과제 간단 해설 및 소개

정우영 조교

최종 업데이트 : 2013-12-14

들어가며

- 이거 자체는 1번 과제 끝나고부터 만들기 시작했는데 어쩌다보니 배포는 이제와서야... ㅜㅜ

- 대략적인 문제만 집었습니다.
- '프기실'에 해당하는 문제는 가능한 집지 않으려 합니다.

- 앞으로 추가할 거 있으면 하겠지만.. 이제 마지막이네요 ——;

sizeof 연산자

- array, struct/union에 대해선 할당 받은 크기
- 포인터에겐 포인터의 크기를 돌려줌

- array를 어떤 함수에게 넘기면, 해당 함수 안에선, 포인터로 접근하기 때문에 크기를 모름.

- 어떤 포인터가 가리키고 있는 메모리의 **크기**를 알 수 있는 방법은 없음

```
int a;
int b[5];
int *c = malloc(5 * sizeof(int));

struct st1 {
    char c[18];
};
union un1 {
    char c[14];
    int i[5];
};

struct st1 d;
struct st1 e[2];
union un1 f;
union un1 *g = malloc(3 * sizeof(union un1));

printf("sizeof a : %ld\n", sizeof a);
printf("sizeof b : %ld\n", sizeof b);
printf("sizeof c : %ld\n", sizeof c);
printf("sizeof d : %ld\n", sizeof d);
printf("sizeof e : %ld\n", sizeof e);
printf("sizeof f : %ld\n", sizeof f);
printf("sizeof g : %ld\n", sizeof g);
printf("sizeof pointer of array : %ld\n", size(b));
```

```
sizeof a : 4
sizeof b : 20
sizeof c : 8 (on 64-bit)
sizeof d : 18
sizeof e : 36
sizeof f : 20
sizeof g : 8
sizeof pointer of array : 8
```

```
long size(int *b)
{
    return sizeof b;
}
```

sizeof 연산자

- 문자열의 크기를 알아내기 위해선, sizeof가 아닌 strlen을 사용해야 함.

```
char h[16] = "hello?";
char *i = "hello?";

printf("sizeof h : %ld\n", sizeof h);
printf("sizeof i : %ld\n", sizeof i);
printf("strlen h : %ld\n", strlen(h));
printf("strlen i : %ld\n", strlen(i));
```

```
sizeof h : 16
sizeof i : 8
strlen h : 6
strlen i : 6
```

무모한 pointer dereferencing

- if (input_url[7] == ‘/’)
- 이 행동을 하기 전에 우선
  input_url의 크기가 7보다 큰가 확인합시다.
- strlen 구현은 딸랑 5줄이면 ok.

```
size_t strlen(const char *str)
{
    size_t size = 0;
    while(str[size++]);
    return size - 1;
}
```

if 남발

- if(input_url[0] != 'f')
- if(input_url[1] != 'i')
- if(input_url[2] != 'l')
- if(input_url[3] != 'e')
- … (이렇게 15번 비교)

- 많이 비효율적이긴 하지만,
- 윗 사례보단 차라리 나음. 안전하기 때문에.
- 그래도 이러진 맙시다...

우리에겐 strncmp가 있다.

- http://www.cplusplus.com/strncmp
- 직접 구현하는 것도 간단

```
33 int strncmp(const char *a, const char *b, size_t size)
34 {
35     for (size_t i=0; i<size; i++)
36     {
37         if (a[i] > b[i])
38             return -1;
39         else if (a[i] < b[i])
40             return 1;
41     }
42     return 0;
43 }
```

getenv 조작

- getenv로 받은 주소의 데이터를 덮어씀

```
28    char *str = “copy me!”;
29    char *home = getenv("HOME");
30    strcat(home, str);
```

➔ 기존 시스템의 환경 변수가 날아감

getenv 조작

- 간단한 확인 코드

```
28    char *home = getenv("HOME");
29
30    // look at the starting point and the ending point
31    for (int i=-200; i<200; i++)
32        putchar(home[i]);
```

- 결과

덮어쓰는 코드로 영향 받는 부분

```
s:/usr/bin/eclips                    ointos/src/utilsLC_IDENTIFICATION=ko_KR.UTF-
8PWD=/home/r.....                    usr/lib/jvm/jdk1.6.0_35LANG=en_US.UTF-
8LC_MEASUREMENT=ko_KR.UTF-
8SHLVL=1HOME=/home/minslLANGUAGE=enLOGNAME=minslSSH_CONNECTION=115.125.135.131 8253
115.125.135.131 29LESSOPEN=|
/usr/bin/lesspipe %sXDG_RUNTIME_DIR=/run/user/minslDISPLAY=localhost:11.0LESSCLOSE=/usr/bin/l
```

- 잠재적인 위협
- 이렇게 문제 터지면 디버깅도 힘들다

getenv 조작

- 제안 : 버퍼를 새로 만들어 받아옵시다.

```
char buffer[1024];
strcpy(buffer, getenv("HOME"));
strcat(buffer, str);
```

(물론 getenv("HOME") 접근이 안전하다고 가정함)

1 byte I/O

- 1 byte로 sys_read/sys_write 호출

```
char c;
read(fd1, &c, 1);
write(fd2, &c, 1);
```

- 만약 PA 1에서 운 좋게 pass됐어도, PA 2에선 long time을 받았을 것임.
- 매우 느립니다.

- 따라서, 적당히 큰 버퍼를 잡아서 사용합시다.
- 권장 : 4K~128K (4,096 ~ 131,072 bytes)

Define한 값을 사용합시다.

- 사례 :

```
if (errno == 2)    // 2 == ENOENT
{
        return -1; // #define ERR_NO_FILE (-1)
}
```

- 과제를 만든 사람들도 -1 -2 -3 -4가 무엇을 의미하는지 기억하지 않습니다.
- 마찬가지로 에러 번호 대신에 ENOENT같은 이름을 씁니다.

- 다음과 같이 하세요.
- return ERR_INVAL_URL;

쓰지 않을 파일은 열지도 말라

- 사용하지 않을 파일을 만들고 보는 경우.

```
int fdout = open(output_path, O_CREAT | O_RDWR, ...)

// returns true if input_url is valid
if ( url_validate(input_url) == false)
    return ERR_INVALID_URL;
```

URL 검사를 하지 않고
쓸 파일부터 만들고 보는 경우
혹은
읽을 파일이 열리지 않지만
쓸 파일부터 열고 보는 경우

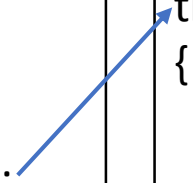
- 만약 O_TRUNC flag가 붙어있으면 지우면 안 되는 멀쩡한 파일을 하나 날릴지도 모름.

자원을 해제할 때

- 최근 문제가 된 코드를 보면,
  double free를 시도하던가 close(0)을 시도한 코드가 많음

```
// main thread
int *connfd = malloc(4);
*connfd = accept();
pthread_create(..., th_main, connfd);
free(connfd); // double free
```

```
th_main(void *arg)
{
    int cfd = *(int *)arg;
    free(arg); // double free
}
```



- thread 사이에선 address space 공유!

자원을 해제할 때

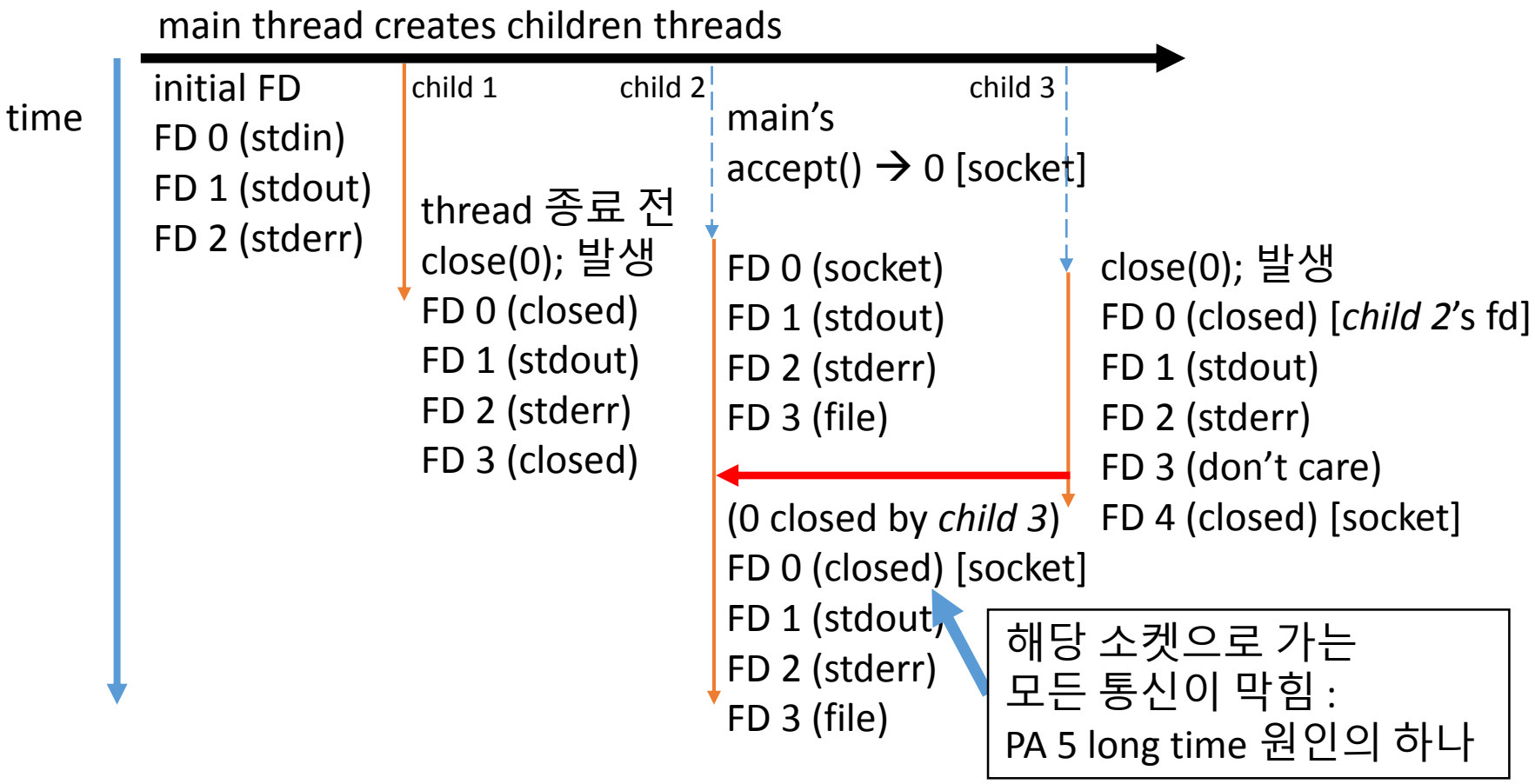
- close(0);
- 문제 예시 :

```
void *thread_main(void *arg)
{
   int fd; ← gcc는 0으로 초기화. (vs에선 0xcccc….)

   // 매우 복잡한 코드,
   // 하지만 fd는 사용하지 않을 '수' 있음

   close(connfd);
   close(fd); ← closing FD 0
   return NULL;
}
```

- 더 이상 표준 입력(stdin)에서 입력을 받지 못 함
- 새로운 (파일) 디스크립터가 0번으로 열림

자원을 해제할 때

- close(0); // 더 나쁜 예시 :



자원을 해제할 때 : 변수 초기값

- 쓰레드를 사용할 땐, 주소 공간을 공유함을 주의.
- 이런 불상사를 방지하기 위해, 변수를 다음처럼 선언할 수 있음.
- int fd = -1;
- (file operation에서 fd가 -1일 경우 아무 일도 안 함; 최소한 예측 불가능한 사태가 일어나진 않음)

- 이건 한 예방책이고, 이런 문제가 애초에 발생하지 않도록 코드를 작성할 것.
- (모든 경로에 대해 자원 사용 여부를 확인하세요)