Concurrent Programming

Prof. Jin-Soo Kim(jinsookim@skku.edu)
TA – Taekyun Roh(tkroh0198@skku.edu)
Sewan Ha(hsewan2495@gmail.com)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

SKK UNIVERSITY

Announcement

- **PA1 is graded**
  - Uploaded on i-campus
  - If any questions, please contact **tkroh0198@skku.edu** or ask **Taekyun Roh** directly.
  - Room # : 85545(Internet Management Technology Lab)

    Time : 2018/11/12 (Monday) 16:00~18:00

- **PA2 will be uploaded until this Sunday**

Echo Server Revisited

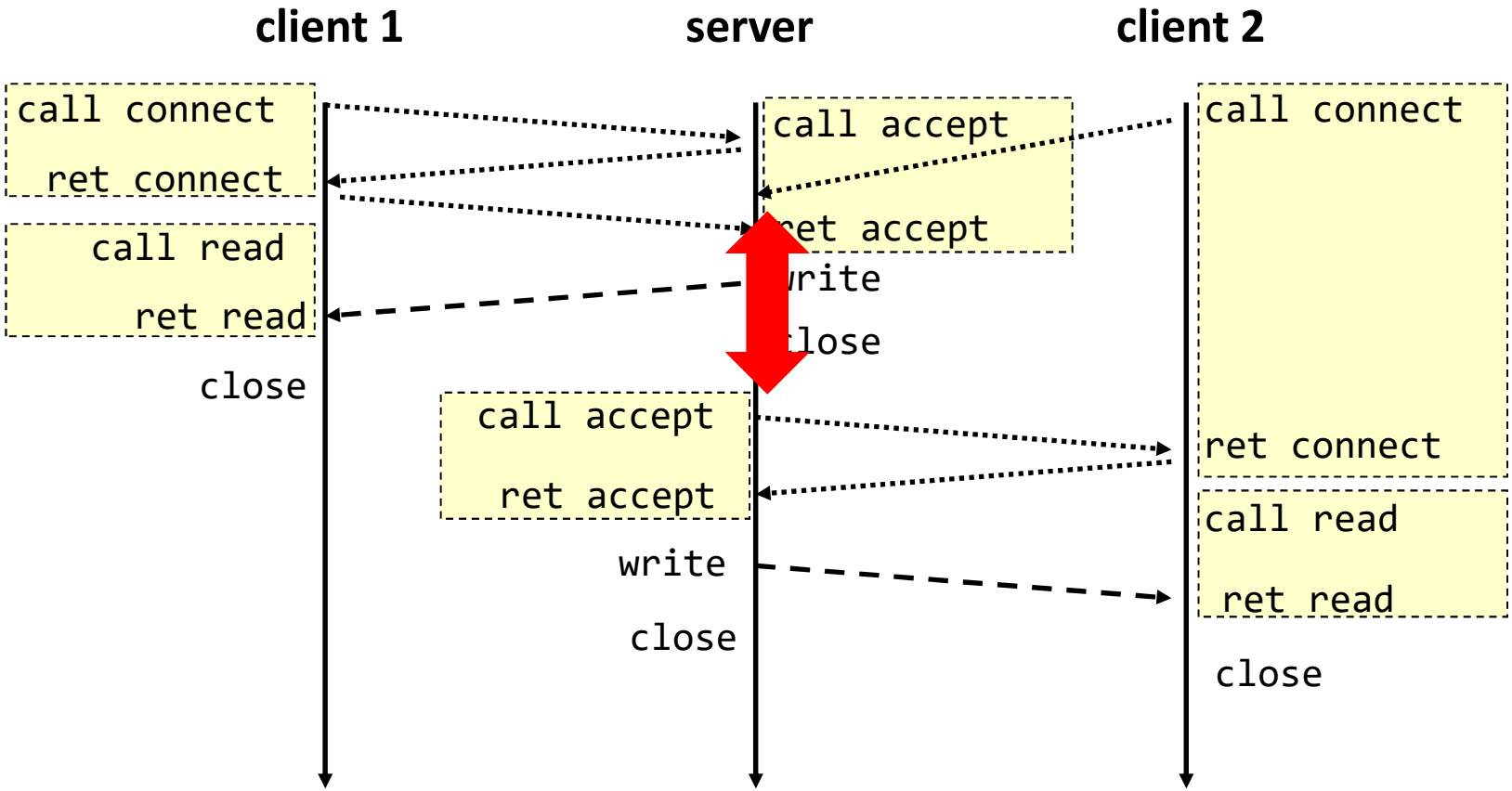
```
int main (int argc, char *argv[]) {
   ...
   listenfd = socket(AF_INET, SOCK_STREAM, 0);

   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);
   }
}
```

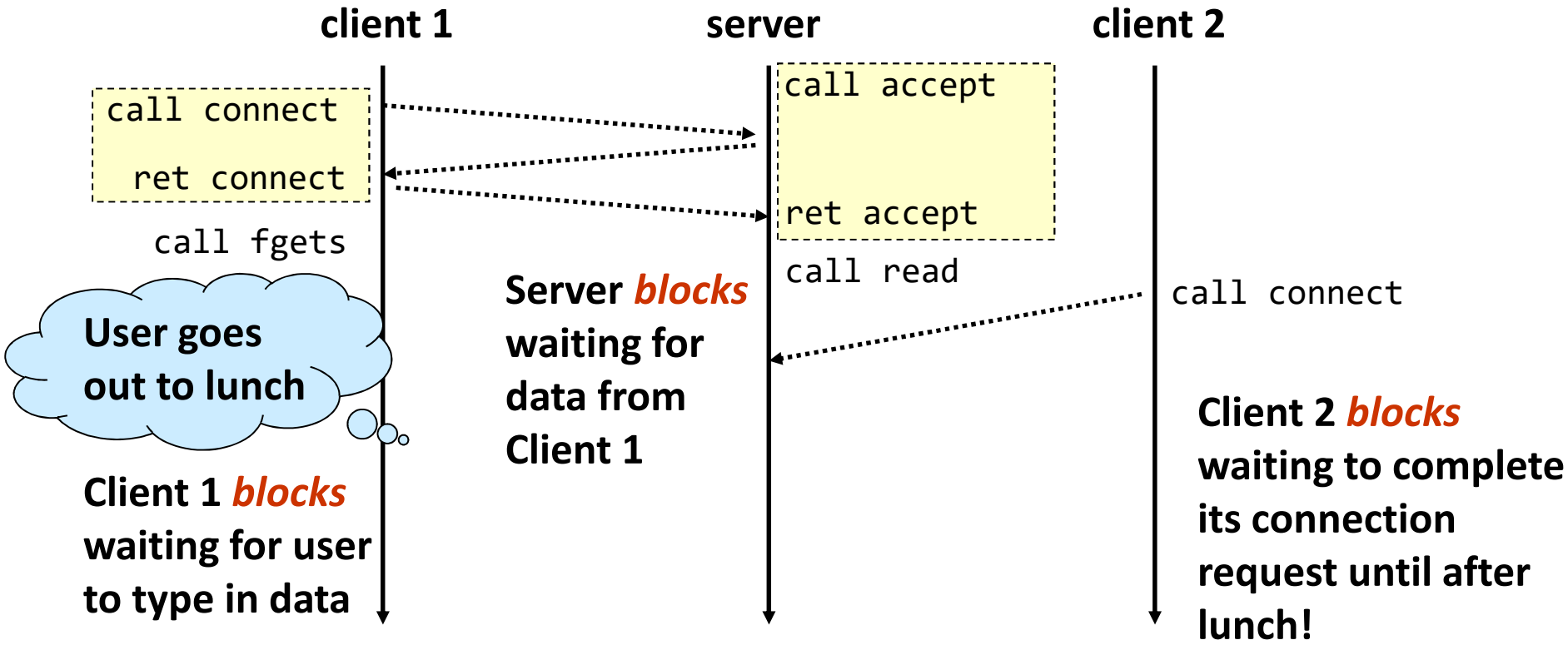
Iterative Servers (1)

- 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

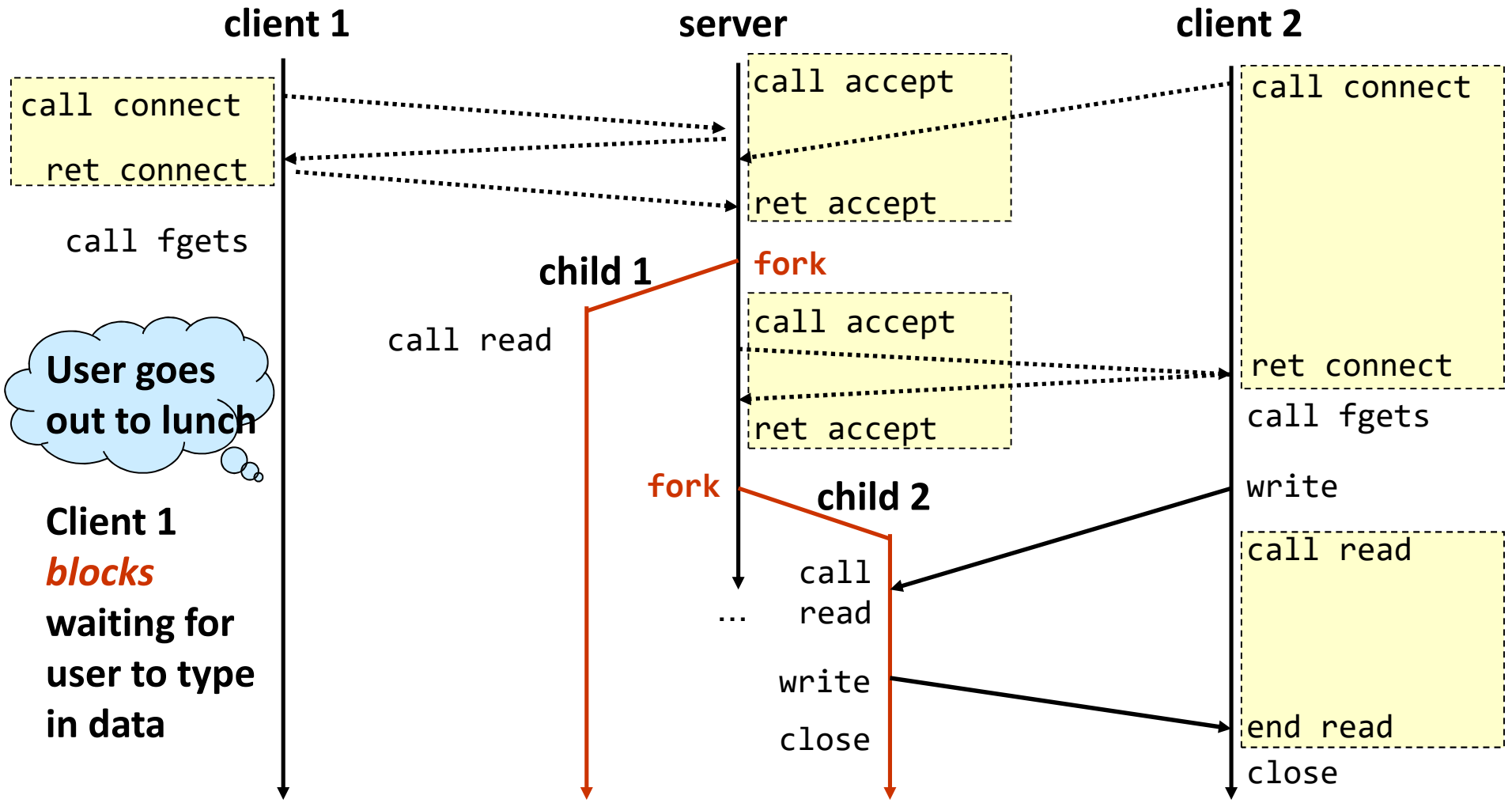
- **Processes**
  - Kernel automatically interleaves multiple logical flows.
  - Each flow has its own private address space.
- **Threads**
  - Kernel automatically interleaves multiple logical flows.
  - Each flow shares the same address space.
  - Hybrid of processes and I/O multiplexing
- **I/O multiplexing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

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**

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;
}
```

Exercise #1

- **With your own code, make calculator server**
  - At server side, calculate string(**only one arithmetic operation(+,-,*,/,%)**) transmitted from client
  - At client side, print the result of calculation
  - Hint)
    - You don't have to change client-side code
    - To change string to integer and vice versa, you can use **sprintf() and sscanf()**

Sample Input)

```
server connected to localhost (127.0.0.1)
connection terminated.
```

Sample Output)

```
host : 127.0.0.1
2+3
result : 5
3+4
result : 7
5*7
result : 35
20-30
result : -10
80/3
result : 26
9%2
result : 1

invalid calculation
^C
```

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?
- **Print process ID**

Sample Result)

```
Server connected to localhost (127.0.0.1)
pid(23810) : got 6 bytes from client.
Server connected to localhost (127.0.0.1)
pid(23812) : got 2 bytes from client.
pid(23812) : got 6 bytes from client.
pid(23812) : got 4 bytes from client.
```