CMSC 313 Spring 2010
Exam 3
May 17, 2010

Name_____Score _____

UMBC Username _____

Notes:

a. Please write clearly. Unreadable answers receive no credit.

b. There are no intentional syntax errors in any code provided with this exam. If you think you see an error that would affect your answer, please bring it to my attention

c. This exam is worth 85 points

Enjoy the summer at the beach !!

Multiple Choice (2 points each)

Write the letter of the **BEST** answer on the line provided for each question.

1._____ In the memory hierarchy, level ***k*** is considered a cache for level ***k+1*** because
- A. it's smaller and faster
- B. it's bigger and slower
- C. it's always in memory
- D. it's always on the CPU chip

2. _____ PROM stands for
- A. Partial Read Only Memory
- B. Programmable Read Only Memory
- C. Potential Read Only Memory
- D. Permanent Read Only Memory

3. _____ (True / False) Using segregated free lists to find a free block in the heap approximates the "best fit" algorithm in terms of memory usage.

4. _____ Dr. Frankenstein has a disk that rotates at 7,200 RPM (8ms per full revolution), has an average seek time of 5ms, and has 1000 sectors per track. How long (approximately) does the average 1-sector access take?
- A. Not enough information to determine the answer
- B. 13ms
- C. 9ms
- D. 0.5ms

5. _____ A zombie process is created when
- A. a child process terminates, but its parent process continues to run
- B. a parent process terminates, but its child process continues to run
- C. when a process is sent the SIGZOMBIE signal
- D. the user types "control-Z" while the process is running

6. _____ A signal handler is
- A. a function in the kernel that sends signals
- B. a function in the kernel that executes when a process receives a signal
- C. a user function that sends signals
- D. a user function that executes when a process receives a signal

7._____ In project 6, we provided code for an implicit list allocator. Many students improved this code by creating a linked list of free blocks. Why did this change increase the performance of the allocator?

A. Traversing a linked list is significantly faster than moving from block to block in the implicit list.

B. The implicit list had to include every block in memory, but the linked list could just include the free blocks.

C. Having a linked list made coalescing significantly faster.

D. None of the above.

8. _____ Imagine a process (called ”process A”) that calls **`fork( )`** three times. If all three child processes terminate before process A is picked by the kernel to be run again, how many times could process A receive SIGCHLD?

A. Not enough information to determine

B. 1

C. 3

D. 1 or 3

9. _____ With respect to disk storage, the term "rotational latency" refers to

A. The number of sectors that pass under the read/write head on each revolution

B. The time required to wait for the desired sector to pass under the read/write head

C. The time it takes for the disk to make one complete revolution

D. The time required to wait for the desired track to pass under the read/write head

10. _____ The system call **`execl( )`** is used to

A. create a child process

B. load and execute a program

C. execute special code in the kernel

D. switch to "exec mode"

11. _____ RAM is considered to be "volatile memory" because

A. its state is occasionally changed without warning

B. its state is lost when power is removed

C. its state fluctuates, but is always recoverable

D. none of the above

12. _____ Which of the following statements regarding disk storage is **FALSE**?

A. The time to access a sector on a disk is affected by the disk's rotational speed.

B. All tracks have the same number of sectors

C. A cylinder is a collection of tracks

D. Transfer time is generally the smallest component of disk access time

Short Answer

13. **(4 points)** In no more than 2 sentences, describe the difference between ***internal fragmentation*** and ***external fragmentation***.

14. **(6 points)** What is a "page table"? Describe how a page table is used to implement virtual memory.

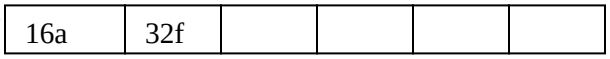
15. **(6 points)** What is a "page fault" and what actions are taken when it occurs?

Heap Memory Allocation

16. (10 points) Consider an allocator with the following specification:

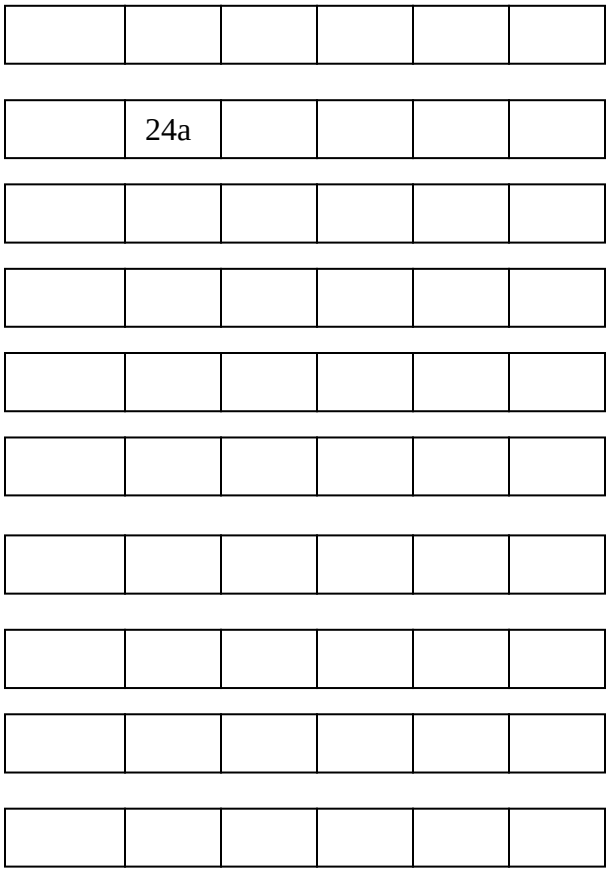
a. Uses a single explicit free list.
b. All memory blocks have a size that is a multiple of 8 bytes and is at least 16 bytes
c. All headers, footers, and pointers are 4 bytes in size
d. Headers consist of the block size in the upper 29 bits, a bit indicating if the block is allocated in the lowest bit (bit 0), and a bit indicating if the previous block is allocated in the second lowest bit (bit 1).
e. Allocated blocks consist of a header and a payload (no footer)
f. Free blocks consist of a header, two pointers for the next and previous free blocks in the free list, and footer at the end of the block.
g. All freed blocks are immediately coalesced.
h. The heap starts with 0 bytes, never shrinks, and only grows large enough to satisfy memory requests.
i. The heap contains only allocated blocks and free blocks. There are is no space used for other data or special blocks to mark the beginning and end of the heap.
j. When a block is split, the lower (left) part of the block becomes the allocated part and the upper (right) part becomes the new free block.
k. Any newly created free block (whether it comes from a call to free, the upper part of a split block, or the coalescing of several free blocks) is inserted at the beginning of the free list.
l. All searches for free blocks start at the head of the list and walk through the list in order.
m. If a request can be fulfilled by using a free block, that free block is used. Otherwise the heap is extended only enough to fulfill the request. If there is a free block at the end of the heap, this can be used along with the new heap space to fulfill the request.

Below you are given a series of memory requests as they might appear in a user's program. You are asked to show what the heap looks like after each request is completed using a **first fit** placement policy. The heap is represented as a row of boxes, where each box is a single block on the heap, and the bottom of the heap is the left-most box. Simulate the calls to **`malloc( )`** or **`free( )`** on the left by marking each block in the corresponding row. In each block you should write the total size (including headers and footers) of the block in bytes and either **'f'** or **'a'** to mark it as free or allocated, respectively. For example, the following heap contains an allocated block of size 16, followed by a free block of size 32.



Simulate the memory requests using **FIRST FIT** for block allocation

```
ptr1 = malloc(32);
ptr2 = malloc(16);
ptr3 = malloc(10);
ptr4 = malloc(40);
free(ptr2);
free(ptr4);
ptr5 = malloc(20);
free(ptr1);
ptr6 = malloc(20);
free(ptr3);
```



Dynamic Memory Coding

17. (10 points) Consider a heap memory allocator that uses an ***implicit free list***. Each memory block, either allocated or free, has a size that is a multiple of eight (8) bytes. Thus, (assuming a 32-bit integer) only the 29 higher order bits in the header and footer are needed to record the block size (in bytes), which includes the header and footer. The usage of the remaining 3 lower order bits is as follows:

bit 0 indicates the use of the current block: 1 for allocated, 0 for free.
bit 1 indicates the use of the previous adjacent block: 1 for allocated, 0 for free.
bit 2 is unused and is always set to be 0.

Five helper routines are defined to facilitate the implementation of **`free(void *p)`.** The functionality of each routine is explained in the comment above the function definition. **Write the letter** of the code that correctly completes the function on the line provided in the function body.

```
/* given a pointer p to the payload of an allocated block, i.e.,
     p is a pointer returned by some previous malloc() call;
     returns the pointer to the header of the block
*/
void * header(void* p)
{
     void *ptr;
     ____________________;
     return ptr;
}
     A. ptr = p - 1;
     B. ptr = (void *)((int *)p - 1);
     C. ptr = (void *)((int *)p - 4);


/* given a pointer to a valid block header or footer,
     returns the size of the block
*/
int size(void *hp)
{
     int result;
     _______________________________;
     return result;
}

     A. result = (*hp) & (~0x7);
     B. result = ((*(char *)hp) & (~0x5)) >> 3;
     C. result = (*(int *)hp) & (~0x7);
```

```
/* given a pointer p to the payload of an allocated block,
 i.e. p is a pointer returned by some previous call to malloc()
 returns the pointer to the footer of the block
*/
void * footer(void *p)
{
     void *ptr;
     ____________________________;
     return ptr;
}
     A. ptr = p + size(header(p)) - 8;
     B. ptr = p + size(header(p)) - 4;
     C. ptr = (int *)p + size(header(p)) - 2;


/* given a pointer to a valid block header or footer,
     returns the state of the current block,
     1 for allocated, 0 for free
*/
int allocated(void *hp)
{
     int result;
     ____________________________;
     return result;
}
     A. result = (*(int *)hp) & 0x1;
     B. result = (*(int *hp)) & 0x0;
     C. result = (*(int *)hp) | 0x1;


/* given a pointer to a valid block header,
    returns the pointer to the header of previous block in memory
*/
void * prev(void *hp)
{
     void *ptr;
     ______________________________;
     return ptr;
}
     A. ptr = hp - size(hp);
     B. ptr = hp - size(hp - 4);
     C. ptr = hp - size(hp - 4) + 4;
```

Process Control

18. (15 points) Consider the following C program. For space reasons, we do not check return codes, so assume that all functions return normally.

```
int main()
{
     pid_t pid1, pid2;

     pid1 = fork();
     pid2 = fork();

     if (pid1 != 0 && pid2 != 0)
          printf("A\n");

     if (pid1 != 0 || pid2 != 0)
          printf("B\n");

     exit(0);
}
```

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

YES	YES	YES	NO	NO

Signals

19. (10 points) Write the output from the following C code in the box below.

```
pid_t pid;
int counter = 2;

void handler1(int sig)
{
     counter = counter - 1;
     printf("%d", counter);
     exit(0);
}

int main()
{
     signal(SIGUSR1, handler1);
     printf("%d", counter);
     pid = fork( );
     if (pid == 0)
     {
          while(1) {};  // simulate doing some work
     }
     kill(pid, SIGUSR1);
     wait(NULL);
     counter = counter + 1;
     printf("%d", counter);
     exit(0);
}
```

OUTPUT: