Announcement (1)

- Due date for PA3 is changed (~ next week)

- PA4 will also be started in the next class

	90	10	-10	100
학번	Correctness	Runtime	Warning	Score
2753		10.000		10.000
3065				
0717	89.700	10.000		99.700
3814	84.000	10.000		94.000
2243		10.000		
4275		10.000		
2019				
0197				
4601				
0963	85.700	10.000		95.700
0399				
1956	90.000	10.000	0	100.000
2759	90.000	10.000	0	100.000
2852	83.000	10.000	0	93.000
2341	84.400	10.000	0	94.400

Not submitted

Not scored

Concurrent Programming

Prof. Jin-Soo Kim(jinsookim@skku.edu)
TA – Sanghoon Han(sanghoon.han@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

Echo Server Revisited

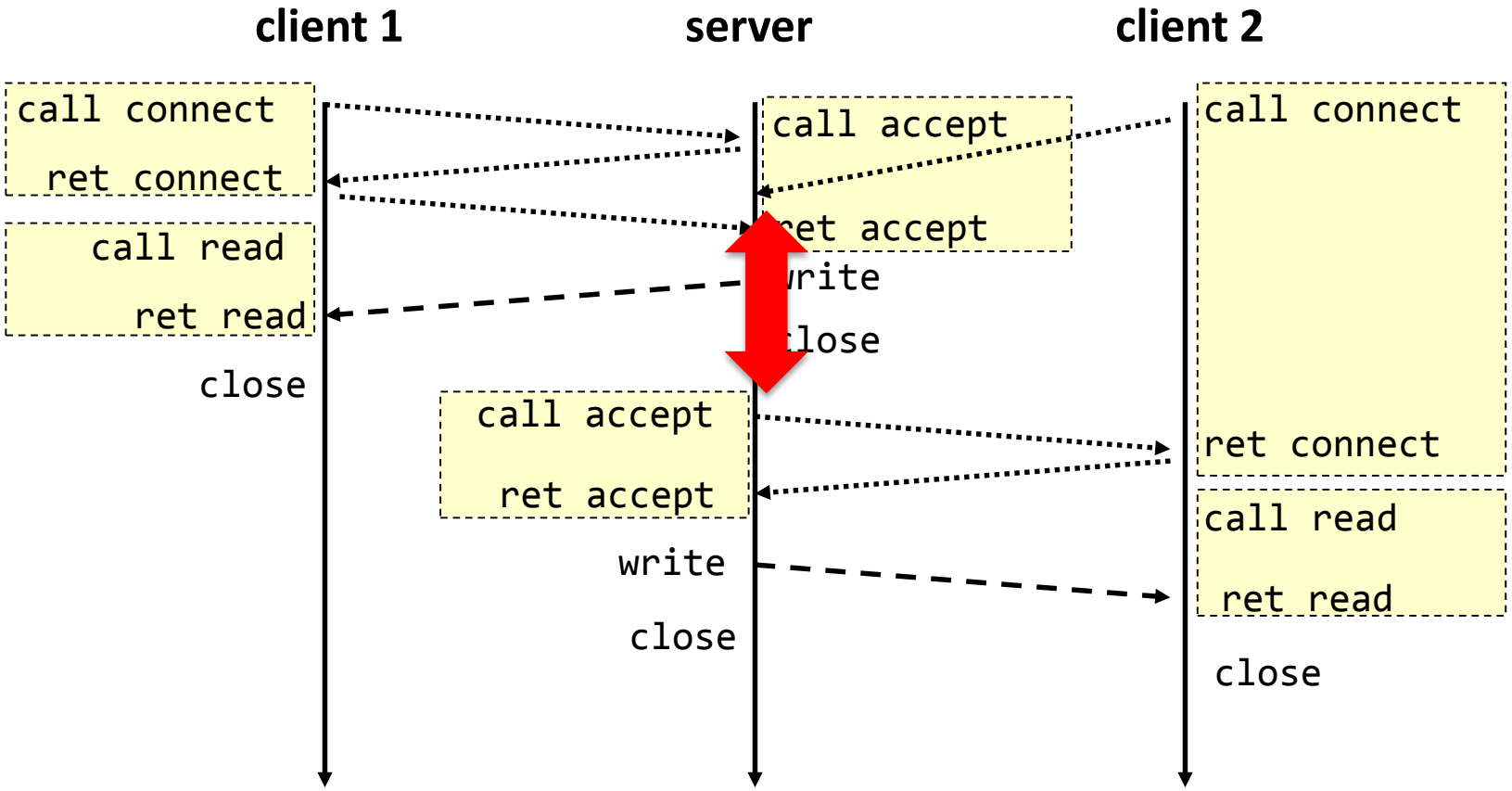
```
int main (int argc, char *argv[]) {
   ...
   listenfd = socket(AF_INET, SOCK_STREAM, 0);

   bzero((char *)&saddr, sizeof(saddr));
   saddr.sin_family = AF_INET;
   saddr.sin_addr.s_addr = htonl(INADDR_ANY);
   saddr.sin_port = htons(port);
   bind(listenfd, (struct sockaddr *)&saddr, sizeof(saddr));

   listen(listenfd, 5);
   while (1) {
      connfd = accept(listenfd, (struct sockaddr *)&caddr, &clen);
      while ((n = read(connfd, buf, MAXLINE)) > 0) {
         printf ("got %d bytes from client.\n", n);
         write(connfd, buf, n);
      }
      close(connfd);
   }
}
```

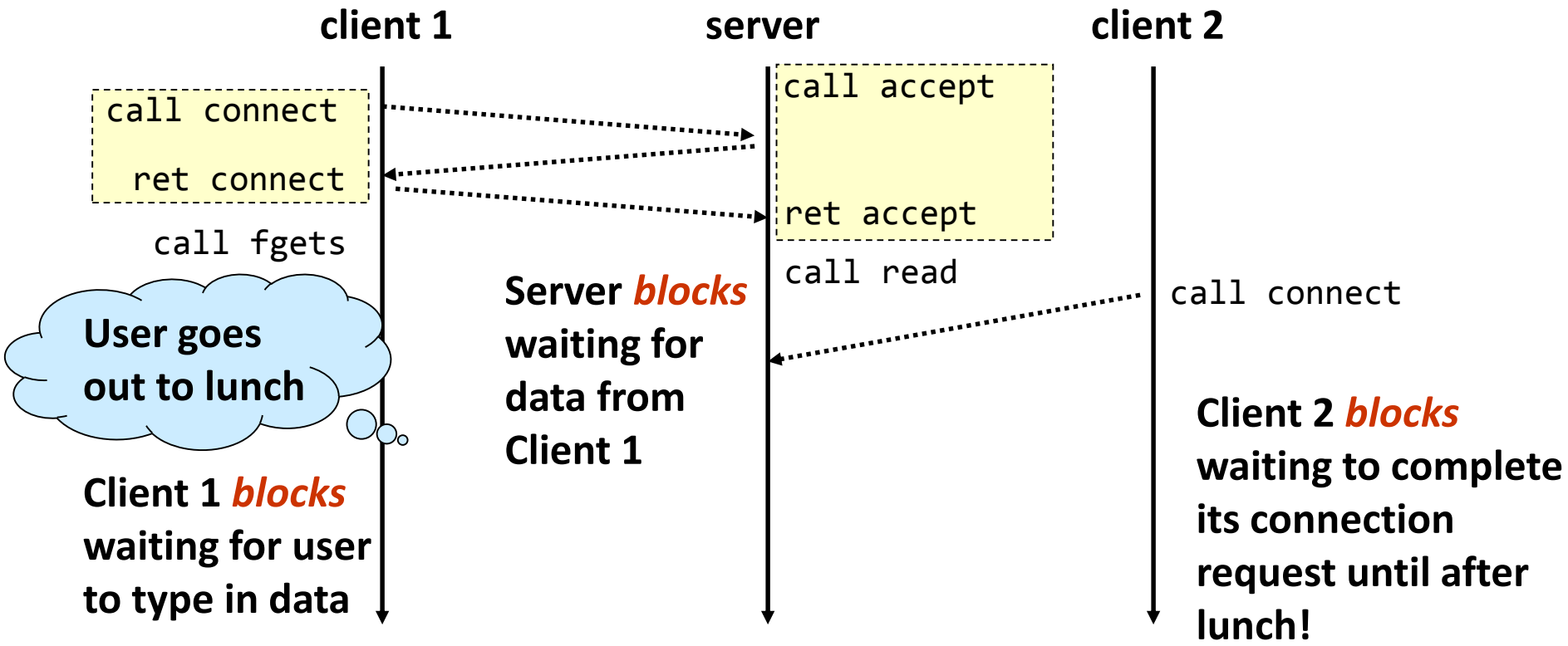
Iterative Servers (1)

- One request at a time



Iterative Servers (2)

- **Fundamental flaw**



- **Solution: use concurrent servers instead**
  - Use multiple concurrent flows to serve multiple clients at the same time.

Creating Concurrent Flows

- **Processes**
  - Kernel automatically interleaves multiple logical flows.
  - Each flow has its own private address space.
- **Threads**
  - Kernel automatically interleaves multiple logical flows.
  - Each flow shares the same address space.
  - Hybrid of processes and I/O multiplexing
- **I/O multiplexing with select()**
  - User manually interleaves multiple logical flows
  - Each flow shares the same address space
  - Popular for high-performance server designs.

Exercise #1

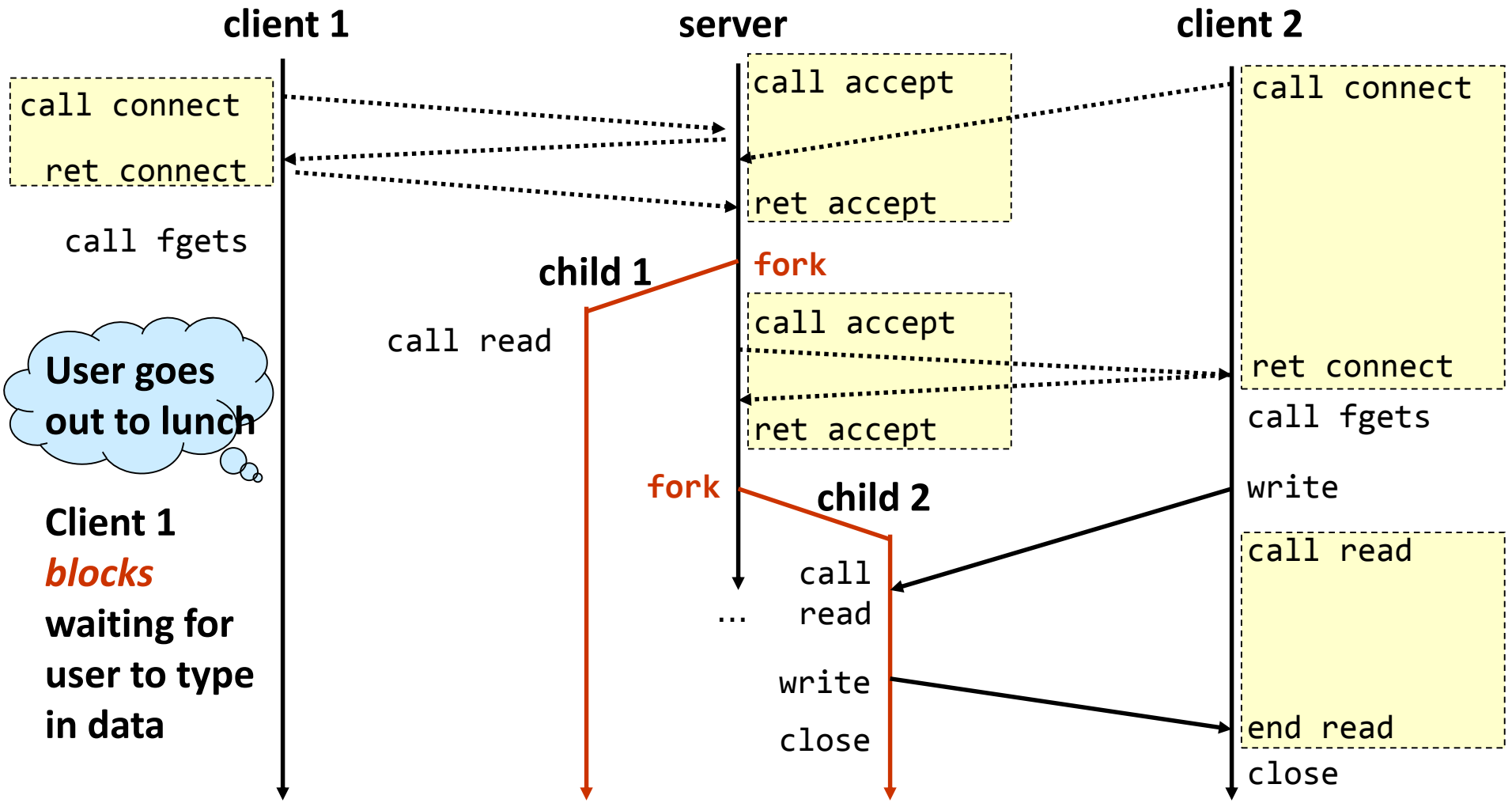
- **With your own code, make echo server**
  - At server side, print the number of characters that received from client
  - At client side, print the string that client typed

Concurrent Programming

Process-based

SKK UNIVERSITY

Process-based Servers



Implementation Issues

- **Servers should restart accept() if it is interrupted by a transfer of control to the SIGCHLD handler**
  - Not necessary for systems with POSIX signal handling.
  - Required for portability on some older Unix systems.
- **Server must reap zombie children**
  - to avoid fatal memory leak
- **Server must close its copy of connfd.**
  - Kernel keeps reference for each socket.
  - After **fork(), refcnt(connfd) = 2**
  - Connection will not be closed until **refcnt(connfd) = 0**

Exercise #2

- **With your own code, make process-based echo server**
  - At the same time, multiple client can be served by echo server
- **There should be no memory leakage**
  - There should be some codes that handle zombie process
  - How about closing files?

Process-based Designs

- **Pros**
  - Handles multiple connections concurrently.
  - Clean sharing model.
    - Descriptors (no), file tables (yes), global variables (no)
  - Simple and straightforward.
- **Cons**
  - Additional overhead for process control.
    - Process creation and termination
    - Process switching
  - Nontrivial to share data between processes.
    - Requires IPC (InterProcess Communication) mechanisms: FIFO's, System V shared memory and semaphores

Echo Server

- **Iterative version**

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

   while (1) {
      connfd = accept (listenfd, (struct sockaddr *)&caddr,
                       &caddrlen));

      while ((n = read(connfd, buf, MAXLINE)) > 0) {
         printf ("got %d bytes from client.\n", n);
         write(connfd, buf, n);
      }

      close(connfd);
   }
}
```

Echo Server: Process-based

```
int main (int argc, char *argv[])
{
    . . .
    signal (SIGCHLD, handler);

    while (1) {
        connfd = accept (listenfd, (struct sockaddr *)&caddr,
                         &caddrlen));
        if (fork() == 0) {
            close(listenfd);
            while ((n = read(connfd, buf, MAXLINE)) > 0) {
                printf ("got %d bytes from client.\n", n);
                write(connfd, buf, n);
            }
            close(connfd);
            exit(0);
        }
        close(connfd);
    }
}
```

```
void handler(int sig) {
    pid_t pid;
    int stat;
    while ((pid = waitpid(-1, &stat,
                          WNOHANG)) > 0);
    return;
}
```