ProcessAPI

Operating System (OS)

안인규

강의자료는 KAIST 원유집 교수님 OS 강의 자료를 참고했습니다.

Overview

- `fork()`
- `exec()`
- `wait()`
- Separation of `fork()` and `exec()`
  - IO redirection
  - pipe

Creating a child process

`fork()`

- Create a child process
  - Child process는 parent process로부터 독립된 메모리 공간을 할당
  - Child process는 자체적인 **Register**와 program counter register(**PC**) 가짐
  - 새롭게 생성된 process는 생성 후 독립적으로 동작
  - Parent process의 경우, `fork()`는 Child Process의 PID를 반환
  - Child process의 경우, `fork()`는 0을 반환

Usage of fork()

p1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {   // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {                    // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

fork()

fork(): parent vs. child

p1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {   // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {                    // parent goes down this path (main)
        printf("hello, I am parent of %d (pid:%d)\n",
        rc, (int) getpid());
    }
    return 0;
}
```

Result (Nondeterministic)

```
prompt> ./p1
hello world (pid:29146)
hello, I am parent of 29147 (pid:29146)
hello, I am child (pid:29147)
prompt>
```

or

```
prompt> ./p1
hello world (pid:29146)
hello, I am child (pid:29147)
hello, I am parent of 29147 (pid:29146)
prompt>
```

Create the dependency between the processes

- 일부 경우, application의 실행 순서를 강제하고 싶어할 수 있음
  - 예: Parent process가 child process가 종료된 후에만 종료되도록 하는 경우

`wait()`

- Child process가 생성되면, parent process의 `wait()`은 child process가 실행되고 종료될 때까지 반환되지 않음
- 부모와 자식 프로세스는 서로 의존성이 없다

The usage of wait() System call

p2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
    } else {               // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
        rc, wc, (int) getpid());
    }
    return 0;
}
```

wait()

The wait() System Call

Result (Deterministic)

```
prompt> ./p2
hello world (pid:29266)
hello, I am child (pid:29267)
hello, I am parent of 29267 (wc:29267) (pid:29266)
prompt>
```

Running a new program

`exec()`

- 호출자가 자신과 다른 프로그램을 실행하고자 할 때 (`fork()` 후)
  - Launch an editor
  - `% ls -l`
- OS needs to load a new binary image, initialize a new stack, initialize a new heap for the new program.
- two parameters
  - The name of the binary file
  - The array of augments

```
char *argv[3];

argv[0] = “echo”;
argv[1] = “hello”;
argv[2] = 0;
exec(“/bin/echo”, argv);
printf(“exec error\n”);
```

Usage of exec()

p3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>

int main(int argc, char *argv[]){
    printf("hello world (pid:%d)\n", (int) getpid());
    int rc = fork();
    if (rc < 0) {              // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {      // child (new process)
        printf("hello, I am child (pid:%d)\n", (int) getpid());
        char *myargs[3];
        myargs[0] = strdup("wc");                  // program: "wc" (word count)
        myargs[1] = strdup("p3.c");     // argument: file to count
        myargs[2] = NULL;               // marks end of array
        execvp(myargs[0], myargs); // runs word count
        printf("this shouldn't print out");
    } else {                           // parent goes down this path (main)
        int wc = wait(NULL);
        printf("hello, I am parent of %d (wc:%d) (pid:%d)\n",
            rc, wc, (int) getpid());
    }
    return 0;
}
```

When exec() is called,...

- 기존 메모리의 내용을 새로운 binary file의 메모리 내용으로 대체
- 그렇기 때문에, `exec()`는 반환되지 않는다!
  (**새로운 프로그램의 실행 시작**)

Usage of exec()

Result

```
prompt> ./p3
hello world (pid:29383)
hello, I am child (pid:29384)
29 107 1030 p3.c
hello, I am parent of 29384 (wc:29384) (pid:29383)
prompt>
```

Why separating fork() and exec()?

- Why don't we just use something like "`forkandexec("ls", "ls -l")`"?
- `fork()`와 `exec()`를 분리하여 사용함으로써, 새로운 프로그램을 실행하기 직전에 다양한 설정을 조작할 수 있음 (예: **IO redirection**, **pipe** 구현 가능)
  - **IO redirection**

```
% cat w3.c > newfile.txt
```

  - **pipe**

```
% echo hello world | wc
```

IO redirection

```
% wc w3.c > newfile.txt
```

- 'wc' 는 'w3.c' 파일의 word 개수를 세는 프로그램
- Save the result of 'wc w3.c' to newfile.txt.
- How?
- 쉘(Shell, e.g., bash)은 command와 augment를 사용하여 fork() 와 exec()를 실행하는 프로그램
  - % ls -l ➔ shell calls fork() and exec("ls", "ls-l") ;
  - Before calling exec("wc", "wc w3.c"), the shell closes STDOUT (close(1)) and opens newfile.txt (open("newfile.txt)).

Details of IO redirection

p4.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>

int
main(int argc, char *argv[]){
    int rc = fork();
    if (rc < 0) {   // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
 // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc");                  // program: "wc" (word count)
        myargs[1] = strdup("p4.c");      // argument: file to count
        myargs[2] = NULL;                // marks end of array
        execvp(myargs[0], myargs);       // runs word count
    } else {                             // parent goes down this path (main)
        int wc = wait(NULL);
    }
    return 0;
}
```

IO redirection

Result

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```

File descriptor and file descriptor table

- File descriptor
  - File, pipe, directory, device를 나타내는 정수
  - Process는 file과 directory를 열기 위해, "File descriptor"를 이용
  - 각 process는 고유한 file descriptor table을 갖음
  - File descriptor table은 **0은 표준 입력(standard input)**, **1은 표준 출력(standard output)**, **2는 표준 오류(standard error)**를 의미

File offset

File descriptor and System Calls

- `open()`
  - 새로운 파일 객체 (file structure)를 할당하고, 새로운 file descriptor를 할당한 뒤, 새로 할당된 file descriptor가 새로운 파일 객체를 가리키도록 설정
  - 새로운 file descriptor를 할당할 때, file descriptor table에서 가장 작은 "사용 가능한 (free)" file descriptor 번호를 사용
- `close()`
  - File descriptor 'fd' 해제
  - 해당 파일 객체를 참조하는 file descriptor가 더 이상 없으면, 파일 객체도 해제
- `fork()`
  - Parent process의 file descriptor table을 child process로 복사
- `exec()`
  - File descriptor table을 유지

fork() and file descriptors

I/O Redirection (input)

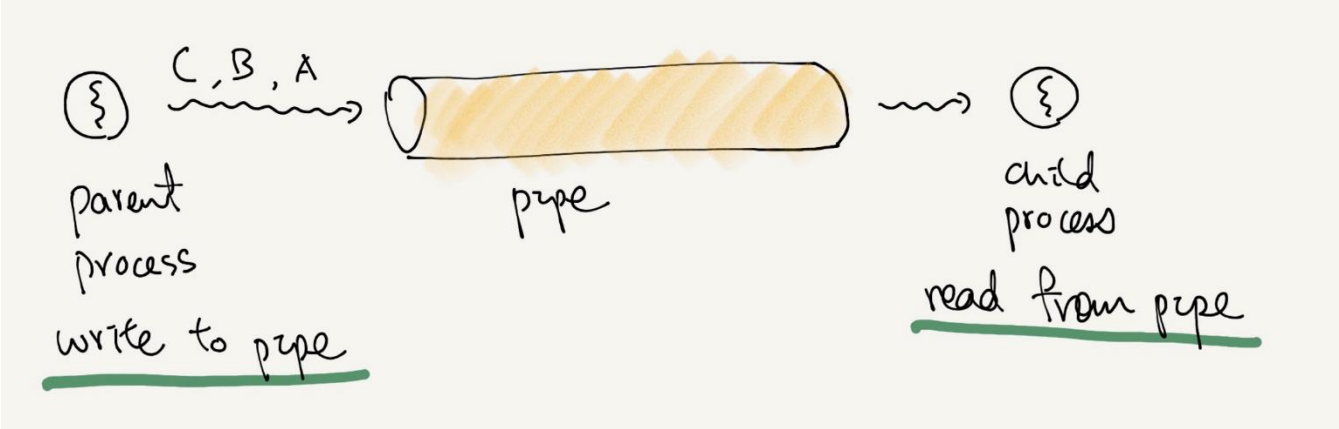
I/O Redirection (output)

pipe: '|'

```
% echo hello world | wc
```

pipe

- 한 process의 표준 출력 (STDOUT)이 다른 process의 표준 입력(STDIN)으로 전달
- Implemented with **dup()** and **pipe()**
- Key innovation of UNIX shell



dup(fd)

- **dup(n)** system call: find an empty slot from the beginning of the file descriptor table

pipe()

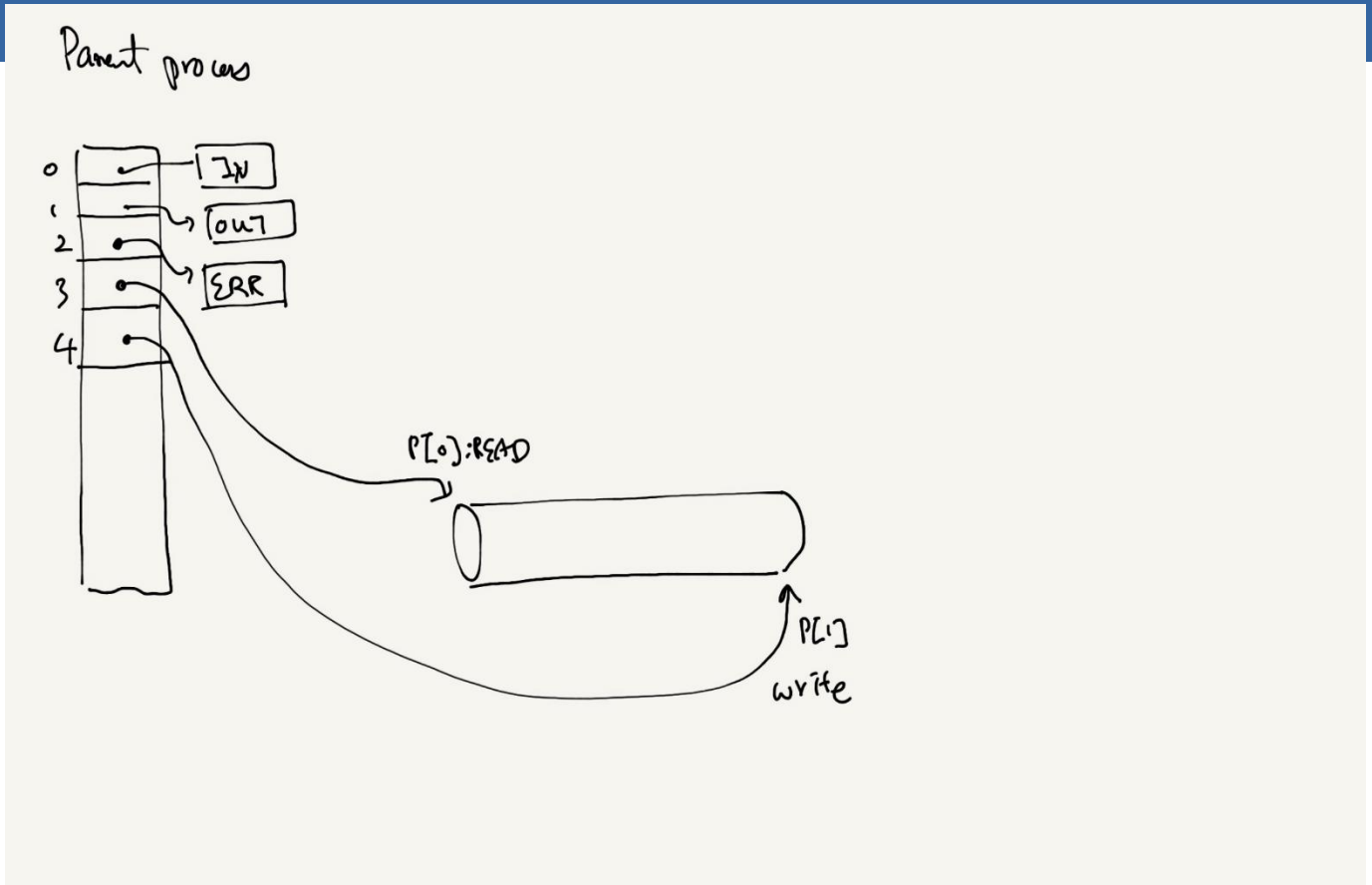
- 특수한 유형의 파일로, 커널 버퍼가 process에 file descriptor 쌍을 통해 제공: p[0]은 읽기 용 끝, p[1]은 쓰기 용 끝을 의미
- 읽기 프로세스 (reader process, 즉, read(p[0]))는 읽을 데이터가 없을 경우 block됨 (대기 상태)

pipe()

```
int p[2];
char *argv[2];

argv[0] = "wc";
argv[1] = 0;

pipe(p);
if(fork() == 0) {
  close(0);
  dup(p[0]);
  close(p[0]);
  close(p[1]);
  exec("/bin/wc", argv);
} else {
  close(p[0]);
  write(p[1], "hello world\n", 12);
  close(p[1]);
}
```



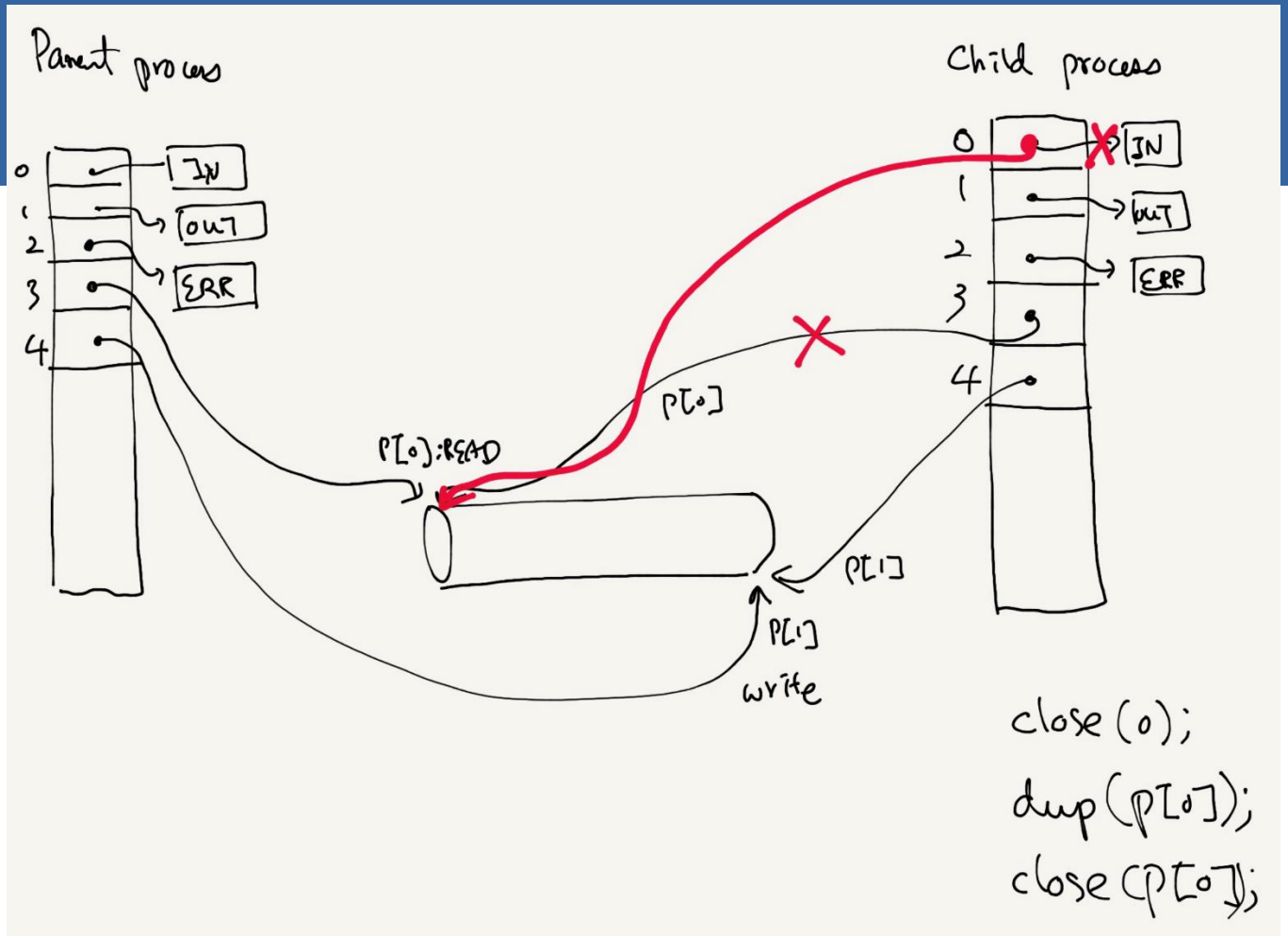

```
% echo hello world | wc
```

pipe()

```
int p[2];
char *argv[2];

argv[0] = "wc";
argv[1] = 0;

pipe(p);
if(fork() == 0) {
  close(0);
  dup(p[0]);
  close(p[0]);
  close(p[1]);
  exec("/bin/wc", argv);
} else {
  close(p[0]);
  write(p[1], "hello world\n", 12);
  close(p[1]);
}
```



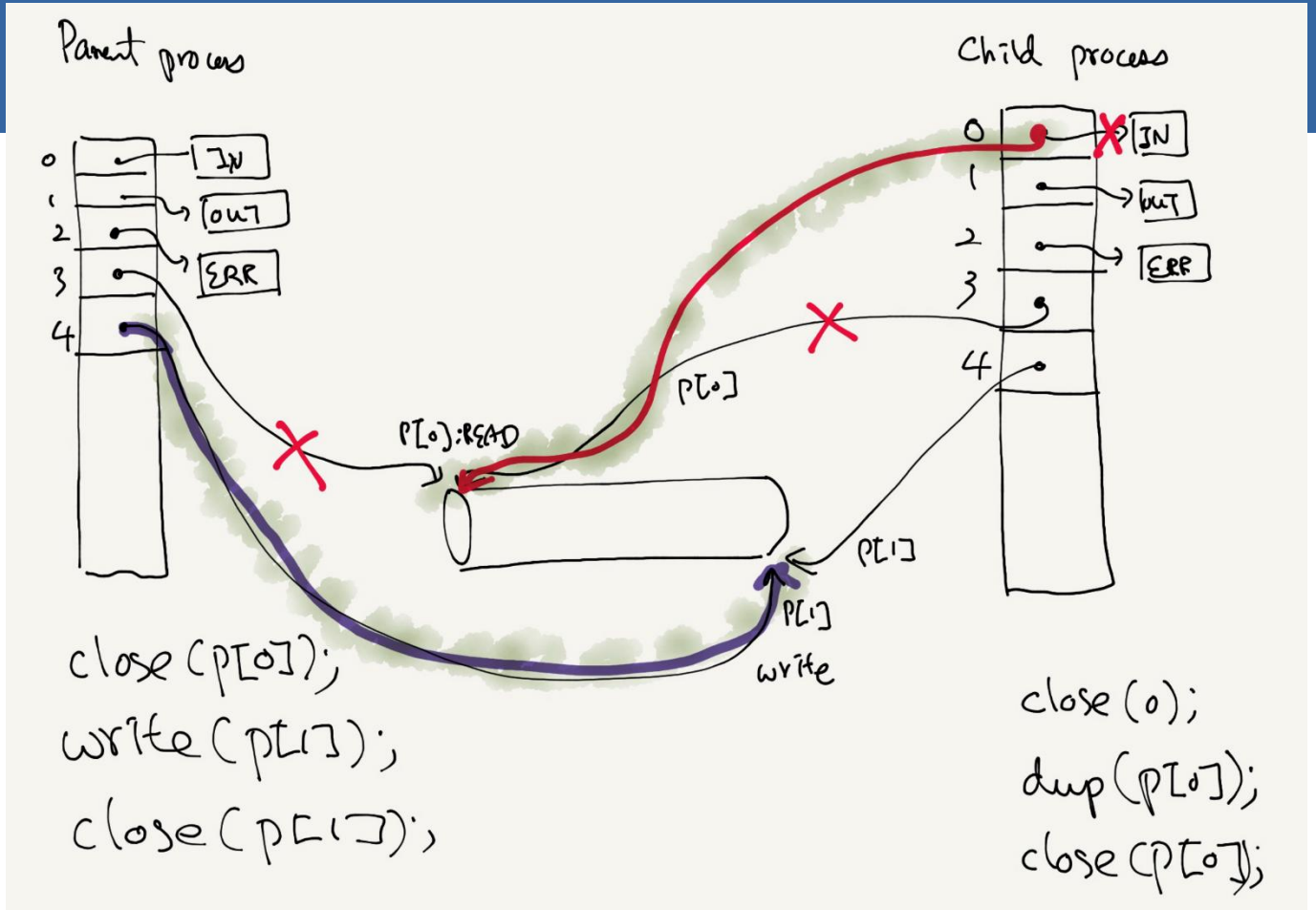

```
% echo hello world | wc
```

pipe()

```
int p[2];
char *argv[2];

argv[0] = "wc";
argv[1] = 0;

pipe(p);
if(fork() == 0) {
  close(0);
  dup(p[0]);
  close(p[0]);
  close(p[1]);
  exec("/bin/wc", argv);
} else {
  close(p[0]);
  write(p[1], "hello world\n", 12);
  close(p[1]);
}
```




```
% echo hello world | wc
```