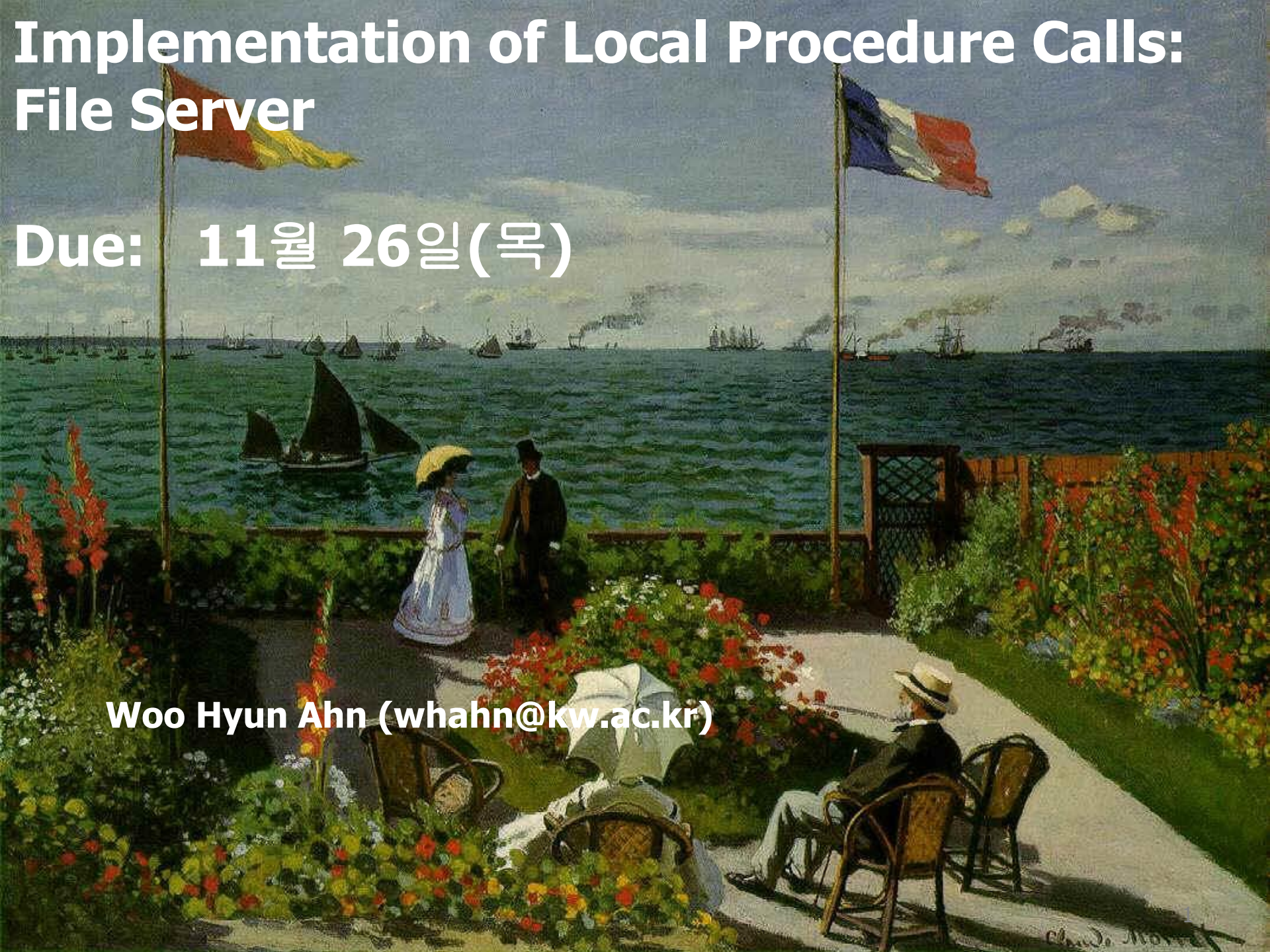
# Implementation of Local Procedure Calls: File Server

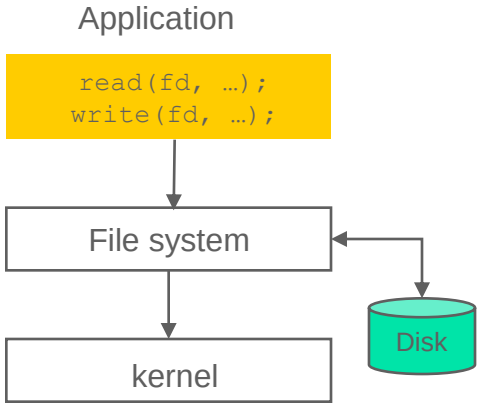
**Due: 11월 26일(목)**

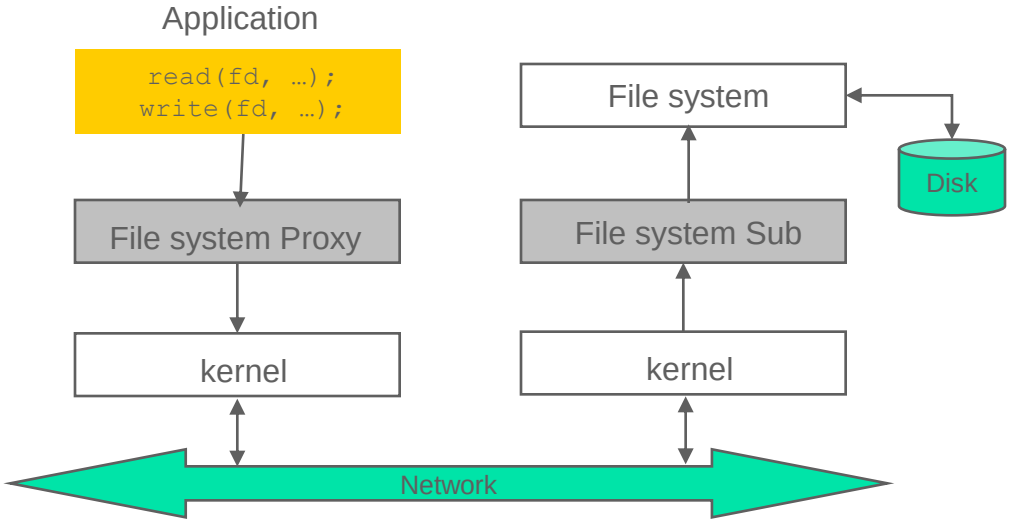


Woo Hyun Ahn (whahn@kw.ac.kr)

# Introduction

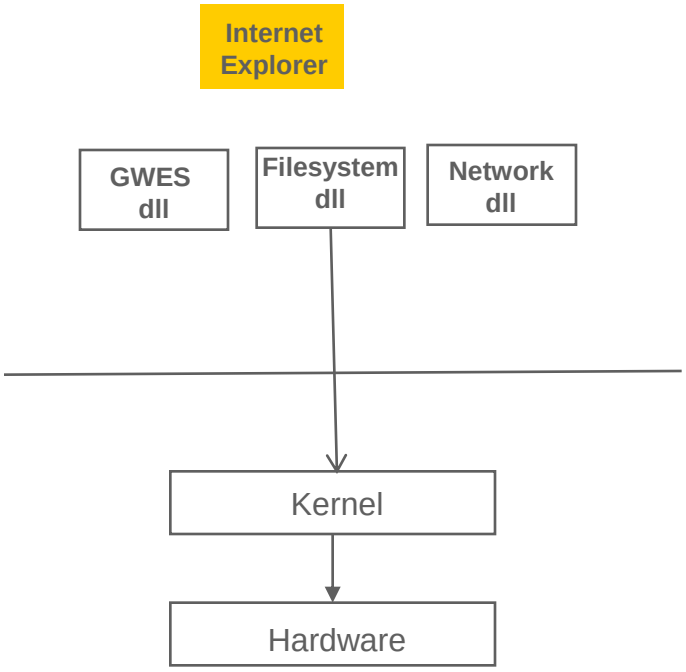
- Remote procedure call (RPC)
  - The primary goal of RPC is to provide clients with services on remote computers as if they are serviced by local procedure calls.
  - RPC server offers services to external systems as remotely callable procedures. A remote RPC client can invoke these procedures.



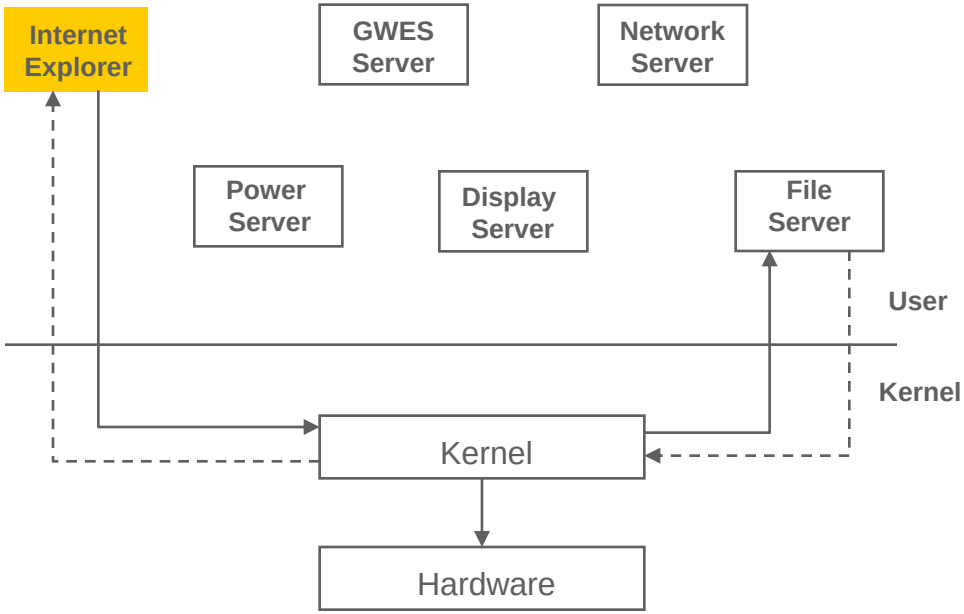



# Introduction (Cont'd)

- Local Procedure Call
  - RPC와 동일하나 원격이 아닌 동일 컴퓨터 내에서 동작이 이루어짐
  - 대표적인 사용 사례, 안드로이드, 윈도우 운영체제

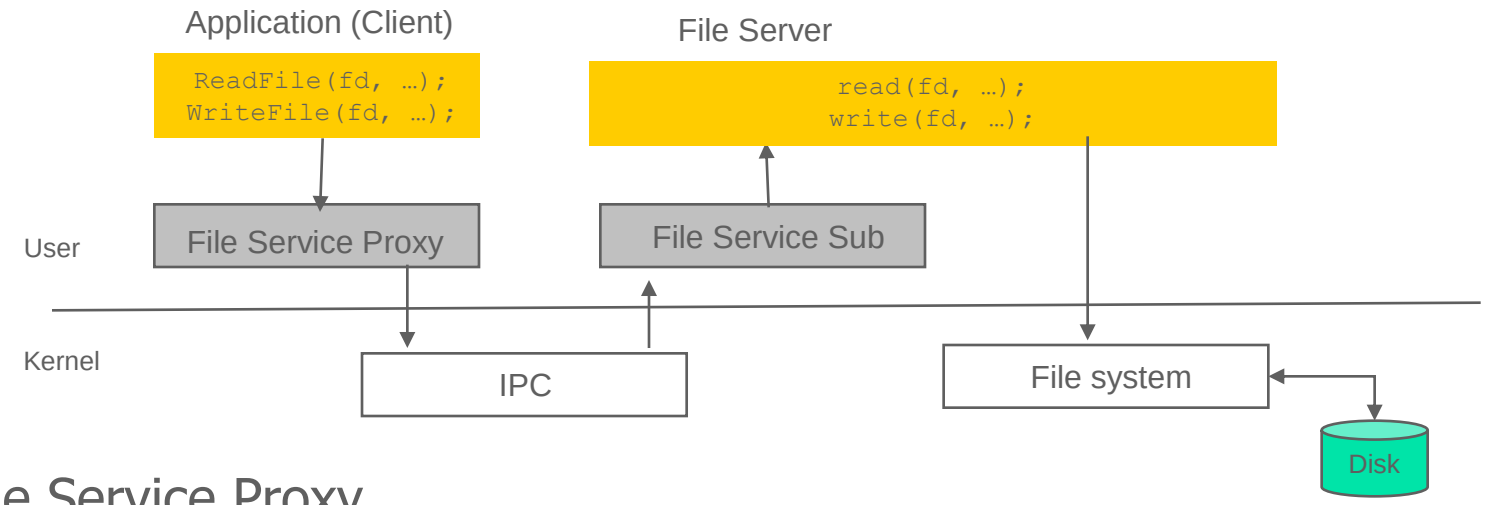



(a) Layered approach




(b) client-server approach via LPC

# File Server




- File Service Proxy
  - Client가 호출한 함수(예, ReadFile)의 정보 (함수 번호, 함수 인자들)을 묶어서(marshalling) File server로 송신하고, File server의 처리 결과를 기다림
  - File server 에서 처리 결과가 전달되면 해당 함수는 그 결과 값을 반환
- File Service Stub
  - Client 에서 전달된 함수 정보를 분해(unmarshalling)하고, 해당 함수에 대응하는 시스템 함수("read" system call)를 호출
  - 시스템 함수를 호출한 결과를 묶어서 Client 로 전달함

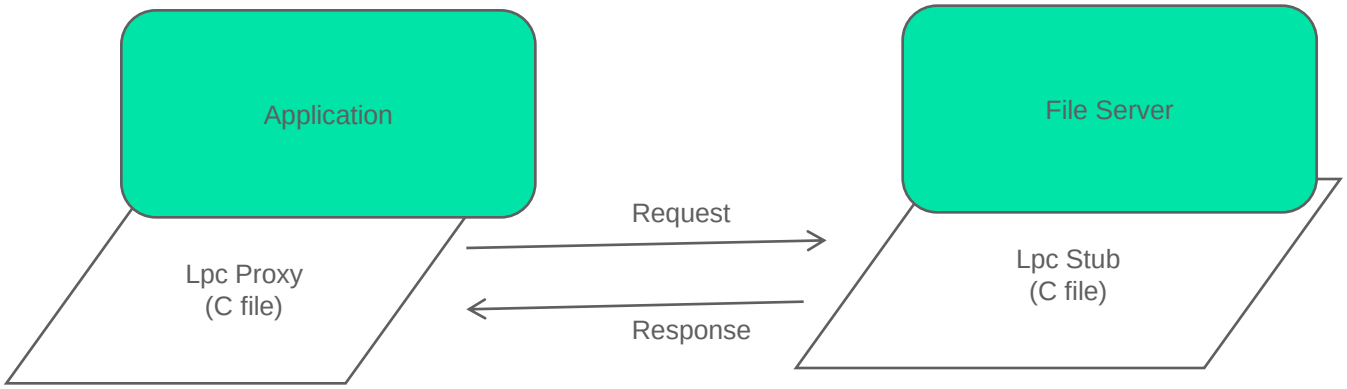
# File Server가 호출하는 System call

- **File server**는 **client**가 요청하는 함수에 대응하여 아래에 정의된 **system call**을 호출하면 된다

- **int open(const char *_path_, const char *_mode_);**
- int read(int fd, void* pBuf, int size);
- int write(int fd, void* pBuf, int size);
- int close(int fd);
- int mkdir(char* path)
- int rmdir(char* path)
- Char* gets(char* str);

- 이슈**:** 어떤 **system call**을 호출할 것인지 어떻게 결정할까**?**
  - File proxy가 전달하는 service 요청 번호에 대응하는 system call을 호출

# Runtime Situation



# File LPC Proxy

- LpcProxy 와 LpcStub 간의 전성되는 메시지 형식

```
#define LPC_DATA_MAX    512
#define LPC_ARG_MAX     4

typedef struct __LpcRequest
{
    long pid;               // message type
    lpcService service;
    int numArg;
    LpcArg lpcArgs[LPC_ARG_MAX];
} LpcRequest; //  request format

typedef struct __LpcArg
{
    int argSize;
    char argData[LPC_DATA_MAX];
} LpcArg; // data format


Typedef struct __lpcResponse
{
    long pid; // message type
    int errorno;
    int responseSize;
    char responseData[LPC_DATA_MAX];
} LpcResponse; // response format
```

```
Typedef enum __lpcService
{
    LPC_OPEN_FILE,
    LPC_READ_FILE,
    LPC_WRITE_FILE,
    LPC_CLOSE_FILE,
    LPC_MAKE_DIRECTORY,
    LPC_REMOVE_DIRECTORY,
} LpcService
```

# File LPC Proxy

- 구현해야 할 함수

Testcase.c

```
Void main(void)
{
   Init();

   MakeFile("tmp");
   int fd = OpenFile("tmp/a.c", O_CREATE);
   ...
   WriteFile(fd, pBuf, size);
   ...
   ReadFile(fd, pBuf, size);
   ...
   CloseFile(fd);
   ...

   ...

}
```

LpcProxy.c

```
Int OpenFile(char* path, int mode)
{
}

Int ReadFile(int fd, void* pBuf, int size)
{
}

Int WriteFile(int fd, void* pBuf, int size)
{
}

Int CloseFile(int fd)
{
}

Int MakeFile(char* path)
{
}

Int RemoveFile(char* path)
{
}
```

- 여러분에 File open, read, write 조합의 테스트 프로그램을 제공
- LpcProxy.c를 구현해야 함
  - LpcStub로 request 메시지를 보내고, response를 받는 코드를 구현함.

# Client Design

```
int main(void)
{
    Init();
    ...
    int fd;
    fd = OpenFile("tmp/a.c", O_CREATE);
    …
}
```

main.c
(테스트 파일)

```
Void Init(void)
{
    ...
    ...
}
int OpenFile(char* path, int mode)
{




    ...
    ...
    ...



}
```

LpcProxy.c
(구현해야 할 파일)

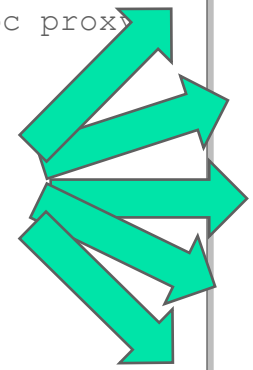
# File Server

(구현해야 할 파일)

FileServer.c

```
void main(void)
{
    Init();
    while(1) {
        receive Lpc request from Lpc proxy
        get Lpc service command
        get Lpc function arguments
        call Lpc service function
            implemented in LpcStub.c

    }
}
```



LpcStub.c

```
Void Init(void)
{
}
Int OpenFile(char* path, int mode)
{
}

Int ReadFile(int fd, void* pBuf, int size)
{
}

Int WriteFile(int fd, void* pBuf, int size)
{
}

Int CloseFile(int fd)
{
}

Int MakeFile(char* path)
{
}

Int RemoveFile(char* path)

Int GetString(char* string)
```

(구현해야 할 파일)

- File server는 무한루프로 실행.
  - File proxy에서 전달된 request 메시지를 기다린다.
  - Request가 전달되면 unmarshalling.
  - 요청한 Lpc service를 확인한다. (feat Lpc service number)
  - 해당 Lpc service에 대응하는 file stub의 함수를 호출함.
  - 호출 결과를 response로 proxy로 전달

# Server Design

LpcStub.c

```
Void Init(void)
{
   ...
   ...
}
int OpenFile
{


   ...
   ...


}
```

FileServer.c

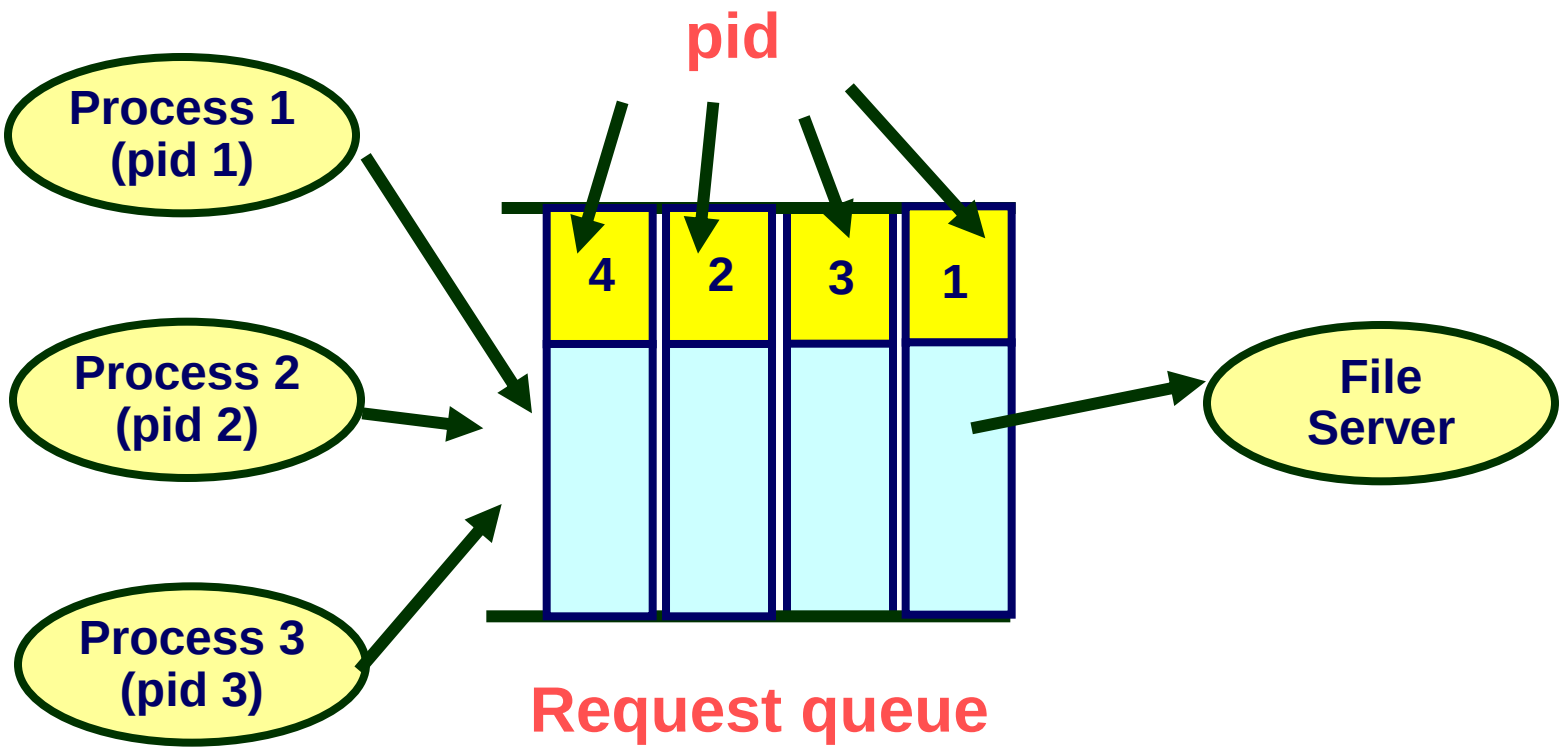
```
void main(void)
{
    Init();

    while (1)
    {
        …
    }
}
```

# File Server 초기화

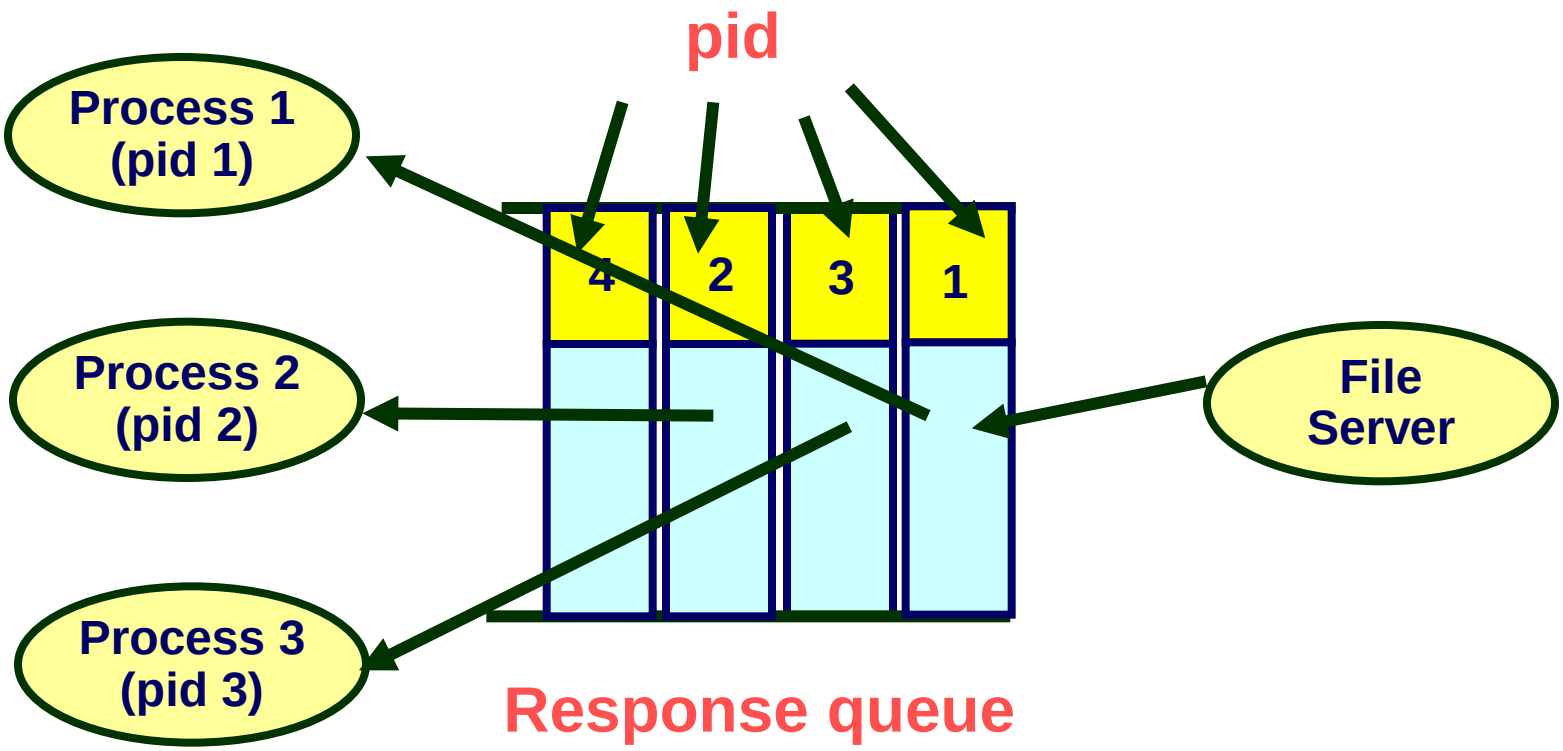
- Client와 File server 간의 통신 방법
  - File server의 Init 함수에서 Message queue를 생성

- Client에서 File server로 request를 전달 용 message queue 생성
  - Request를 전달할 때 client 자신의 pid를 LpcRequest message에 담아서 전달
  - File server는 message queue에서 FIFO 순서로 message를 받음



# File Server 초기화

- File server에서 client로 Response를 전달하기 위한 Message queue 생성
  - File server의 처리 결과를 response를 담아서 response queue에 전달
  - Response를 원하는 client의 pid를 LpcResponse에 담아서 전달함
  - Client는 자신의 pid로 msgrecv를 호출함



# OpenFile 구현: LpcProxy Code

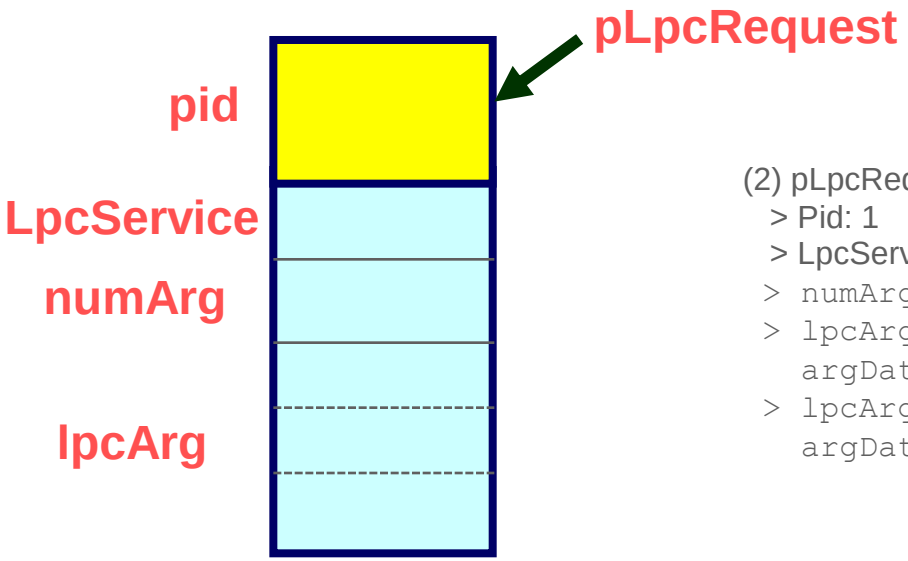
- LpcProxy Code 구현
  - OpenFile로 전달된 argment를 LpcRequest에 담아서 message queue에 전달
  - Request를 전달한 후 response를 기다린다

**Client (pid 1)**

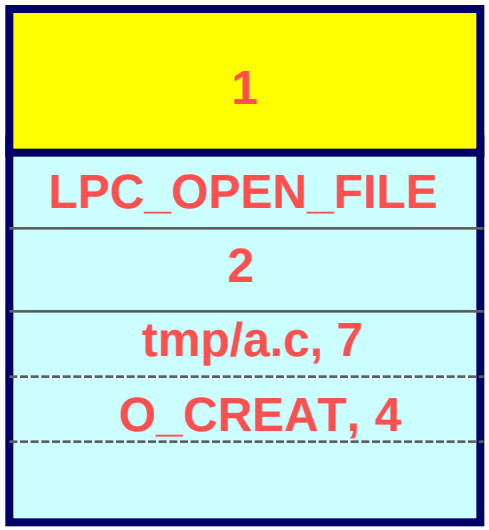
```
OpenFile("tmp\a.c", O_CREAT)
```

---

(1) LpcRequest를 위한 메모리 공간을 할당




(2) pLpcRequest를 채운다
- > Pid: 1
- > LpcService: `LPC_OPEN_FILE`
- > `numArg: 2`
- > `lpcArg[0]:`
  `argData: "tmp/a.c",size: 7`
- > `lpcArg[1]:`
  `argData: O_CREATE, size: 4`



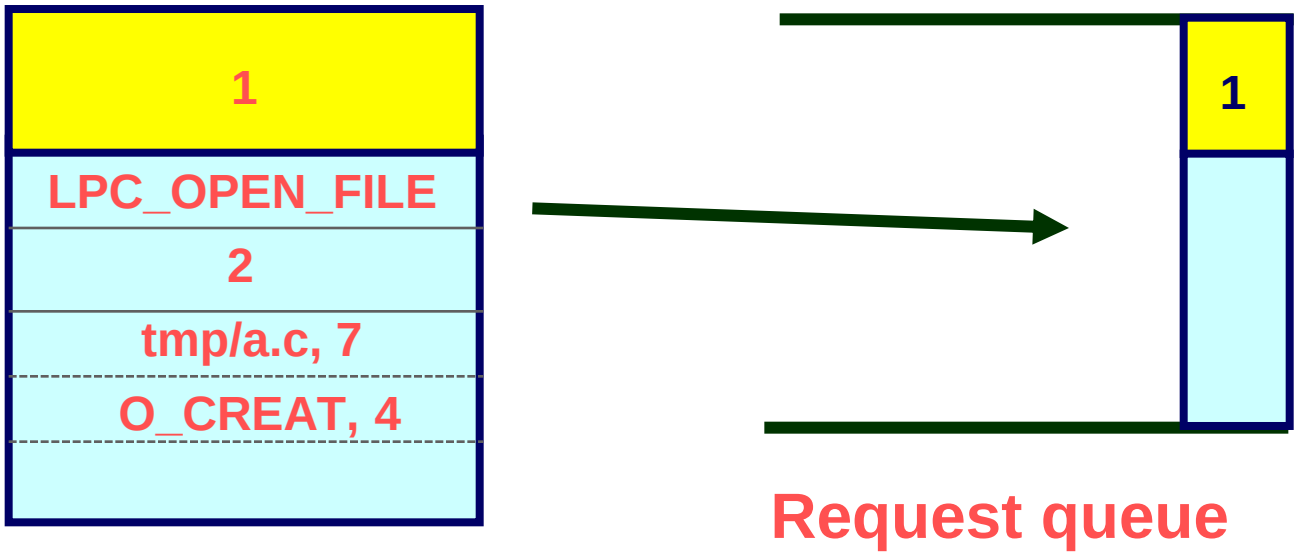
# OpenFile 구현: LpcProxy Code

Client (pid 1)

```
OpenFile("tmp\a.c", O_CREAT)
```

---

(3) pLpc를 msgsnd로 request queue에 보냄



# OpenFile 구현: LpcProxy Code

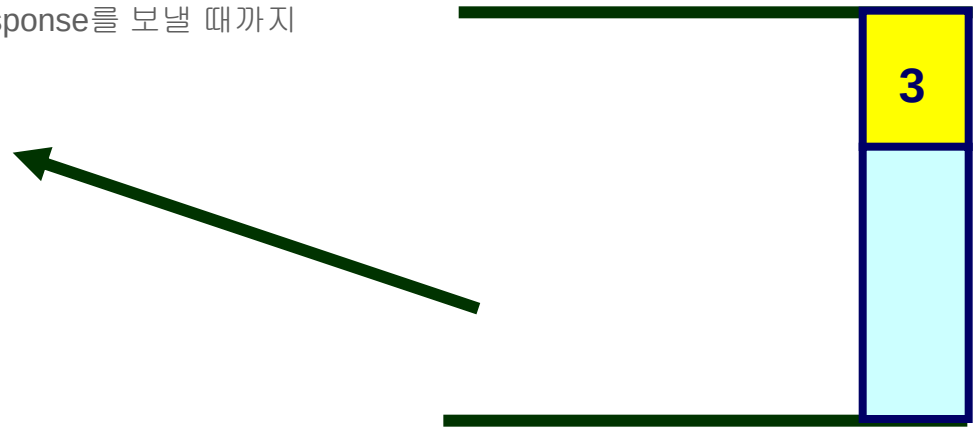
- LpcProxy Code 구현
  - OpenFile로 전달된 argment를 LpcRequest에 담아서 message queue에 전달
  - Request를 전달한 후 response를 기다린다

**Client (pid 1)**

```
OpenFile("tmp\a.c", O_CREAT)
```

---

(4) File server가 LpcResponse를 보낼 때까지 기다린다




**Response queue**

# OpenFile 구현: File Server

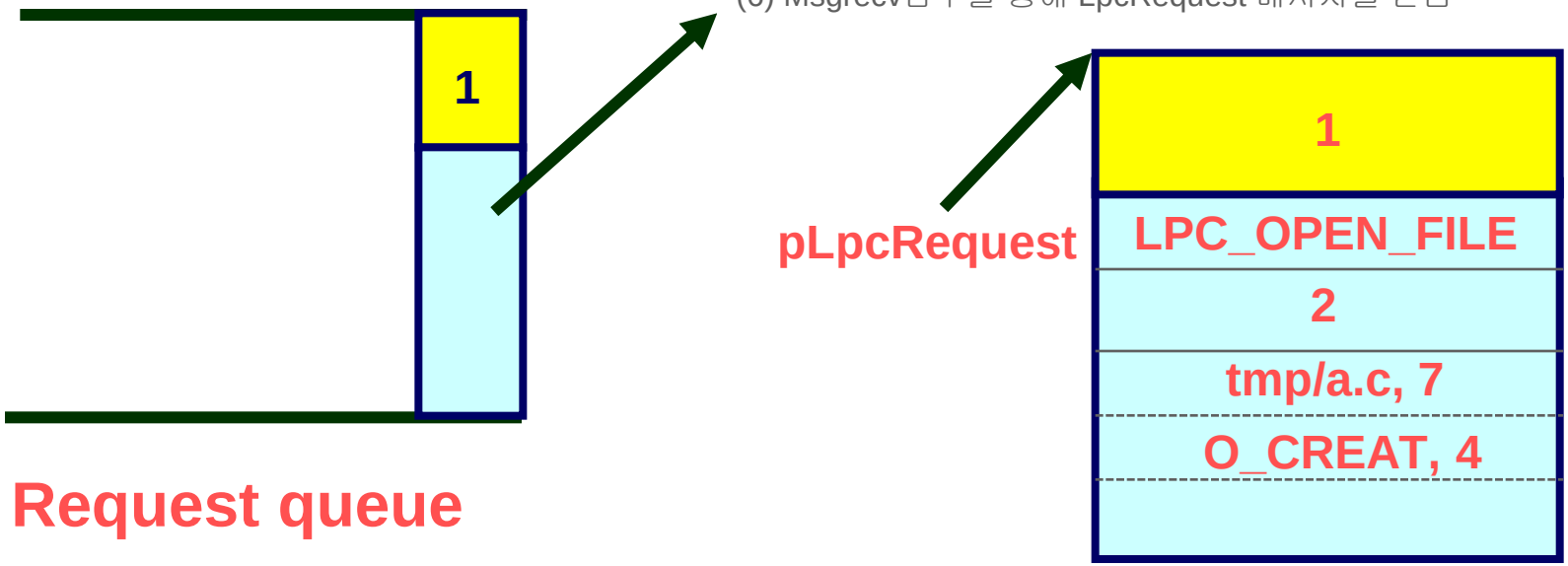
- File server Code 구현
  - File server는 request queue에서 LpcRequest를 수신할 때까지 기다림

**File server**

```
Main() {
    ...
    msgrecv(...);
    ...
}
```

(5) pLpcRequest의 메모리 공간을 할당

(6) Msgrecv함수를 통해 LpcRequest 메시지를 받음

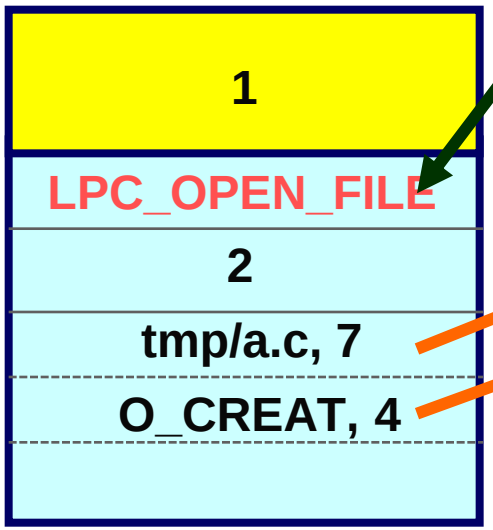


# OpenFile 구현: File Server

- LpcStub code구현
  - LpcService code에 대응하는 system call을 호출
  - LPC_OPEN_FILEL → open, LPC_READ_FILE→ read,
  - LPC_MAKE_DIRECTORY → mkdir 등등

---

(7) LpcServer가 LPC_OPEN_FILE임을 확인하고, LpcStub의 OpenFile 함수를 호출함



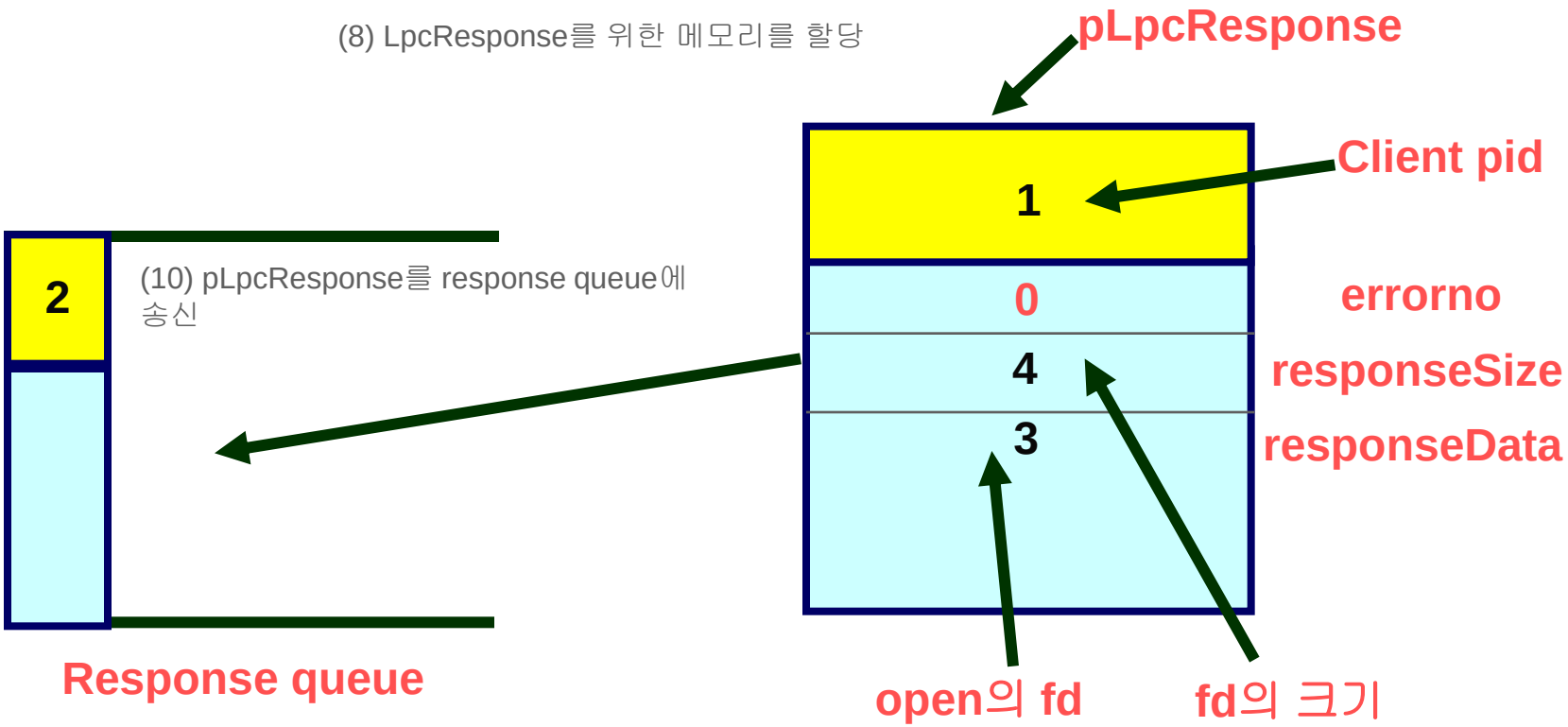

(8) OpenFile 함수는 open() 시스템 콜을 호출

```
Int OpenFile  (name  ,   flag  ) {
   ...
   fd = open(name, flag)
   ...
}
```

(9) Open 함수로 전달될 2개의 argumen를 LpcRequest에서 획득함

# OpenFile 구현: File Server

- LpcStub code구현
  - Open 시스템 콜 호출 결과를 Client로 전달

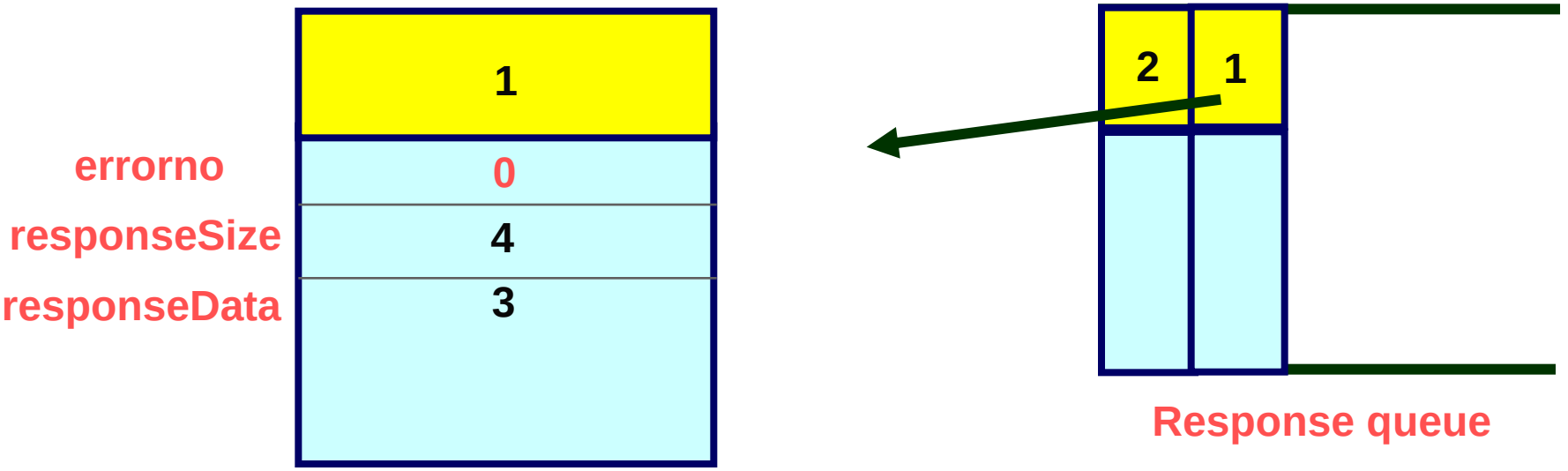


# OpenFile 구현: Proxy code

- LpcResponse로 전달된 결과를 OpenFile를 통해 반환
  - LpcReponse의 메시지를 분해해서 fd 값을 반환

```
OpenFile("tmp\a.c", O_CREAT)
```

(11) Response queue에서 LpcResponse를 수신




(12) LpcResponse에서 fd 값을 얻어서 OpenFile 함수의 반환 값으로 전달

# ReadFile 구현: LpcProxy Code

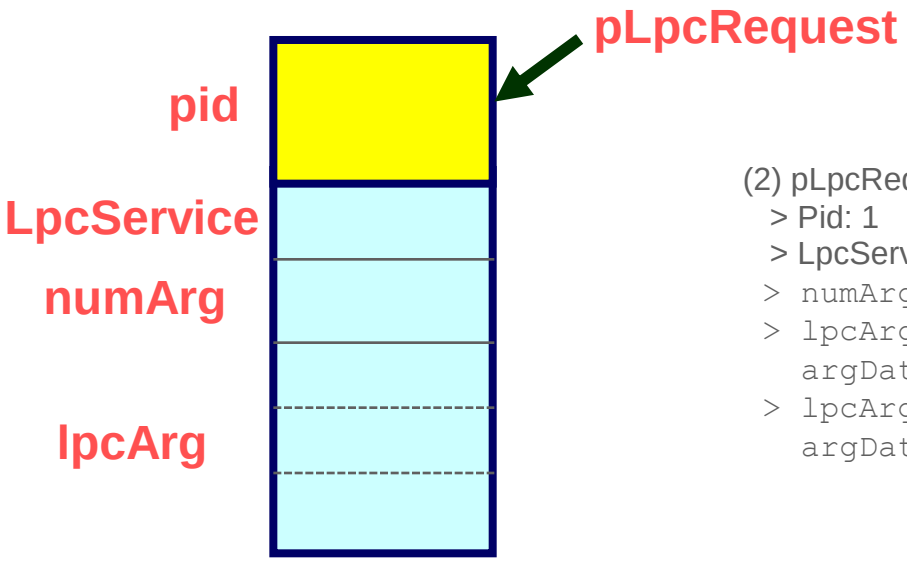
- LpcProxy Code 구현
  - fd와 size만 LpcRequest에 담아서 송신.
  - pBuf는 FileServer에서 전달된 데이터를 저장하는 메모리

**Client (pid 1)**

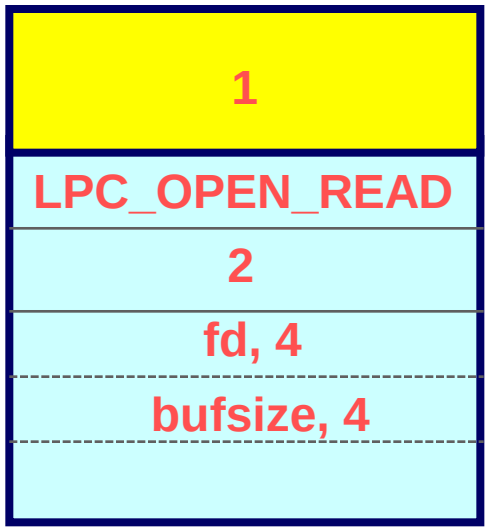
```
ReadFile(fd, pBuf, bufsize)
```

---

(1) LpcRequest를 위한 메모리 공간을 할당

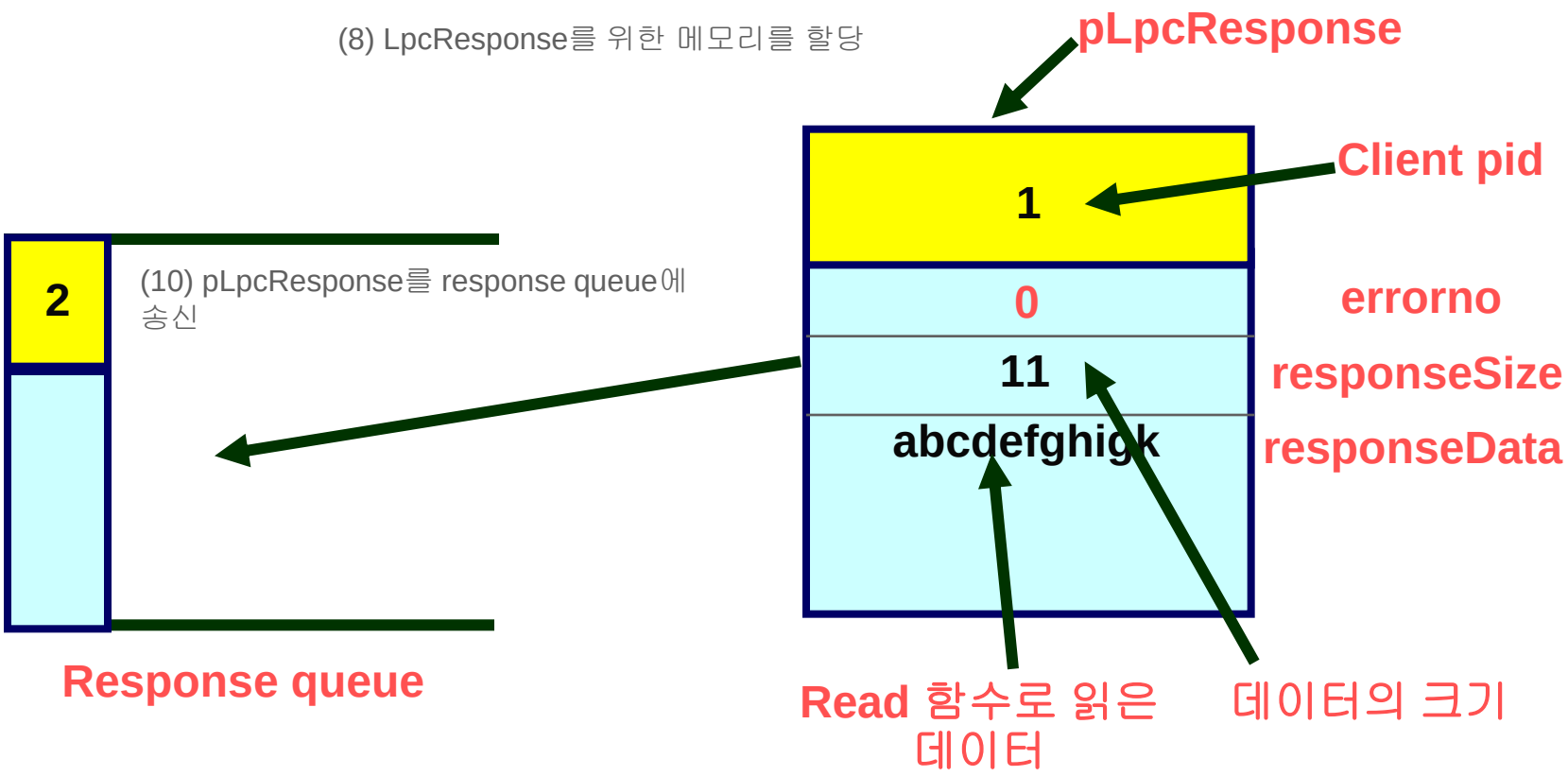



(2) pLpcRequest를 채운다
- > Pid: 1
- > LpcService: `LPC_READ_FILE`
- > `numArg: 2`
- > `lpcArg[0]:`
  `argData: fd, size: 4`
- > `lpcArg[1]:`
  `argData: bufsize, size: 4`



# ReadFile 구현: File Server

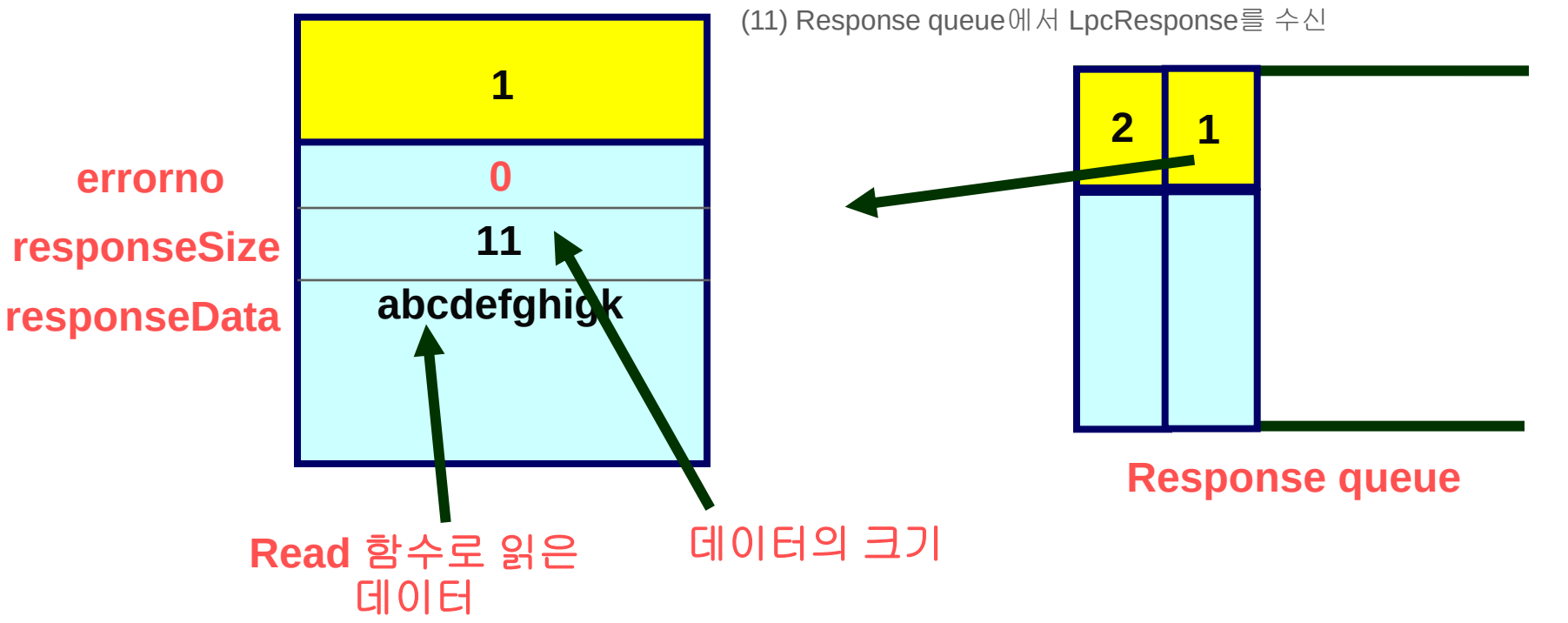
- LpcStub code구현
  - read 시스템 콜 호출 결과를 Client로 전달
  - LpcReponse의 responseData에 read 함수로 읽은 데이터를 저장함



# OpenFile 구현: Proxy code

- LpcResponse로 전달된 결과를 OpenFile를 통해 반환
  - LpcReponse의 메시지를 분해해서 fd 값을 반환

```
ReadFile(fd, O_CREAT)
```




(12) LpcResponse의 responseData는 pBuf에 복사하고, responseSize는 리턴 값으로 전달

# 컴파일

- Main.c에 동작 테스트
- 11월 17일까지 testcase.c를 배포. Main.c 대신에 테스트
- Client process는 2~3개 생성하여 테스트할 것