**시스템 소프트웨어와 실습**

**연습과제 #3**

**보고서**

**이름: 정여준**

**학번: 2017112138  
학과: 컴퓨터공학과  
교수님: 문봉교 교수님**

**강의명: 시스템소프트웨어와 실습**

**1. Chapter 9의 Homework problem 9.14, 9.15를 해결하고 보고서에 포함한다.**

**<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.**

MAPShared옵션으로 mapping되지 않으면 메모리 mapping된 개체에 대한 변경내용이 반영되지 않는다는 것을 알아야한다.

#include <stdio.h>

#include <assert.h>

#include "vm/csapp.h”

int mkfile(char\* filename, char\* content) { //content를 filename의 파일에 쓰고 저장하는 함수

int fd;

umask(DEF\_UMASK);

fd = Open(filename, O\_WRONLY|O\_CREAT|O\_TRUNC, DEF\_MODE);

/\* O\_WRONLY: 쓰기 전용 모드로 파일 열기, O\_CREAT: 필요한 경우 파일 생성, O\_TRUNC: 존재하던 데이터 모두 삭제 \*/

Write(fd, content, strlen(content));

/\* fd: 데이터 전송 영역을 나타내는 파일 디스크립터, content: 전송할 데이터를 가지고 있는 버퍼의 포인터 \*/

Close(fd);

}

int main(int argc, char\* argv[]) {

mkfile("hello.txt", "Hello, world!"); // Hello, world!를 입력한 hello.txt파일 생성

struct stat stat;

int fd;

char\* bufp;

size\_t size;

fd = Open("hello.txt", O\_RDWR, 0); //O\_RDWR: 읽기, 쓰기 겸용 모드로 파일 열기

fstat(fd, &stat);

size = stat.st\_size;

/\*size는 맵핑시킬 메모리 영역의 길이, PROW\_WRITE: 쓰기 가능한 페이지, MAP\_SHARED: 동일 파일을 맵핑한 모든 프로세스들이 공유 객체 접근에 대한 동기화를 위해 munmap을 사용, fd는 파일기술자, 맵핑할 때 size의 시작점을 0으로 지정 즉, 배열 첫번째에 있는 원소라고 생각하면 된다 \*/

bufp = Mmap(NULL, size, PROT\_WRITE, MAP\_SHARED, fd, 0);

\*bufp = 'J'; //배열 0번째 인덱스 즉, H를 J로 바꾼다.

Munmap(bufp, size); //Mmap()으로 만들어진 맵핑 제거

return 0;

}

**<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** | **Block Header** |
| --- | --- | --- |
| malloc(4) | 8 | 0x9 |
| malloc(7) | 16 | 0x11 |
| malloc(19) | 24 | 0x19 |
| malloc(22) | 32 | 0x21 |

malloc(4) => Block Size 4 + 4 =8 => 8 | Block Header 00001001 = 0x9

malloc(7) => Block Size 4 + 7 = 11 -> 16 | Block Header 00010001 = 0x11

malloc(19) => Block Size 4 + 19 = 23 -> 24 | Block Header 00011001 = 0x19

malloc(22) => Block Size 4 + 22 = 26 -> 32 | Block Header 00100001 = 0x21

**<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);**

**}**

Int dup(int oldfd); Int dup2(int oldfd, int newfd); 파일 디스키립터를 복제하는 함수

파일지정자 oldfd에 대한 복사본을 생성한다. 성공적으로 수행될 경우 oldfd 지정자에 대한 복사본은 서로 공유되어서 사용된다. 즉 lock, 파일위치 포인터, 플래그 등을 공유한다. 만약 원본 파일지정자에서 위치변경이 일어났다면, 다른 복사된 파일지정자에도 영향을 미친다. 그러나 이 두개의 파일지정자 간 close-on-exe 플래그는 공유되지 않는다. Dup()를 이용해서 복사되어지는 새로운 파일 지정자는 사용되지 않는 가장 작은 파일 지정자를 이용한다. Dup2()는 디스크립터는 파일 지정자를 지정할 수 있는데, 이전에 열린 newfd가 있다면 닫고 나서, oldfd를 newfd에 복사하면 된다. 따라서 다음과 같이 코드를 짜면 된다.

if (Fork() == 0) {

Dup2(0, 3);

Execve("fstatcheck", argv, envp);

}

**<11.10>**

**A. Write an HTML form for the CGI adder function in Figure 11.27. Your form should include two text boxes that users fill in with the two numbers to be added together. Your form should request content using the GET method.**

<!DOCTYPE html>

<html>

<head>

<meta charset = “UTF-8”/>

<title>Q11.10</title>

</head>

<body>

<form action = “/cgi-bin/adder” method = “get” >

<br>first number: <input type = “text” name = “num1”/>

<br>second number: <input type = “text” name = “num2”/>

<br><input type = “submit” value = ”okay”/>

</form>

</body>

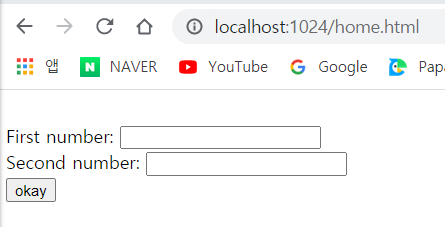
</html>

form method = “get”을 통해서 get방식으로 요청하고 num1과 num2로 받은 값을 adder.c 파일을

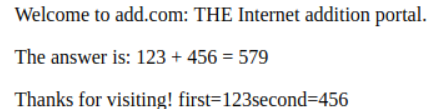
이용해서 만든 실행가능한 파일 adder로 전달해서 adder에서 처리한다.

**B. Check your work by using a real browser to request the form from Tiny, submit the filled-in form to Tiny, and then display the dynamic content generated by adder.**

http://localhost:1024/home.html



http://localhost:1024/cgi-bin/adder?first=123&second=456



이전의 adder.c

if((buf=getenv(“QUERY\_STRING”)) != NULL){

P = strchr(buf, ’&’);

\*p = ‘\0’;

strcpy(arg1,buf);

strcpy(arg2, p+1);

n1 = atoi(arg1);

n2 = atoi(arg2);

}

수정

if((buf=getenv(“QUERY\_STRING”)) != NULL){

P = strchr(buf, ’&’);

\*p = ‘\0’;

strcpy(arg1,buf);

strcpy(arg2, p+1);

n1 = atoi(strchr(arg1, ’=’)+1);

n2 = atoi(strchr(arg2, ’=’)+1);

}

**<12.29>**

**Can the following program deadlock? Why or why not?**

**Initially: a = 1, b = 1, c = 1.**

**Thread 1: Thread 2:**

**P(a); P(c);**

**P(b); P(b);**

**V(b); V(b);**

**P(c); V(c);**

**V(c);**

**V(a);**

일단 Deadlock상태는 프로세스가 자원을 얻지 못해서 다음 처리를 하지 못하

는 상태로 교착상태라고도 한다. 즉, 시스템적으로 한정된 자원을 여러 곳에서 사용하려고 할 때

발생한다.

스레드 2는 a에 관해서 조작하지 않고 초기 a는 1이다. 따라서 P(a), V(a)는 교착상태에 영향을 주

지 않는다. 그러므로 다음과 같이 나타낼 수 있다.

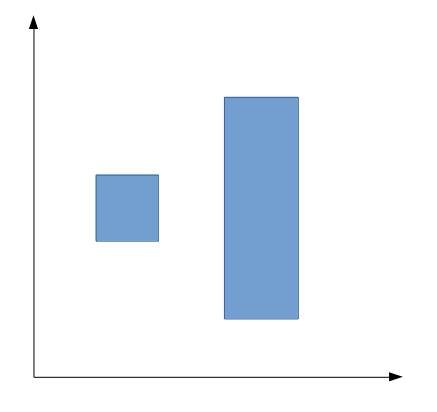
Thread 1: Thread 2:

P(b); P(c);

V(b); P(b);

P(c); V(b);

V(c); V(c);



따라서 Deadlock상태가 아니다.

**<12.30>**

**Consider the following program that deadlocks.**

**Initially: a = 1, b = 1, c = 1.**

**Thread 1: Thread 2: Thread 3:**

**P(a); P(c); P(c);**

**P(b); P(b); V(c);**

**V(b); V(b); P(b);**

**P(c); V(c); P(a);**

**V(c); P(a); V(a);**

**V(a); V(a); V(b);**

**A. For each thread, list the pairs of mutexes that it holds simultaneously.**

Thread 1: a와b, a와c ( P(a)와 V(a)사이에 P(b)와 V(b) 그리고 P(c)와 V(c)가 있다. )

Thread 2: b와c ( P(c)와 V(c)사이에 P(b)와 V(b)가 있다. )

Thread 3: a와b ( P(b)와 V(b)사이에 P(a)와 V(a)가 있다. )

**B. If a<b<c, which threads violate the mutex lock ordering rule?**

Thread 2는 c<b<a순으로 a<b<c를 위반하였고 Thread 3은 c<b<a순으로 a<b<c를 위반하였다.

**C. For these threads, show a new lock ordering that guarantees freedom from deadlock**

모든 Thread에서 동일한순서 P(a) P(b) P(c)를 유지한다.