8. NIO 네트워킹 프로그래밍

1. 파일 비동기 채널
2. TCP 블로킹 채널
3. TCP 넌블로킹 채널
4. TCP 비동기 채널
5. UDP 채널

1. 파일 비동기 채널

❖ FileChannel의 단점

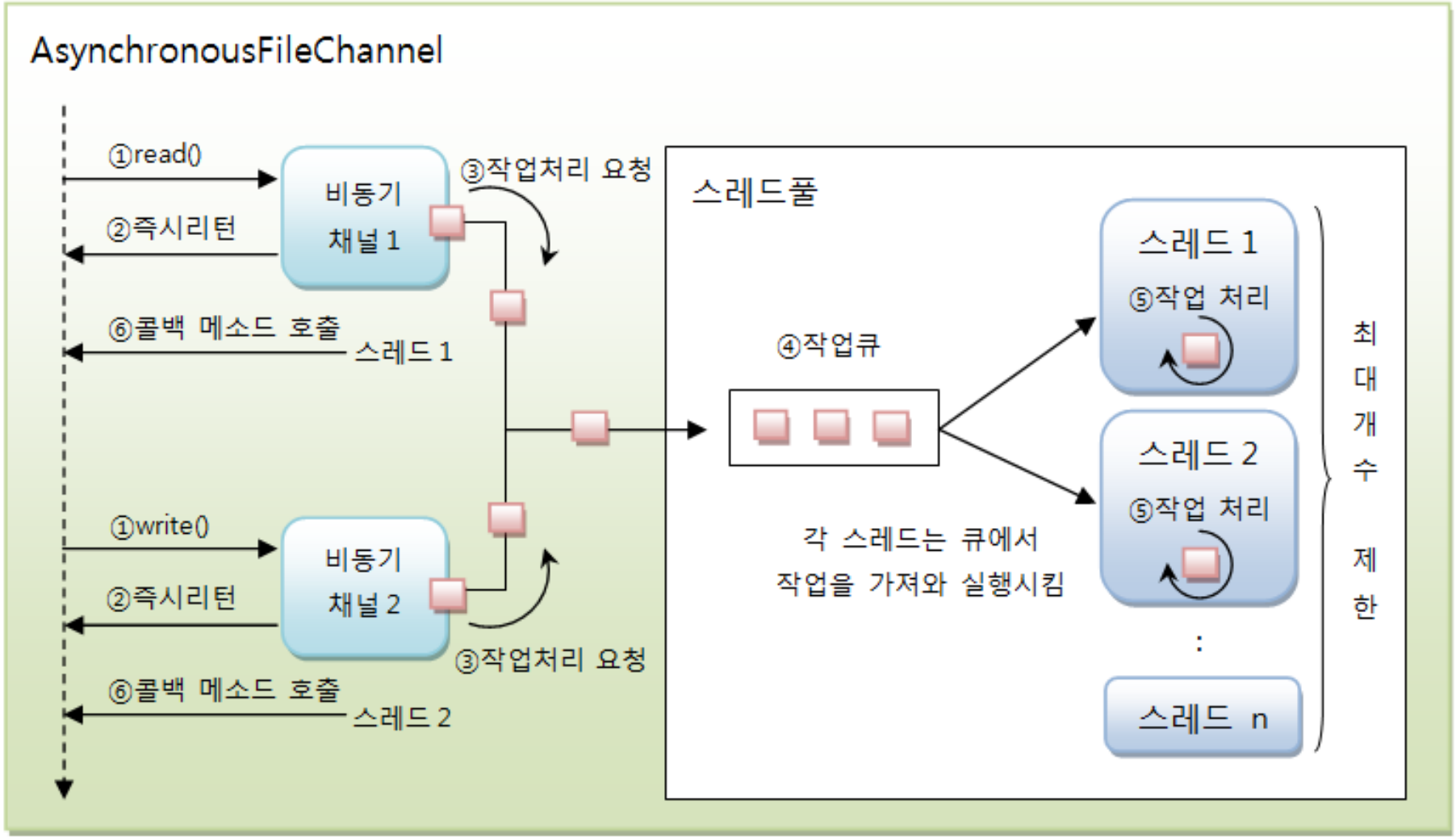
- read()와 write() 메소드는 작업하는 동안 블로킹
  - 블로킹 동안에 UI 갱신이나 이벤트 처리를 할 수 없음
  - 따라서 별도의 작업 스레드를 생성해서 이들 메소드를 호출해야
  - 동시에 처리해야 할 파일 수가 많다면 스레드 수 증가로 문제 유발 가능

❖ AsynchronousFileChannel

- read()와 write() 메소드는 즉시 리턴
  - 이들 메소드는 스레드풀에게 작업 처리를 요청하고 즉시 리턴
  - 작업 스레드가 파일 입출력 완료 - 콜백(callback) 메소드 자동 호출
  - 불특정 다수의 파일 및 대용량 파일의 입출력 작업 시 유리

1. 파일 비동기 채널

❖ AsynchronousFileChannel의 동작



1. 파일 비동기 채널

❖ AsynchronousFileChannel 생성과 닫기

- 생성 – 정적 메소드 open() 사용

```
AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(
                                path,
                                EnumSet.of(StandardOpenOption.READ),
); // 내부적으로 생성되는 기본 스레드 풀 사용
```

```
AsynchronousFileChannel fileChannel = AsynchronousFileChannel.open(
                                path,
                                EnumSet.of(StandardOpenOption.READ),
                                executorService //사용자 정의 스레드 풀 사용
);
```

- 스레드 풀 생성

```
ExecutorService executorService = Executors.newFixedThreadPool(
                                        Runtime.getRuntime().availableProcessors()
                                );
```

- 닫기 – 채널을 더 이상 쓰지 않을 때

```
fileChannel.close();
```

1. 파일 비동기 채널

❖ 파일 읽기와 쓰기

```
read(ByteBuffer dst, long position, A attachment, CompletionHandler<Integer A> handler);
```

```
write(ByteBuffer src, long position, A attachment, CompletionHandler<Integer A> handler);
```

- 매개변수
  - dst, src: 읽거나 쓰기 위한 ByteBuffer
  - position: 파일에서 읽을 위치이거나 쓸 위치
  - attachment: 콜백 메소드로 전달할 첨부 객체
  - handler: CompletionHandler<Integer, A> 구현 객체
- CompletionHandler<Integer, A>
  - Integer: 입출력 작업 처리 후 결과 타입 (결과값은 읽거나 쓴 바이트 수) (고정)
  - A: 첨부 객체 타입으로 첨부 객체가 필요 없다면 Void 지정(개발자 지정)
  - 콜백 메소드

리턴타입	메소드명(매개변수)	설명
void	completed(integer result, A attachment)	작업이 정상적으로 완료된 경우 콜백
void	faild(Throwable exc, A attachment)	에외 때문에 작업이 실패된 경우 콜백

- CompletionHandler 구현 클래스

```
CompletionHandler<Integer, Attachment>completionHandler=
new CompletionHandler<Integer, Attachment>(){

        @Override
        public void completed(Integer result, Attachment attachment) {
        }

        @Override
        public void failed(Throwable exc, Attachment attachment) {
        }

};
```

2. TCP 블로킹 채널

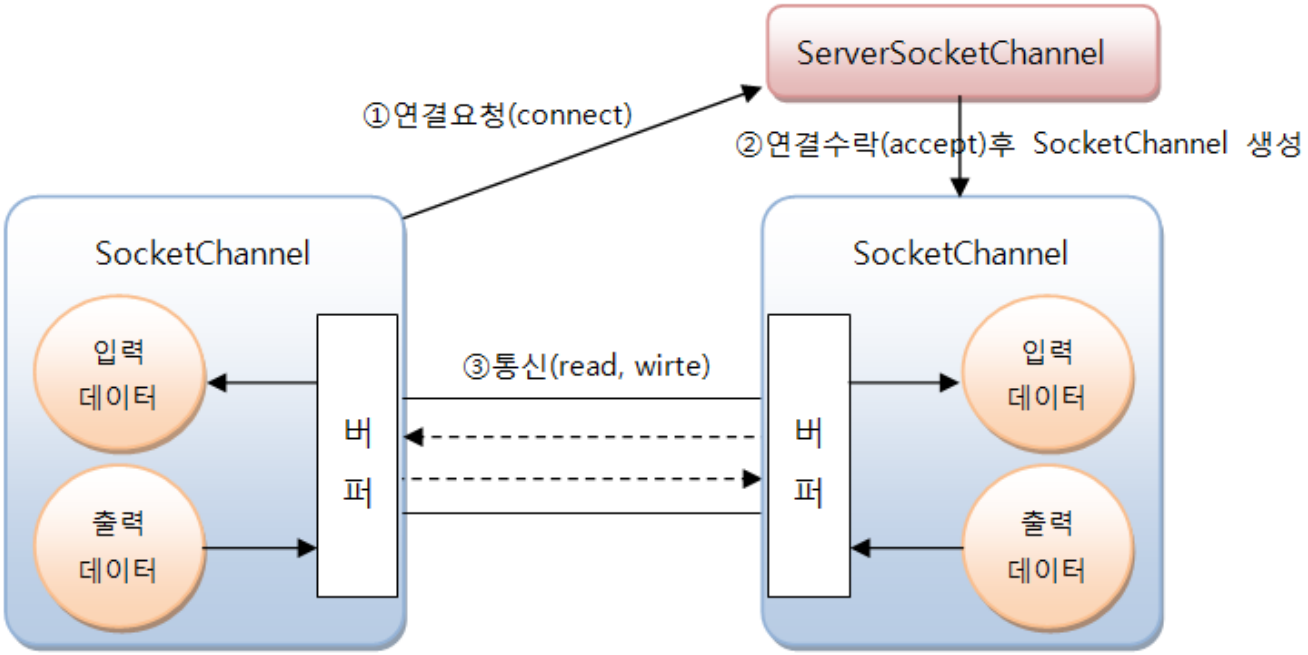
❖ TCP 서버/클라이언트 세가지 구현 방식

- 블로킹
  - 연결요청, 연결수락, 입출력 작업 시 블로킹
- 넌블로킹
  - 연결요청, 연결수락, 입출력 작업 시 넌블로킹
  - 작업 처리 준비된 것만 셀렉터가 선택해서 처리하는 방식
- 비동기
  - 연결요청, 연결수락, 입출력 작업 시 넌블로킹
  - 스레드풀에서 처리 후 콜백 메소드 호출

2. TCP 블로킹 채널

❖ 서버소켓 채널과 소켓채널의 용도

- ServerSocketChannel
- SocketChannel



2. TCP 블로킹 채널

❖ 서버소켓 채널 생성과 연결 수락

- ServerSocketChannel 생성

```
ServerSocketChannel serverSocketChannel = ServerSocketChannel.open();
serverSocketChannel.configureBlocking(true);
serverSocketChannel.bind(new InetSocketAddress(5001));
```

  - IP 와 포트 정보를 리턴해 주는 메소드

리턴 타입	메소드명(매개 변수)	설명
String	getHostName( )	클라이언트 IP 리턴
int	getPort( )	클라이언트 포트 번호 리턴
String	toString( )	"IP:포트번호" 형태의 문자열 리턴

- 연결 수락

```
SocketChannel socketChannel = serverSocketChannel.accept();
```

- 닫기

```
serverSocketChannel.close();
```

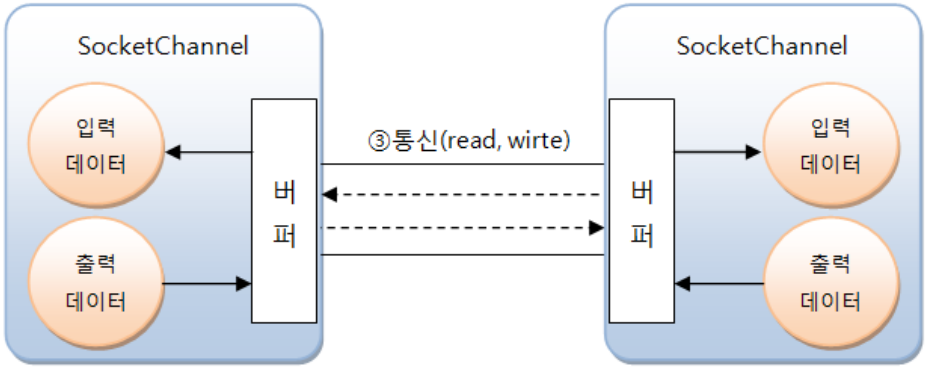
2. TCP 블로킹 채널

❖ 소켓 채널 생성과 연결 요청

- SocketChannel 생성과 연결 요청
  - 클라이언트가 서버에 연결 요청할 때 쓰이는 소켓
- 닫기
  - 클라이언트가 종료되거나, 필요에 따라 연결 끊을 때 Close()
- 서버가 열려있어야 클라이언트 통신 가능

2. TCP 블로킹 채널

❖ 소켓 채널 데이터 통신



- read()가 블로킹이 해제 + 리턴 되는 경우

블로킹이 해제되는 경우	리턴값
상대방이 데이터를 보냄	읽은 바이트 수
상대방이 정상적으로 SocketChannel 의 close()를 호출	-1
상대방이 비정상적으로 종료	IOException 발생

2. TCP 블로킹 채널

❖ 소켓 채널 데이터 통신

- write() 데이터 보내기

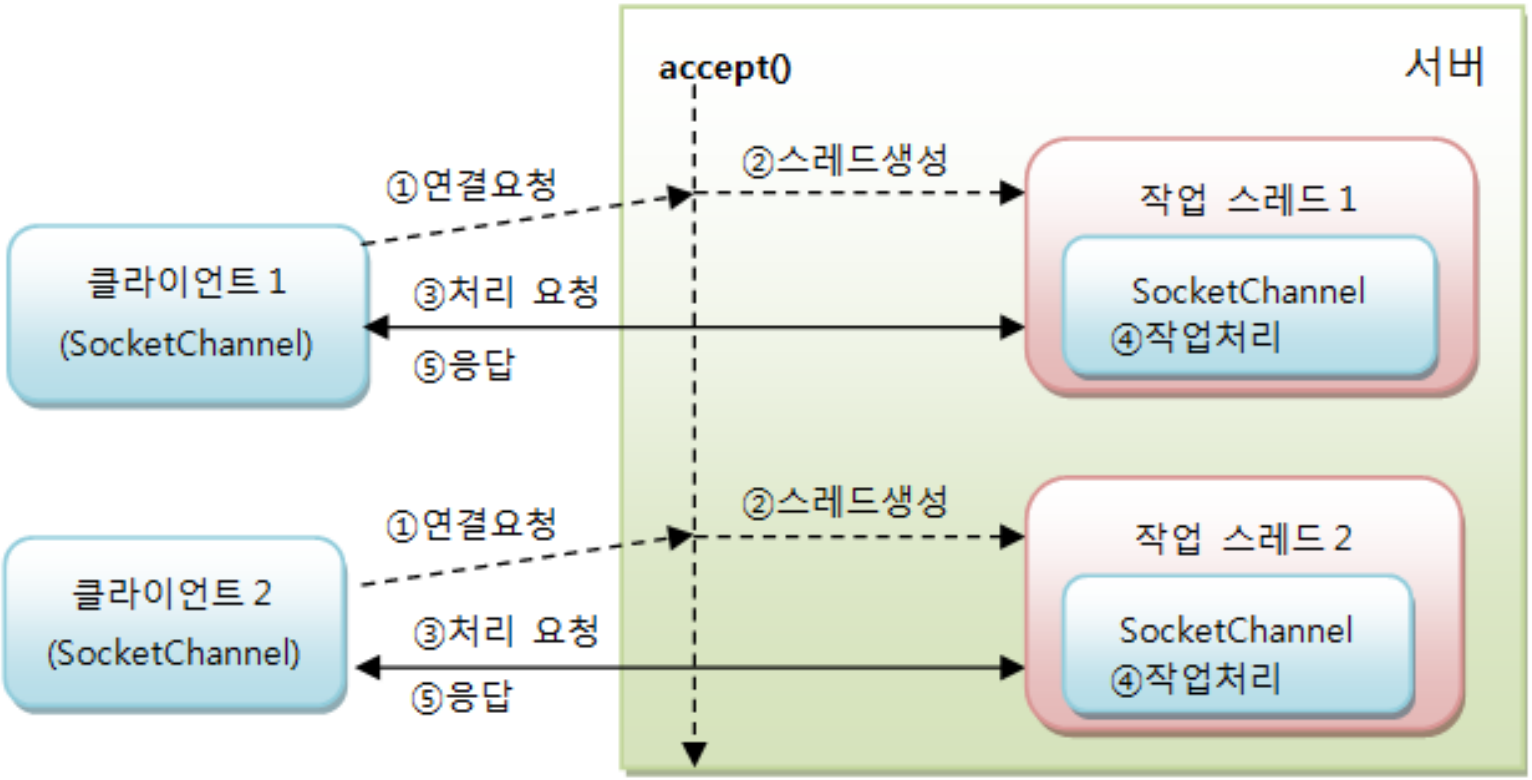
```
Charset charset = Charset.forName("UTF-8");
Bytebuffer byteBuffer = charset.encode("Hello Server");
socketChannel.write(byteBuffer);
```

- read() 데이터 수신

```
Charset charset = Charset.forName("UTF-8");
ByteBuffer byteBuffer = ByteBuffer.allocate(100);
int byteCount = socketChannel.read(byteBuffer);
```

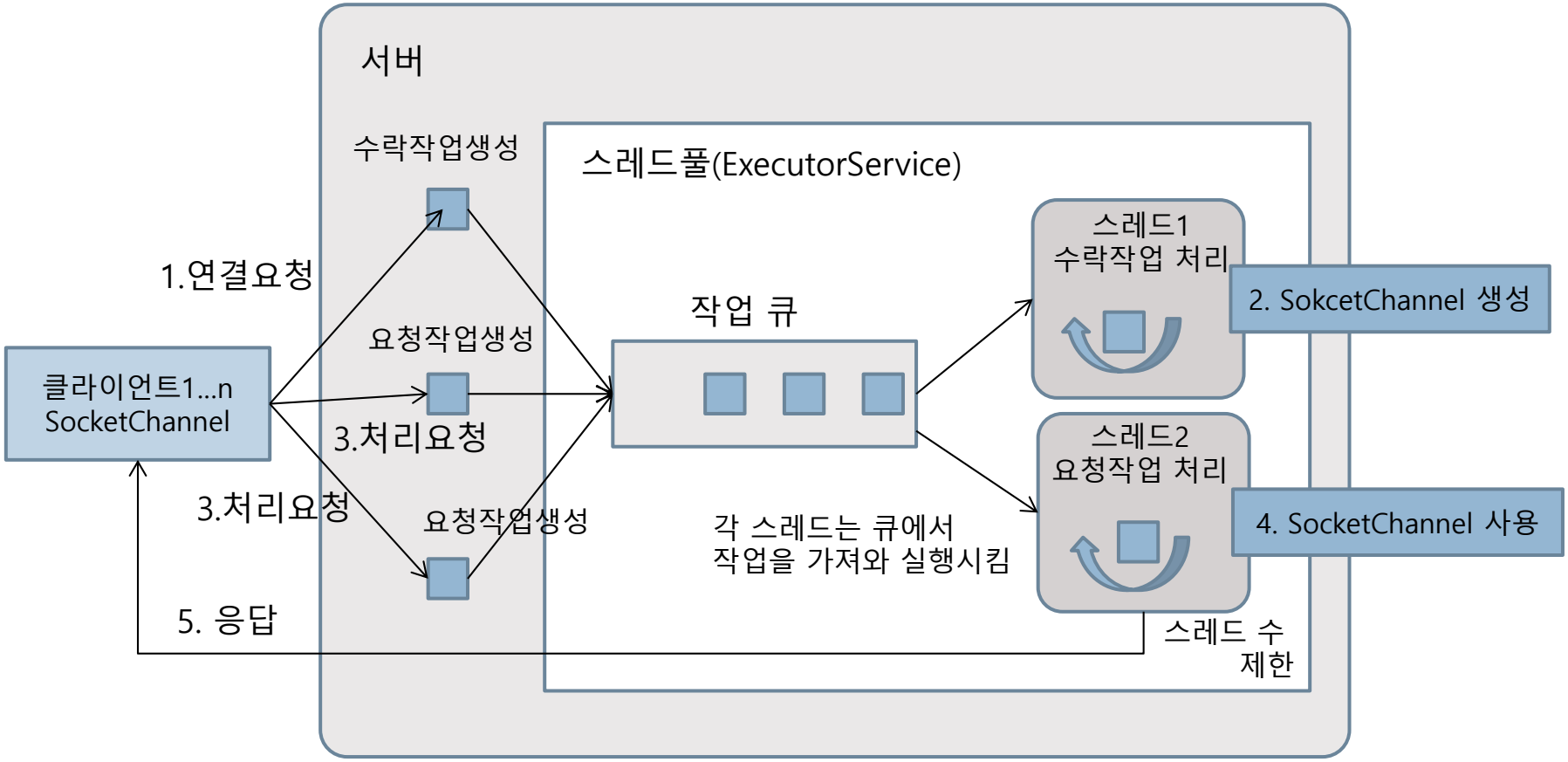
2. TCP 블로킹 채널

❖ 스레드 병렬 처리



2. TCP 블로킹 채널

❖ 스레드풀 이용한 서버 구현 방식



2. TCP 블로킹 채널

- **채팅 서버 및 클라이언트 구현**
- **채팅 서버의 경우 스레드 풀, 서버 소켓채널, 소켓채널의 동작 확인**
  - 채팅 클라이언트의 경우 소켓채널의 동작을 유심히 확인

- **블로킹과 인터럽트**
  - IO 소켓에서는 입출력 스트림에서 작업스레드가 블로킹 된 경우
    - 다른 스레드가 작업 스레드의 인터럽트 메소드 호출해도 블로킹이 풀리지 않음
  - NIO 소켓 채널의 경우
    - 인터럽트 만으로도 소켓채널이 닫히면서 블로킹 풀림

3. TCP 넌블로킹 채널

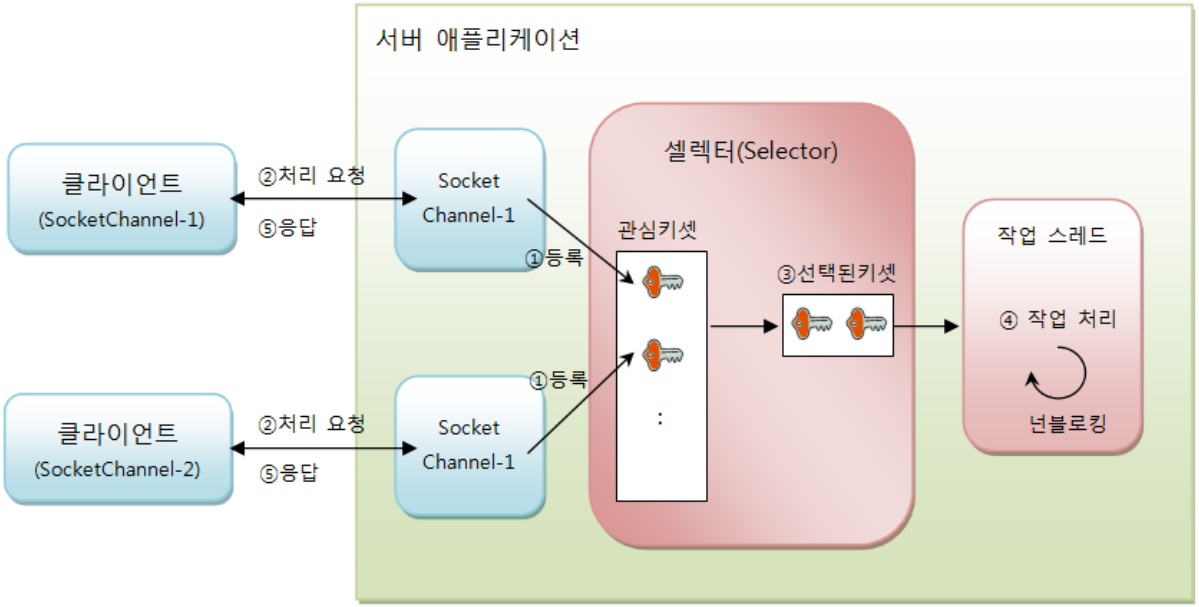
❖ 넌블로킹(non-blocking) 방식의 특징

- connect(), accept(), read(), write() 메소드는 블로킹 없이 즉시 리턴
  - 작업 처리 준비가 된 상태에서 메소드 실행할 것
  - 작업 처리 준비가 된 채널만 선택해 처리
- 셀렉터가 작업 처리 준비된 채널 선택
  - 넌블로킹 채널은 이벤트 리스너 역할 하는 셀렉터(Selector) 사용
  - 채널이 작업 처리 필요할 경우 셀렉터에 통보
  - 셀렉터는 통보한 채널 선택
- 멀티 채널 작업을 싱글 스레드에서 처리 가능
  - 작업 스레드가 블로킹되지 않음
    - 셀렉터가 선택한 채널들을 싱글 스레드에서 모두 처리 가능
  - 스레드 풀 사용할 경우, 적은 수의 스레드로 많은 양의 작업 처리

3. TCP 넌블로킹 채널

❖ 셀렉터(Selector)의 동작 원리

- 채널은 자신의 작업 유형을 키(SelectionKey)로 생성
- 셀렉터의 관심 키셋(interest-set)에 키 등록
- 셀렉터는 작업 처리 준비가 된 키를 선택
- 선택된 키셋에 별도로 저장
- 작업 스레드는 선택된 키셋에서 키를 하나씩 꺼냄
  - 연관된 채널 작업 처리



3. TCP 넌블로킹 채널

❖ 셀렉터 생성과 등록

- 셀렉터 생성 – Open() 메소드 호출해 생성
  - Exception 발생 가능하므로 예외처리 필요
- 넌블로킹 채널 생성 – 하위 클래스도 넌블로킹이어야.
- 셀렉터 등록
  - 첫 번째 매개값은 Selector
  - 두 번째 매개값은 작업 유형별 SelectionKey의 상수

SelectionKey 의 상수	설명
OP_ACCEPT	ServerSocketChannel 의 연결을 수락 작업
OP_CONNECT	SocketChannel 의 서버 연결 작업
OP_READ	SocketChannel 의 데이터 읽기 작업
OP_WRITE	SocketChannel 의 데이터 쓰기 작업

3. TCP 넌블로킹 채널

❖ 선택된 키셋

- Selector의 select() 메소드
  - 관심 키셋의 SelectionKey로부터 작업 처리 준비가 되었다는 통보 올 때까지 블로킹
    - 최소한 하나의 SelectionKey로부터 작업 처리 준비가 되었다는 통보가 오면 리턴
    - 리턴값은 통보를 해온 SelectionKey의 수
- select() 메소드 종류

리턴타입	메소드명(매개변수)	설명
int	select()	최소한 하나의 채널이 작업 처리 준비가 될때까지 블로킹된다.
int	select(long timeout)	select()와 동일한데, 주어진 시간(밀리세컨)동안만 블로킹된다.
int	selectNow()	작업을 처리할 준비가 된 채널만 선택하고 즉시 리턴된다.

  - 주로 첫 번째 메소드를 많이 사용

3. TCP 넌블로킹 채널

❖ 선택된 키셋

- select()가 리턴 되는 경우
  - 최소한 하나의 채널이 작업 처리 준비가 되었다는 통보를 할 때
  - Selector의 wakeup() 메소드를 호출할 때
  - select()를 호출한 스레드가 인터럽트될 때
- SelectionKey 의 작업 유형 변경
  - Selector의 wakeup() 메소드 호출
  - 블로킹되어 있는 select() 즉시 리턴
  - 변경된 작업 유형을 감시하도록 select() 재실행
- 선택된 키셋 얻기
  - select() 메소드가 1 이상의 값을 리턴 할 경우
  - selectedKeys() 메소드로 작업 처리 준비된 SelectionKey들을 Set 컬렉션으로 얻음

3. TCP 넌블로킹 채널

❖ 작업 스레드에서 채널 작업 처리

- 선택된 키셋에서 SelectionKey를 하나씩 꺼내어 작업 유형별 채널 작업 처리
- SelectionKey가 어떤 작업 유형인지 알아내는 방법
  - 다음 메소드 중 어느 것이 true를 리턴하는가

리턴타입	메소드명(매개변수)	설명
boolean	isAcceptable()	작업 유형이 OP_ACCEPT 인 경우
boolean	isConnectable()	작업 유형이 OP_CONNECT 인 경우
boolean	isReadable()	작업 유형이 OP_READ 인 경우
boolean	isWritable()	작업 유형이 OP_WRITE 인 경우

- SelectionKey로 부터 채널 객체 얻기

```
ServerSocketChannel serverSocketChannel = (ServerSocketChannel) selectionKey.channel();
```

- 첨부 객체 저장과 얻기
  - SelectionKey에 첨부해두고, 사용
  - attach() 메소드는 객체 첨부
  - attachment() 메소드는 첨부된 객체를 얻을 때 사용

3. TCP 넌블로킹 채널

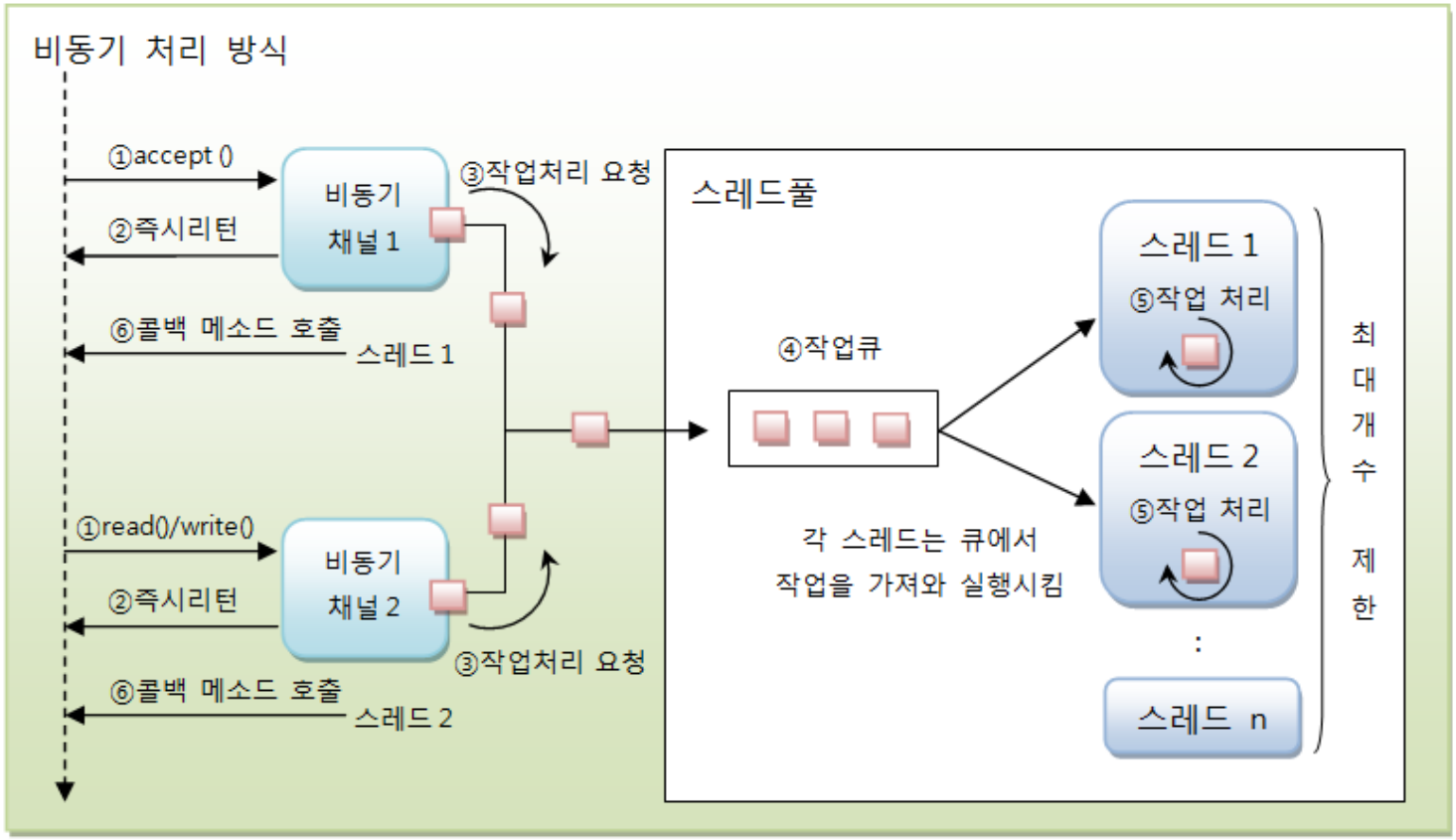
❖ 채팅 서버 및 클라이언트 구현 (p.1186~1195)

- 셀렉터와 넌블로킹 서버채널, 넌블로킹 채널의 작동 이해
- 클라이언트의 경우 고유한 데이터 저장의 필요성 있음
  - 연결 수락 시 마다 Client 인스턴스 생성해 관리
  - 넌블로킹 방식의 소켓 채널로 개발 가능하나 서버 구현에 넌블로킹이 주로 쓰임
- 실행 방법은 TCP 블로킹 방식과 동일

4. TCP 비동기 채널

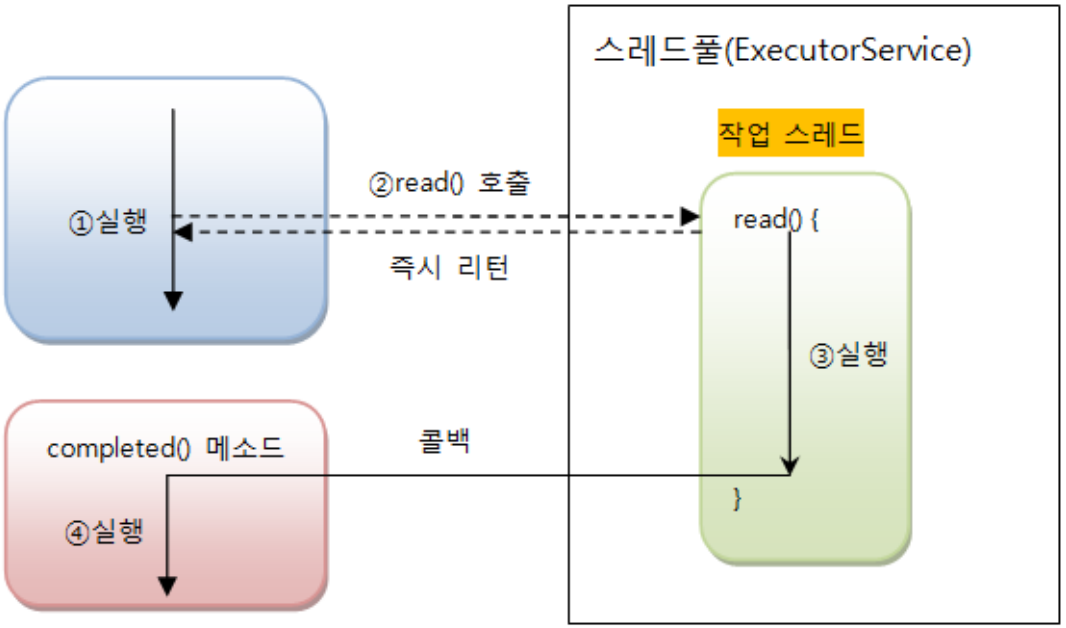
❖ TCP 비동기 채널의 특징

- connect(), accept(), read(), write()를 호출하면 즉시 리턴
  - 실질적 입출력 작업 처리는 스레드 풀의 스레드가 담당
  - 스레드가 작업 처리 완료하면 콜백 메소드 호출



4. TCP 비동기 채널

❖ read() 메소드 호출 예

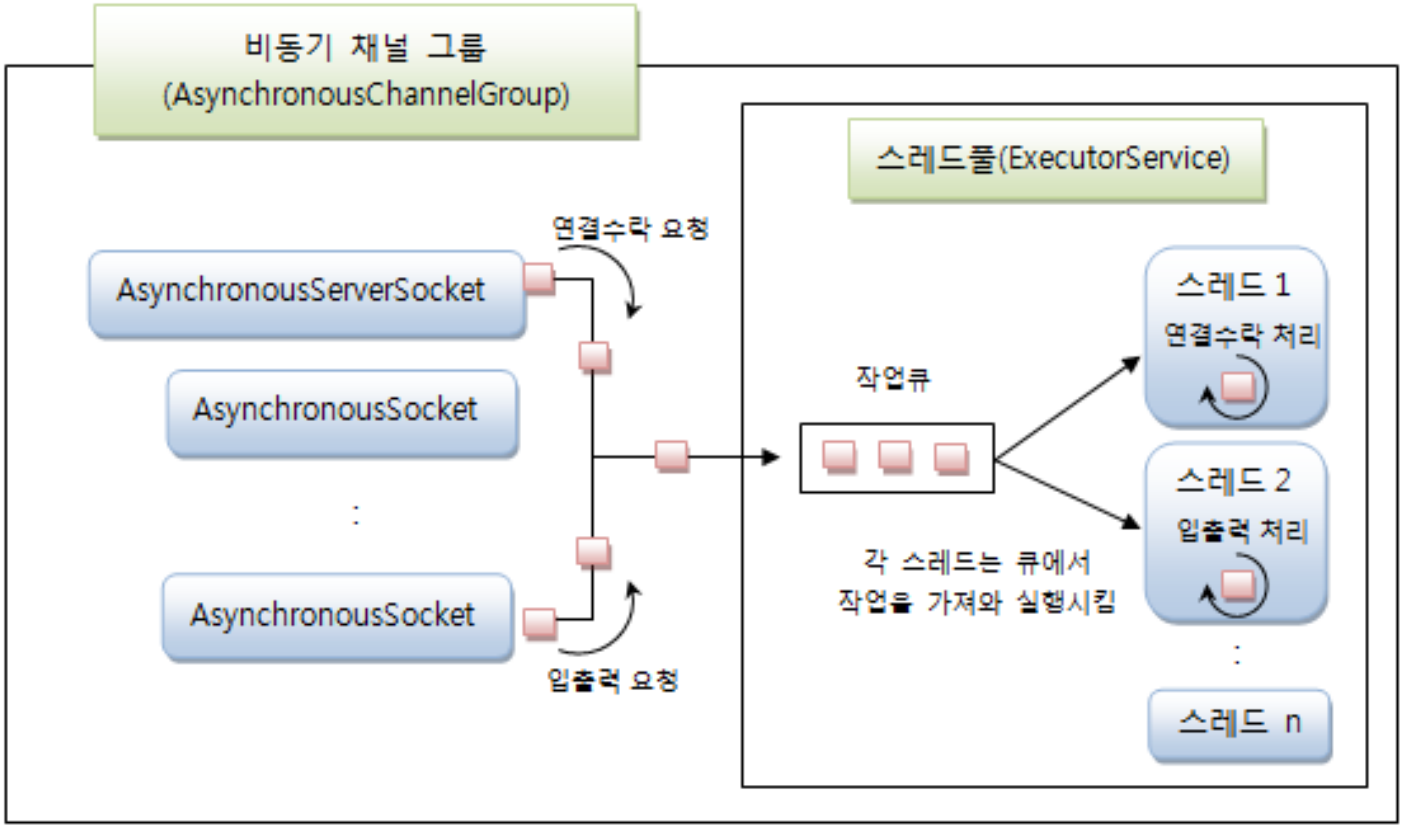


- 애플리케이션에서 read() 메소드를 호출하면 즉시 리턴
- 내부적으로 스레드풀의 작업 스레드가 read() 메소드 실질적으로 실행
- 작업 스레드가 read() 메소드를 모두 실행하고 나면 completed() 콜백
  - completed() 메소드 실행하는 스레드는 스레드풀의 작업 스레드

4. TCP 비동기 채널

❖ 비동기 채널 그룹 생성 및 종료

- 비동기 채널 그룹(AsynchronousChannelGroup)
  - 같은 스레드풀 공유하는 비동기 채널들의 묶음
  - 하나의 스레드풀을 사용한다면 모든 비동기 채널은 같은 채널 그룹



4. TCP 비동기 채널

- 비동기 채널 그룹 생성
  - 비동기 채널 생성할 때 채널 그룹 지정하지 않으면 기본 비동기 채널 그룹
  - 기본 비동기 채널 그룹은 내부적으로 생성되는 스레드풀 이용

- 비동기 채널 그룹 종료
  - shutdown()
    - 비동기 채널 그룹을 종료하겠다는 의사만 전달
    - 즉시 비동기 채널 그룹을 종료하지 않음
    - 비동기 채널 그룹에 포함된 모든 비동기 채널이 닫히면 종료
    - 새로운 비동기 채널을 포함시키려고 하면 ShutdownChannelGroupException이 발생
  - shutdownNow()
    - 강제적으로 비동기 채널 그룹에 포함된 모든 비동기 채널 닫고 비동기 채널 그룹을 종료
    - 완료 콜백 실행하고 있는 스레드는 종료되거나 인터럽트 X

4. TCP 비동기 채널

❖ 비동기 서버소켓 채널 생성 및 연결 수락 (p.1199~1201)

- 기본 비동기 채널 그룹에 포함되는 비동기 서버 채널 생성
- 새로 생성한 비동기 채널 그룹에 포함되는 비동기 서버 채널 생성
- 포트 바인딩
- 더 이상 소켓 채널이 사용되지 않으면 닫기
- 연결 수락

```
accept(A attachment, CompletionHandler<AsynchronousSocketChannel, A> handler);
```

  - 첫 번째 매개값은 콜백 메소드의 매개값으로 제공할 첨부 객체
  - 두 번째 매개값은 콜백 메소드를 가지고 있는 CompletionHandler<AsynchronousSocketChannel, A> 구현 객체

❖ 비동기 소켓 채널 (p.1202~1204)

- 클라이언트와 서버 연결 후의 통신

4. TCP 비동기 채널

❖ 채팅 서버, 클라이언트와 UI 구현

- P.1204~1218

- 비동기 서버소켓채널과 비동기 소켓채널 사용법 이해
- 서버에는 다수의 클라이언트가 붙는다는 것을 전제로 작업

- UI의 경우는 TCP 동기채널의 작동과 동일

5. UDP 채널(DatagramChannel)

❖ NIO에서의 UDP 채널

- DatagramChannel
  - 동기(블로킹)과 넌블로킹 방식 모두 사용 가능

❖ 발신자 만들기

- DatagramChannel 생성 - Open() 사용
  - Open() 할 때 ProtocolFamily 타입 매개값
    - StandardProtocolFamily 열거 상수
      - IPv4, IPv6 구분하기 위함
- 데이터 보내기 – send() 사용

```
int byteCount = datagramChannel.send(byteBuffer, new InetSocketAddress("localhost", 5001));
```

- 닫기 – close() 사용

```
datagramChannel.close();
```

5. UDP 채널(DatagramChannel)

❖ 수신자 만들기

- DatagramChannel 생성 및 포트 바인딩 – open(), bind()

```
DatagramChannel datagramChannel = DatagramChannel.open(StandardProtocolFamily.INET);
datagramChannel.bind(new InetSocketAddress(5001));
```

- 데이터 받기

```
SocketAddress socketAddress = datagramChannel.receive(ByteBuffer dst);
```

  - 데이터 받기 전까지 receive() 메소드 블로킹, 데이터 받으면 리턴
  - 작업 스레드를 생성해 receive() 메소드 반복적 호출
  - 작업 스레드 종료 방법
    - 작업 스레드의 interrupt() 호출시켜 ClosedByInterruptException 예외 발생
    - DatagramChannel의 close() 호출시켜 AsynchronousCloseException 예외 발생
    - 예외가 발생되면 예외 처리 코드에서 작업 스레드 종료

- 닫기

```
datagramChannel.close();
```