15-213 Introduction to Computer Systems

Exam 2

April 10, 2007

Name: _____

Andrew User ID: _____

Recitation Section: _____

- This is an open-book exam.
- Notes and calculators are permitted, but not computers.
- Write your answer legibly in the space provided.
- You have 80 minutes for this exam.
- We will drop your lowest score among questions 1–6.

	Problem	Max	Score
Symbols and Linking	1	**15**	
Virtual Address Translation	2	**15**	
Process Control	3	**15**	
Signals	4	**15**	
Garbage Collection	5	**15**	
Cyclone	6	**15**	
	Total	**75**	

1. Symbols and Linking (15 points)

In this problem we consider issues of symbols and linking. Consider the following three files.

File `polygon.h`:

```
struct Node {
  float pos[2];
  int marked;                    /* only for GC */
  struct Node* next;
  struct Node* prev;
};
typedef struct Node node;

node* alloc();                   /* allocated new node */
void init();                     /* initialize store */
void gc();                       /* call garbage collector */
```

File `main.c` (with portions of the function elided)

```
#include "polygon.h"
node* root_ptr;                 /* root pointer, for GC */

int main () {
  node* p;
  init();
  p = alloc();
  root_ptr = p;                 /* GC root is first allocated pointer */
  ...
  gc();
  ...
  return 0;
}
```

File `gc.c` (with function bodies elided)

```
#include "polygon.h"
#define N (1<<20)
static node polygon[N];          /* polygon array */
static node* free_ptr;           /* free list */
static node* root_ptr;           /* root pointer, for GC */

void mark(node* v) {...}
void sweep () {...}
void gc () {...}
void init () {...}
node* alloc () {...}
```

Possible definition of the functions in `gc.c` are revisited in Problems 5 and 6 although this is not relevant here.

Recall that a line `#include "`*file*`"` will literally be replaced by the contents of "*file*" by the C preprocessor. We compile the files to obtain an executable object file `polygon` with `gcc -o polygon main.c gc.c`. Questions related to symbols and linking therefore refer to the *expanded* versions of the files.

1. (9 points) Fill in the following tables by stating for each name whether it is local or global, and whether it is strong or weak. Cross out any box in the table that does not apply. For example, cross out the first box in a line if the symbol is not in the symbol table, or cross out the second box in a line if the symbol is not global (and therefore neither weak nor strong).

   `main.c`

	Local or Global?	Strong or Weak?
`root_ptr`		
`init`		
`main`		

   `gc.c`

	Local or Global?	Strong or Weak?
`N`		
`polygon`		
`root_ptr`		
`alloc`		

2. (3 points) Explain why linking succeeds to create an executable despite the fact that some symbols are declared in both files.

3. (3 points) Explain why the executable fails to be correct despite the fact that it compiles and links without warning. Propose how to fix the bug. Be clear in your suggestion correction (for example, *replace the line . . . with . . .*).

2. Virtual Address Translation (15 points)

In this problem we consider virtual address translation using a **two-level page table**. The parameters of the machine are as follows:

- The memory is byte addressable.
- Virtual addresses are 32 bits wide.
- The 32 bits of the virtual address are divided into 8 bits for VPN_1, 8 bits for VPN_2, and 16 bits for the VPO.
- Physical addresses are 24 bits wide.

1. (3 pts) Given the parameters above, fill in the blanks below.

   (i) The page size is _____.

   (ii) The maximal physical memory size is _____.

   (iii) Assuming a page table entry is 32 bits, the size of a page table is _____.

Next we consider the behavior of address translation using the two-level page table. We assume that page table entries have the following form, where unspecified bits are irrelevant for this problem:

- Level 1:
  - Bits 31–16: High 16 bits of level 2 page table physical base address
  - Bit 0: 1 = Present in physical memory
- Level 2:
  - Bits 31–24: PPN
  - Bit 0: 1 = Present in physical memory

For the following questions, assume that the VPN is not cached in the TLB, so we have to consult the page tables. The PTBR is set at `0xC80000`.

Address	Content
0xC80000	0xCA1C0001
0xC80004	0xCA1B0000
0xC80008	0xCA1D0001
0xC8000C	0xCA1B0001
0xC80010	0x00000000
0xC80014	0xCA1B0001
0xC80018	0xC8000000
0xC8001C	0xCA1B0001

Address	Content
0xCA1B00	0x07000001
0xCA1B04	0xFF000000
0xCA1B08	0x08000001
0xCA1B0C	0xCA000001
0xCA1B10	0x01000000
0xCA1B14	0x00000000
0xCA1B18	0x09000001
0xCA1B1C	0xCA000000

2. (6 points) For the virtual address `0x0300F218`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the answer and all subsequent results. All answers should be given in hexadecimal.

Parameter	Value
VPN_1	
VPN_2	
PTE (level 1)	
Page Fault? (Y/N)	
PTE (level 2)	
Page Fault? (Y/N)	
PPN	
Physical Address	

3. (6 points) For the virtual address `0x0507EE00`, indicate the physical address and various results of the translation. If there is a page fault, enter "—" for the PPN and Physical Address. All answers should be given in hexadecimal.

Parameter	Value
VPN_1	
VPN_2	
PTE (level 1)	
Page Fault? (Y/N)	
PTE (level 2)	
Page Fault? (Y/N)	
PPN	
Physical Address	

3. Process Control (15 points)

Consider the following C program. For space reasons, we do not check return codes, so assume that all functions return normally. Also assume that `printf` is unbuffered, and that each process successfully runs to completion.

```
int main() {
  int pid;

  pid = fork() && fork();
  if (!pid)
    printf("A\n");
  else
    printf("B\n");
  exit(0);
}
```

Mark each column that represents a valid possible output of this program with 'Yes' and each column which is impossible with 'No'.

A	A	A	A	A
A	A	A	A	A
	A	B	B	A
			B	B

4. Signals (15 points)

Consider the following code.

```
int val = 1;

void handler(int sig) {
  waitpid(-1, NULL, 0);
  val++;
}

int main() {
  int pid;
  signal(SIGCHLD, handler);
  pid = fork();
  if (pid == 0) {
    val = 0;
    exit(0);
  }
  printf("%d\n", val);
  exit(0);
}
```

Assume that all system calls succeed, and that all processes terminate normally. List all possible outputs of this program.

5. Garbage Collection (15 points)

In this problem we consider code for a specialized mark-and-sweep garbage collector, a skeleton of which was introduced in Problem 1. We first explain the data structures. A *polygon* is implemented as a doubly linked list of vertices, where each vertex has a link to its successor and predecessor vertices (the `next` and `prev` pointers, respectively). In addition the x and y position of each vertex are stored in `pos[0]` and `pos[1]`.

```
struct Node {
  float pos[2];
  int marked;                    /* only for GC */
  struct Node* next;
  struct Node* prev;
};
typedef struct Node node;
```

Finally, we have a *mark* (field `marked`) which is used exclusively by the garbage collector and should not be touched by the user program. For simplicity, we store this as a whole `int` instead of as a bit.

There is a static array `polygon` of size `N` containing all vertices. There is also a pointer to the beginning of a list of free vertices, `free_ptr`, and a single root pointer, `root_ptr` pointing to the polygon. The free pointer must be maintained by the garbage collector; the root pointer is maintained by the user program and may not be changed by the garbage collector.

```
#define N (1<<20)
node polygon[N];               /* polygon store, max 1M vertices */
node* free_ptr;                /* free list */
node* root_ptr;                /* root pointer */
```

First, the garbage collector. This function is invoked either when the free list is empty when a node supposed to be allocated, or explicitly from the user program to "clean up" the store.

```
void gc () {                   /* call when free_ptr == NULL */
  mark(root_ptr);
  sweep();
}
```

1. (3 pts) Traversal of the heap during the marking phase of a mark-and-sweep garbage collector is (circle correct answer)

   (i) Depth first

   (ii) Breadth first

   (iii) Best first

   (iv) Arbitrary

2. (5 pts) The `mark` function takes a node as argument and marks all nodes reachable from the given node. Rather than using pointer reversal techniques, this `mark` function should be recursive. Complete the definition of `mark`. There are many solutions—for calibration, our solution adds 5 lines to the function body.

```
void mark(node* v) {









}
```

3. (7 pts) Next, complete the function `sweep`. Maintain the free list as a **singly linked** list with `next` pointing to the next free element. There are many solutions–for calibration, our solutions adds 7 lines to the function body.

```
void sweep () {
  int i;
  node* v;                /* use if needed */
  free_ptr = NULL;        /* initialize free list */
  for (i=0; i<N; i++) {









  }
}
```

6. Cyclone (15 points)

In this problem we reconsider the garbage collector sketched above and port it from C to Cyclone. We have left out some type declarations for you to fill in.

```
struct Node {
  float pos[2];
  int marked;                    /* only for GC */
  struct Node* next;
  struct Node* prev;
};
typedef struct Node node;

#define N (1<<20)
node polygon[N];                 /* polygon store, max 1M vertices */

_________ free_ptr;  /* 1 */     /* free list */

_________ root_ptr;  /* 2 */     /* root pointer */


_________ alloc () { /* 3 */     /* return pointer to new vertex */

  _________ new_ptr;  /* 4 */
  if (!free_ptr) {
    gc();
    if (!free_ptr) {
      printf("Out of space!\n");
      exit(0);
    }
  }
  new_ptr = free_ptr;
  free_ptr = free_ptr->next;
  new_ptr->next = NULL;          /* init next pointer */
  new_ptr->prev = NULL;          /* init prev pointer */
  new_ptr->marked = 0;           /* unmarked */
  /* do not initialize pos[2] */
  return new_ptr;
}
```

```
int main () {

  _________ p1;   /* 5 */

  _________ p2;   /* 6 */
  init();                        /* initialize polygon store */
  p1 = alloc();                  /* create two-vertex "polygon" */
  p2 = alloc();
  p1->next = p2; p1->prev = p2;
  p2->next = p1; p2->next = p1;
  root_ptr = p1;
  gc();
  return 0;
}
```

1. (6 pts) For each of the 6 missing types, fill in the most precise type among the following which makes the code type-check without any implicit casts (and hence without warnings).

   (i) `node@` (most precise)

   (ii) `node*`

   (iii) `node?` (least precise)

2. (3 pts) Assume that Cyclone considers global variables as living on the heap (region `'H`). Each of the following expressions denotes a pointer. Indicate the region that this pointer points to, or write *unknown* if this cannot be determined.

   (i) `p1` in function `main`. _____

   (ii) `&p1` in function `main`. _____

   (iii) `new_ptr` in function `alloc`. _____

3. (3 pts) The `pos[]` fields are not initialized in the `alloc` function. Explain why this does not compromise safety of Cyclone.