p + (*p & ~01)      //go to next block if used
  else {                  // for free block
    if((*P & ~01) > max) // biggest seen so far
    max = (*p & ~01);     // set new maximum
  }
}
```

(b) (5 points) Describe two types of serious errors that application developers can make when performing explicit memory management.

1. **neglecting to free allocated memory – leads to memory leaks**
2. **freeing memory that is still in use – leads to program errors**

A dynamic memory allocator uses explicit lists with a LIFO ordered free list and a first-fit allocation policy. A particular application makes requests in three phases as follows:

phase 1: make a large number of pairs of request – one for a block of size 50 and for a block of size 100
phase 2: free all the blocks of size 50
phase 3: allocate a large number of blocks for size 200

(c) (5 points) During phase 3 the application experiences poor throughput performance from the dynamic memory allocator. Explain why.

Because the beginning of the free list contains a large number of blocks that are too small to satisfy the requested allocations in phase 3. Since this is a LIFO list the small blocks released in phase 2 will all be at the front of the list. First-fit starts at the beginning of the list. Thus, a large number of iterations is needed on each allocation to find a suitable block.

(d) (5 points) What are two ways in which the dynamic memory allocator could be changed to achieve better throughput performance for the above application?

1. use segregated storage
2. use next fit algorithm

[other valid answers also acceptable]