CSED211 컴퓨터SW시스템개론

Homwork #4

20180551
컴퓨터공학과
이준석

1. Exercise 10.6 on page 950.

10.6 ◆
What is the output of the following program?

```
1   #include "csapp.h"
2
3   int main()
4   {
5       int fd1, fd2;
6
7       fd1 = Open("foo.txt", O_RDONLY, 0);
8       fd2 = Open("bar.txt", O_RDONLY, 0);
9       Close(fd2);
10      fd2 = Open("baz.txt", O_RDONLY, 0);
11      printf("fd2 = %d\n", fd2);
12      exit(0);
13  }
```

file descriptor 0 : stdin / 1 : stdout / 2 : stderr

line 7에서, fd1 == 3

line 8에서, fd2 == 4

line 9에서, file descriptor table에서 4번 자리가 비워진다.

line 10에서, fd2 == 4

답 : fd2 = 4

2. Exercise 10.9 on page 950

10.9 ◆◆

Consider the following invocation of the `fstatcheck` program from Problem 10.8:

```
linux> fstatcheck 3 < foo.txt
```

You might expect that this invocation of `fstatcheck` would fetch and display metadata for file `foo.txt`. However, when we run it on our system, it fails with a "bad file descriptor." Given this behavior, fill in the pseudocode that the shell must be executing between the `fork` and `execve` calls:

```
if (Fork() == 0) { /* child */
    /* What code is the shell executing right here? */
    Execve("fstatcheck", argv, envp);
}
```

10.8 ◆◆

Write a version of the `statcheck` program in Figure 10.10, called `fstatcheck`, that takes a descriptor number on the command line rather than a filename.

statbuf.h (included by sys/stat.h)

```
/* Metadata returned by the stat and fstat functions */
struct stat {
    dev_t         st_dev;      /* Device */
    ino_t         st_ino;      /* inode */
    mode_t        st_mode;     /* Protection and file type */
    nlink_t       st_nlink;    /* Number of hard links */
    uid_t         st_uid;      /* User ID of owner */
    gid_t         st_gid;      /* Group ID of owner */
    dev_t         st_rdev;     /* Device type (if inode device) */
    off_t         st_size;     /* Total size, in bytes */
    unsigned long st_blksize;  /* Block size for filesystem I/O */
    unsigned long st_blocks;   /* Number of blocks allocated */
    time_t        st_atime;    /* Time of last access */
    time_t        st_mtime;    /* Time of last modification */
    time_t        st_ctime;    /* Time of last change */
};
```

statbuf.h (included by sys/stat.h)

Figure 10.9 The `stat` structure.

code/io/statcheck.c

```
1   #include "csapp.h"
2
3   int main (int argc, char **argv)
4   {
5       struct stat stat;
6       char *type, *readok;
7
8       Stat(argv[1], &stat);
9       if (S_ISREG(stat.st_mode))     /* Determine file type */
10          type = "regular";
11      else if (S_ISDIR(stat.st_mode))
12          type = "directory";
13      else
14          type = "other";
15      if ((stat.st_mode & S_IRUSR)) /* Check read access */
16          readok = "yes";
17      else
18          readok = "no";
19
20      printf("type: %s, read: %s\n", type, readok);
21      exit(0);
22  }
```

code/io/statcheck.c

Figure 10.10 Querying and manipulating a file's `st_mode` bits.

fstatcheck는 fstat을 이용하여 파일의 상태를 확인하게 될 것이다.

그런데, fstat 함수를 이용하기 위해서는 파일이 열려있는 상태여야만 한다.

statcheck에서 open 함수가 사용되지 않은 것으로 보아 fstatcheck에서도 open 함수가 사용되지 않을 것이다. 따라서 execve를 하기 전에 미리 open 함수를 통해 foo.txt 파일을 열어주어야 한다.

부모 프로세스와 자식 프로세스에서 어떤 open 함수도 쓰이지 않았다면 다음 open에서 채워질 file descriptor table의 entry는 3번째 칸이다. 즉, foo.txt를 open 할 때 할당되는 file descriptor 값은 3이 된다는 의미이다.

쉘 커맨드라인을 통해 redirection을 해주고 있으므로 stdin의 file descriptor를 foo.txt의 file descriptor로 변경해야 한다. 따라서 그것을 위해 Dup2 함수를 사용한다.

Dup2(fd, 0)는 file descriptor 0에 열려있던 것을 닫고 file descriptor 3이 가리키고 있는 것을 file descriptor 0도 가리키게 된다는 의미이다. 즉, 커맨드라인에서 원했던 redirection을 적절히 수행하게 된다.

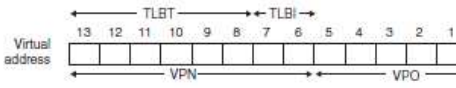
```
if(Fork() == 0) { /* child */
        fd = Open("foo.txt", O_RDONLY, 0); // fd == 3
        Dup2(fd, 0) // 0 is the file descriptor of the standard input

        Execve("fstatcheck", argv, envp);
}
```

3. Exercise 9.11 on page 912

9.6.4 Putting It Together: End-to-End Address Translation

In this section, we put it all together with a concrete example of end-to-end address translation on a small system with a TLB and L1 d-cache. To keep things manageable, we make the following assumptions:

- The memory is byte addressable.
- Memory accesses are to *1-byte words* (not 4-byte words).
- Virtual addresses are 14 bits wide ($n = 14$).
- Physical addresses are 12 bits wide ($m = 12$).
- The page size is 64 bytes ($P = 64$).
- The TLB is 4-way set associative with 16 total entries.
- The L1 d-cache is physically addressed and direct mapped, with a 4-byte line size and 16 total sets.



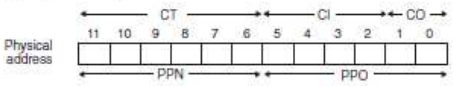
Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

(a) TLB: 4 sets, 16 entries, 4-way set associative

VPN	PPN	Valid
00	28	1
01	—	0
02	33	1
03	02	1
04	—	0
05	16	1
06	—	0
07	—	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	—	0
0C	—	0
0D	2D	1
0E	11	1
0F	0D	1

(b) Page table: Only the first 16 PTEs are shown



Idx	Tag	Valid	Blk 0	Blk 1	Blk 2	Blk 3
0	19	1	99	11	23	11
1	15	0	—	—	—	—
2	1B	1	00	02	04	08
3	36	0	—	—	—	—
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	—	—	—	—
7	16	1	11	C2	DF	03
8	24	1	3A	00	51	89
9	2D	0	—	—	—	—
A	2D	1	93	15	DA	3B
B	0B	0	—	—	—	—
C	12	0	—	—	—	—
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	—	—	—	—

(c) Cache: 16 sets, 4-byte blocks, direct mapped

Figure 9.20 **TLB, page table, and cache for small memory system.** All values in the TLB, page table, and cache are in hexadecimal notation.

P = 64 = 2^6 이므로 p = 6

TLB의 set은 16/4 = 4(개)

TLBI는 2bit

VPO는 6bit

VPN은 8bit

PPO는 6bit

PPN은 6bit

TLBT는 6bit

cache set은 총 16 = 2^4(개)

CI는 4bit

CO는 2bit

CT는 6bit

VA : 0x027c

A. Virtual address format

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	1	1	1	0	0

B. Address translation

Parameter	Value
VPN	0x09
TLB index	0x1
TLB tag	0x02
TLB hit? (Y/N)	N
Page fault? (Y/N)	N
PPN	0x17

C. Physical address format

11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	1	1	1	0	0

D. Physical memory reference

Parameter	Value
Byte offset	0x0
Cache index	0xF
Cache tag	0x17
Cache hit? (Y/N)	N
Cache byte returned	—

cache miss가 발생한다.

4. Exercise 9.13 on page 914

P = 64 = 2^6 이므로 p = 6

TLB의 set은 16/4 = 4(개)

TLBI는 2bit

VPO는 6bit

VPN은 8bit

PPO는 6bit

PPN은 6bit

TLBT는 6bit

cache set은 총 16 = 2^4(개)

CI는 4bit

CO는 2bit

CT는 6bit

VA : 0x0040

A. Virtual address format

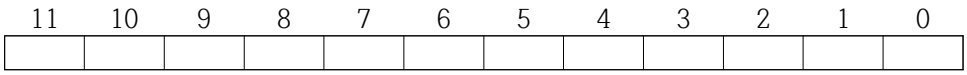
13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0

B. Address translation

Parameter	Value
VPN	0x01
TLB index	0x1
TLB tag	0x00
TLB hit? (Y/N)	N
Page fault? (Y/N)	Y
PPN	—

page fault가 발생하면 C와 D는 빈칸으로 둔다.

C. Physical address format



D. Physical memory reference

Parameter	Value
Byte offset	
Cache index	
Cache tag	
Cache hit? (Y/N)	
Cache byte returned	

5. Exercise 9.14 on page 915

9.14 ◆◆

Given an input file `hello.txt` that consists of the string `Hello, world!\n`, write a C program that uses `mmap` to change the contents of `hello.txt` to `Jello, world!\n`.

```c
/*
 * 20180551 컴퓨터공학과 이준석
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>


int main(int argc, char **argv) {

    char buf = 'J';
    int flag = PROT_WRITE | PROT_READ;
    size_t pagesize = getpagesize();

    int fd = open("hello.txt", O_RDWR);
    char* file = (char*)mmap((void*)(pagesize*(1<<20)), pagesize, flag, MAP_FILE | MAP_PRIVATE, fd, 0);

    write(fd, &buf, 1);

    munmap(file, pagesize);
    close(fd);

    return 0;
}
```

6. Exercise 9.15 on page 915

9.15 ◆
Determine the block sizes and header values that would result from the following sequence of `malloc` requests. Assumptions: (1) The allocator maintains double-word alignment and uses an implicit free list with the block format from Figure 9.35. (2) Block sizes are rounded up to the nearest multiple of 8 bytes.

Request	Block size (decimal bytes)	Block header (hex)
`malloc(4)`	_____	_____
`malloc(7)`	_____	_____
`malloc(19)`	_____	_____
`malloc(22)`	_____	_____

Request	Block size (decimal bytes)	Block header (hex)
malloc(4)	8	0x9
malloc(7)	16	0x11
malloc(19)	24	0x19
malloc(22)	32	0x21

block header에서 하위 3비트는 할당 여부에 관한 비트이다.

할당이 되어 있으면 LSB가 1이고 아니라면 0이다.

모두 malloc으로 인해 할당이 되어 있으므로 LSB가 1이 된다.

7. Exercise 9.19 on page 916

9.19 ◆
You are given three groups of statements relating to memory management and garbage collection below. In each group, only one statement is true. Your task is to indicate which statement is true.

1. (a) In a buddy system, up to 50% of the space can be wasted due to internal fragmentation.
   (b) The first-fit memory allocation algorithm is slower than the best-fit algorithm (on average).
   (c) Deallocation using boundary tags is fast only when the list of free blocks is ordered according to increasing memory addresses.
   (d) The buddy system suffers from internal fragmentation, but not from external fragmentation.
2. (a) Using the first-fit algorithm on a free list that is ordered according to decreasing block sizes results in low performance for allocations, but avoids external fragmentation.
   (b) For the best-fit method, the list of free blocks should be ordered according to increasing memory addresses.
   (c) The best-fit method chooses the largest free block into which the requested segment fits.
   (d) Using the first-fit algorithm on a free list that is ordered according to increasing block sizes is equivalent to using the best-fit algorithm.
3. Mark&Sweep garbage collectors are called conservative if
   (a) They coalesce freed memory only when a memory request cannot be satisfied.
   (b) They treat everything that looks like a pointer as a pointer.
   (c) They perform garbage collection only when they run out of memory.
   (d) They do not free memory blocks forming a cyclic list.

1. 답 : (a)

(b) first-fit은 best-fit 방식보다 빠르다.

first-fit은 속도는 빠르지만 메모리 효율이 나쁘다.

best-fit은 속도는 느리지만 메모리 효율이 좋다.

(c) boundary tag를 이용하면 상수 시간 안에 free가 수행된다.

(d) buddy system도 external fragmentation이 발생한다.

2. 답 : (d)

(a) external fragmentation은 여전히 발생한다. 만약 첫 free block이 1000바이트이고 프로그래머가 할당하려는 공간이 2바이트라면 first-fit 방식에서는 처음으로 적합한 free block에 할당하므로 1000바이트 free block에 2바이트가 할당이 된다. 그리고 1000바이트의 다음 free block이 300바이트였다면, 만약 프로그래머가 500바이트를 할당하려고 시도할 때 공간은 충분하지만 할당을 할 수 없게 된다. 즉, external fragmentation이 발생한다.

(b) 그럴 필요 없다. 어차피 best-fit은 free list 전부를 탐색한다.

(c) best-fit은 메모리 효율성을 최대화하는 방식이다. 가장 큰 free block을 선택한다면 그 본질적인 의의를 잃게 된다. 따라서 틀렸다.

(d) 옳다. 블록 크기에 따른 오름차순으로 free list가 정렬되어 있다면 first-fit에 따라 할당하려는 크기에 적합한 첫 free block을 찾았을 때 가장 작은 크기의 free block이 선택된다. 따라서 free list 전부를 탐색하여 적합한 크기의 가장 작은 block을 선택하는 best-fit 방식과 본질적으로는 같다.

3. 답 : (b)

GC가 conservative하다는 것은 reachable을 reachable이라고 제대로 판단하지만 unreachable를 reachable이라고 판단할 수 있다는 의미이다.

mark & sweep 방식을 사용하는 garbage collector는 conservative하다 만약

(a) garbage collector의 conservative 여부는 coalescing과 직접적인 관련이 없다. 따라서 거짓

(c) 메모리를 전부 사용하였을 때만 garbage collection을 한다고 하지만, 만약 reachable과 unreachable을 정확히 구별한다면 그것은 conservative하지 않다.