System Programming
Lab #6

2020-04-29

sp-tas

Lab Assignment #3 : Malloc Lab

- Download skeleton code & pdf from eTL
  - malloclab-handout.tar, malloclab-handout.pdf
- Hand In
  - Upload your files eTL
    - 압축파일 양식 : [학번]_[이름]_malloclab.tar (or .zip, etc)
    - Ex) 2020-12345_홍길동_malloclab.tar
  - A zip file should include
  - (1)mm.c (2)report
    - mm.c 양식 : mm-[학번].c eg) mm-2020-12345.c (제출할때만 바꿔서)
    - Report 양식 : [학번]_[이름]_malloclab_report.pdf (or .hwp, .txt etc)
- Please, READ the Hand-out thoroughly!
- Assigned : April 22
- Deadline : May 6, 23:59:59
- Delay policy : Same as before

Macros

- Why macros?
  - 1. Faster than function calls
  - 2. Encapsulate pointer arithmetic code lines
    - pointer arithmetic is error-prone & confusing
- Differences (with inline functions)
  - Macros are done with preprocessor (before compile time)
  - No call, return overheads
- Drawbacks
  - Less expressive than functions
  - Arguments are not type-checked
  - Unintended side effects
    - Ex) #define xsquared(x) (x*x)
    - What happens when xsquared(x++) is called?
  - Use () frequently!!

In the textbook

mm.c에서 Macro 자유롭게 사용

code/vm/malloc/mm.c

```
1   /* Basic constants and macros */
2   #define WSIZE       4       /* Word and header/footer size (bytes) */
3   #define DSIZE       8       /* Double word size (bytes) */
4   #define CHUNKSIZE  (1<<12)  /* Extend heap by this amount (bytes) */
5
6   #define MAX(x, y) ((x) > (y)? (x) : (y))
7
8   /* Pack a size and allocated bit into a word */
9   #define PACK(size, alloc)  ((size) | (alloc))
10
11  /* Read and write a word at address p */
12  #define GET(p)       (*(unsigned int *)(p))
13  #define PUT(p, val)  (*(unsigned int *)(p) = (val))
14
15  /* Read the size and allocated fields from address p */
16  #define GET_SIZE(p)  (GET(p) & ~0x7)
17  #define GET_ALLOC(p) (GET(p) & 0x1)
18
19  /* Given block ptr bp, compute address of its header and footer */
20  #define HDRP(bp)       ((char *)(bp) - WSIZE)
21  #define FTRP(bp)       ((char *)(bp) + GET_SIZE(HDRP(bp)) - DSIZE)
22
23  /* Given block ptr bp, compute address of next and previous blocks */
24  #define NEXT_BLKP(bp)  ((char *)(bp) + GET_SIZE(((char *)(bp) - WSIZE)))
25  #define PREV_BLKP(bp)  ((char *)(bp) - GET_SIZE(((char *)(bp) - DSIZE)))
```

code/vm/malloc/mm.c

Figure 9.43 Basic constants and macros for manipulating the free list.

Malloc Lab Preview

- Implementing your own dynamic storage allocator
  - mm_init, mm_malloc, mm_free, mm_realloc
- Ways to keep track of free, allocated blocks of memory
  - Implicit linked list of blocks
  - Explicit linked list of free blocks
  - Segregated lists of different size free blocks
- Other design decisions :
  - How to look for free blocks? (First fit, next fit, best fit, …)
  - Should the linked lists be doubly linked?
  - When to coalesce blocks?

mm.c에서 mm_check 함수를 정의하여 Heap Consistency Check 수행 (style 10pt)
디버깅 용도로 사용하고 제출 코드에서는 작동하지 않게 수정

Watch Out!

7 Programming Rules

- You should not change any of the interfaces in `mm.c`.
- You should not invoke any memory-management related library calls or system calls. This excludes the use of `malloc`, `calloc`, `free`, `realloc`, `sbrk`, `brk` or any variants of these calls in your code.
- You are not allowed to define any global or `static` compound data structures such as arrays, structs, trees, or lists in your `mm.c` program. However, you *are* allowed to declare global scalar variables such as integers, floats, and pointers in `mm.c`.
- For consistency with the `libc malloc` package, which returns blocks aligned on 8-byte boundaries, your allocator must always return pointers that are aligned to 8-byte boundaries. The driver will enforce this requirement for you.

mm.c 이외의 파일들은 수정하지 마세요.

Testing with trace files

- `-t <tracedir>`: Look for the default trace files in directory `tracedir` instead of the default directory defined in `config.h`.
- `-f <tracefile>`: Use one particular `tracefile` for testing instead of the default set of trace-files.
- `-h`: Print a summary of the command line arguments.
- `-l`: Run and measure `libc` malloc in addition to the student's malloc package.
- `-v`: Verbose output. Print a performance breakdown for each tracefile in a compact table.
- `-V`: More verbose output. Prints additional diagnostic information as each trace file is processed. Useful during debugging for determining which trace file is causing your malloc package to fail.

```
ta@sp3:~/yschoi/testing/malloclab-handout/src$ ./mdriver -f ./traces/short1.rep  -V
Team Name:implicit first fit
Member 1 :Dave OHallaron:droh
Measuring performance with gettimeofday().

Testing mm malloc
Reading tracefile: ./traces/short1.rep
Checking mm_malloc for correctness, efficiency, and performance.

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   66%      12  0.000001 20000
Total          66%      12  0.000001 20000

Perf index = 40 (util) + 40 (thru) = 80/100
```

Performance(100pt) 채점에 Perf index가 사용
가중치 0.6 기본값 / handout file 계산식 사용

Evaluation

- Our evaluation
  - Correctness(20 points)
  - Performance(100 points)
    - Space utilization + Throughput
  - Style(10 points)
  - Report(10 points)

Correctness(20 points) – valid
Performance(100 points) – perf index
Style(10 points) – comment and mm_check
Report(10 points)

Last year's Questions

- Trace file의 경로변경은 어떻게 하나요?
  - ->src/config.h
- mm_malloc의 input이 0인 경우의 return값?
  - -> NULL.
- Global/Static으로 array 선언하면 안되나요?
  - -> 네 안됩니다.
- 고친 부분이 없는데 Trace file을 돌릴 때 가끔 점수가 다르게 나옵니다.
  - -> 서버의 CPU 사용량에 따라 점수가 다르게 나올 수 있습니다.

끝

- 과제 기한 5월 6일까지
- 질문
  - etl Q&A 게시판
  - ta_sp20@dcslab.snu.ac.kr
- 다음 시간에
  - Lab Assignment #4