네트워크 프로그래밍 6장

중급 소켓 프로그래밍 (3)

목포해양대 해양컴퓨터공학과

목차

멀티태스킹

- 멀티태스킹이란?
  - 사전적 의미
    - 한 사람의 사용자가 한 대의 컴퓨터로 2가지 이상의 작업을 동시에 처리하거나, 2가지 이상의 프로그램들을 동시에 실행시키는 것
- 소켓에서의 멀티태스킹
  - 다중 접속 서버의 구현을 의미
    - Fork을 이용한 멀티 프로세스, thread를 이용한 멀티 스레드 기법을 이용하여 하나의 TCP 서버가 다수개의 TCP 클라이언트를 동시에 처리하게 하는 기법
- 소켓에서의 멀티태스킹 기법
  - fork를 이용한 멀티태스킹
  - Thread를 이용한 멀티태스킹

fork()

- fork()
  - 자신과 완전히 동일한 코드를 가진 새로운 프로세스를 생성
    - 부모프로세스(데이터영역, 힙, 스택)를 그대로 복사
    - 원본 소스의 PC(program counter)까지 복사를 하기 때문에 새로 생성된 프로세서도 fork() 이후부터 실행
  - Process
    - 리눅스 기반에서 실행되는 모든 프로그램
    - 각 프로세스는 ID(PID)라고 불리는 번호를 가지고 있다
  - 부모 프로세스 vs 자식 프로세스
    - 부모 프로세스 : 새로운 프로세스를 호출(fork)한 프로세스
    - 자식 프로세스 : 새롭게 호출된(forked) 프로세스
- fork를 이용한 멀티태스킹 시 주의점
  - 새로 생성된 자식 프로세스와 부모 프로세스는 변수나 메모리를 공유하지 않음(단 외부 파일, 소켓 등은 공유가능)
  - 프로세스 증가로 인한 성능 감소
  - 변수나 메모리 공유가 필요할 경우 => 스레드 사용

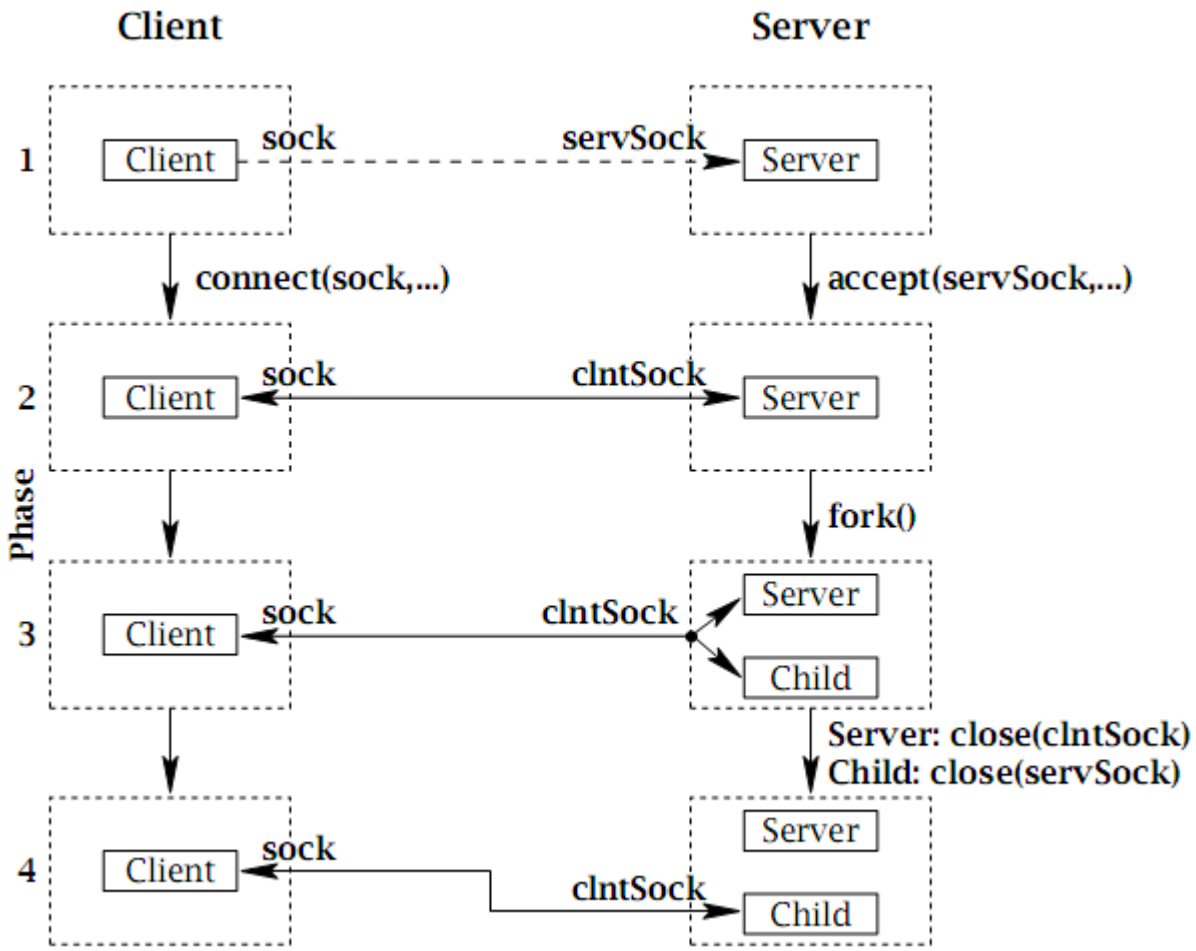
fork() example

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void); /* 프로세스를 복사 */
```

- fork()가 호출되면 동일한 프로세스가 두 개로 복사되어 실행된다
  - 부모와 자식 프로세스를 구분하기 위하여 반환값을 검사해야 함
    - 부모 프로세스의 fork()는 자식 프로세스의 process id(pid)를 리턴
    - 자식 프로세스의 fork()는 숫자 0을 리턴
    - 에러 -> -1

```
#include <unistd.h>
#include <sys/types.h>
pid_t pid;
pid=fork(); /* copy new process */
if(pid==0){
        /* new process code here */
}
else{
        /* parent code here */
}
```

fork()를 이용한 다중 클라이언트의 처리



fork()를 이용한 다중 클라이언트의 처리

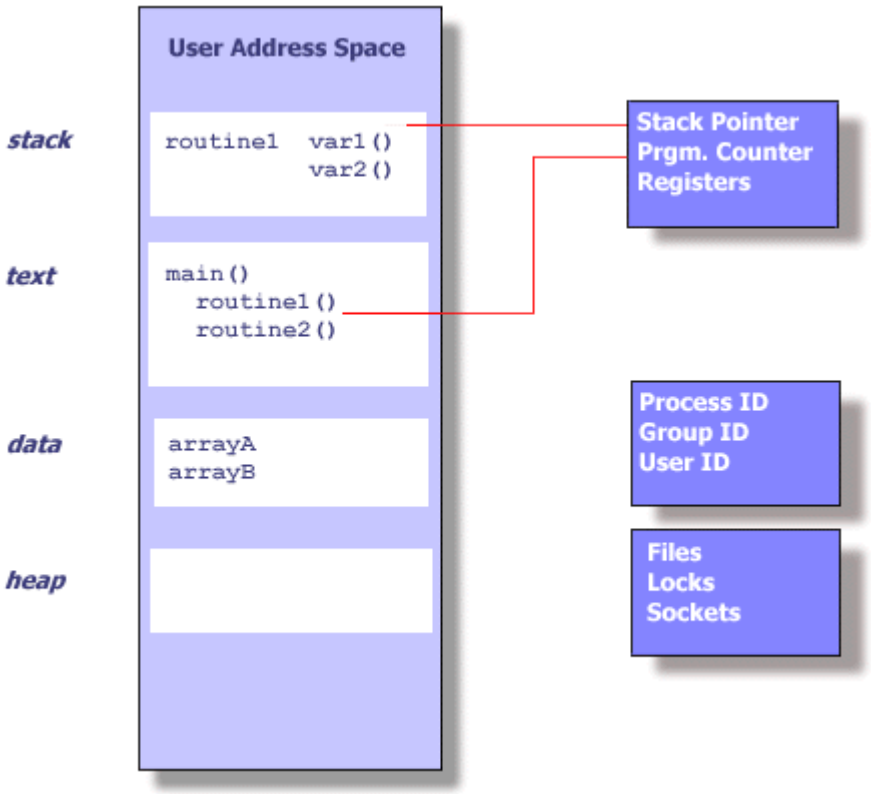
```
pid_t processID;
for (;;)        {
   if(clntSock = accept(servSock, (struct sockaddr *)
        &echoClntAddr, &clntLen)) < 0)
        DieWithError("accept() failed");

   if ((processID = fork())<0)
                DieWithError("fork() failed");

   else if (processID == 0) { /* 자식프로세스: 클라이언트 처리 */
        close(servSock);
        HandleTCPClient(clntSock);
        exit(0);
   }
   /* 부모 프로세스 : 반복적으로 클라이언트의 접속을 처리 */
}
```

Thread를 이용한 멀티태스킹

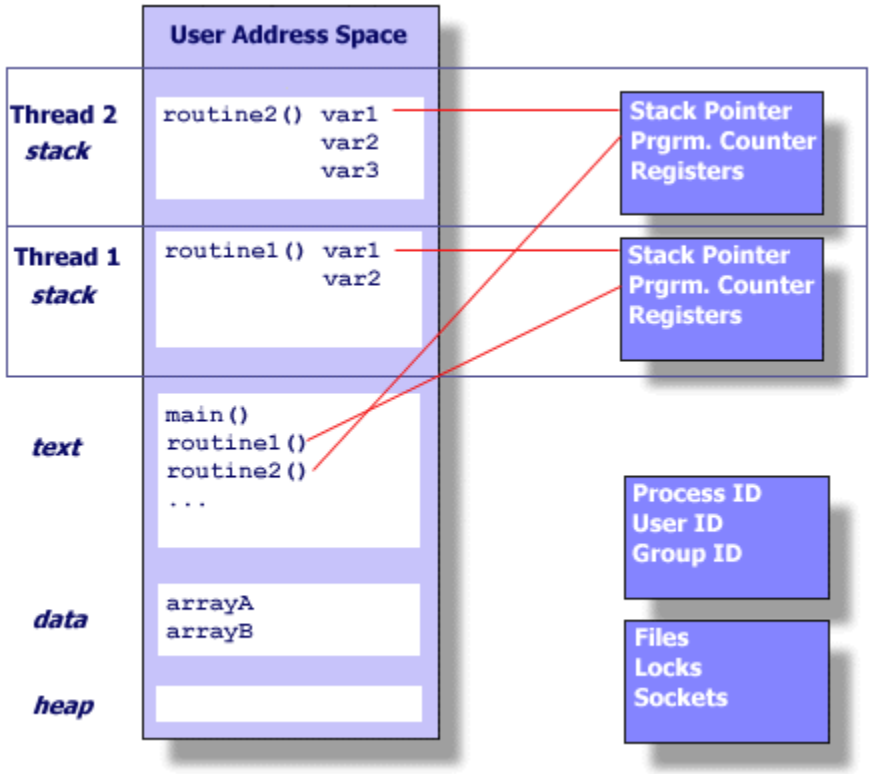
- Thread란?
  - semi process, light weight process
  - thread간 메모리 공유
    - fork에 비해서 빠른 프로세스 생성 능력과 적은 메모리를 사용
- Network Programming에서의 thread
  - 다중 클라이언트 처리를 위한 서버프로그래밍 작업
  - 공유변수에서 값을 처리할 경우 사용
    - 동기화 문제 어려움->쓰기의 경우 mutex사용

Process 와 Thread

단일 프로세스



멀티 쓰레드



Pthread

- POSIX thread
  - POSIX에서 표준으로 제안한 thread 함수 set
  - POSIX란?
    - portable operating system interface
    - 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격
- pthread 실행 순서
  - pthread create()
    - worker(thread)가 생성
  - worker 시작
    - 각 worker는 그들의 작업을 실행
  - worker 종료
    - pthread_join()에 의해서 worker를 하나로 모음

Thread 생성

```
int pthread_create(pthread_t * thread,
                   const pthread_attr_t * attr,
                   void * (*start_routine)(void *),
                   void *arg);
```

- pthread_t thread
  - 생성된 스레드 ID를 저장할 변수
- pthread_attr_t attr
  - Set to NULL if default thread attributes are used.
- void * (*start_routine)
  - pointer to the function to be threaded.
  - Function has a single argument: pointer to void.
- void *arg
  - pointer to argument of function

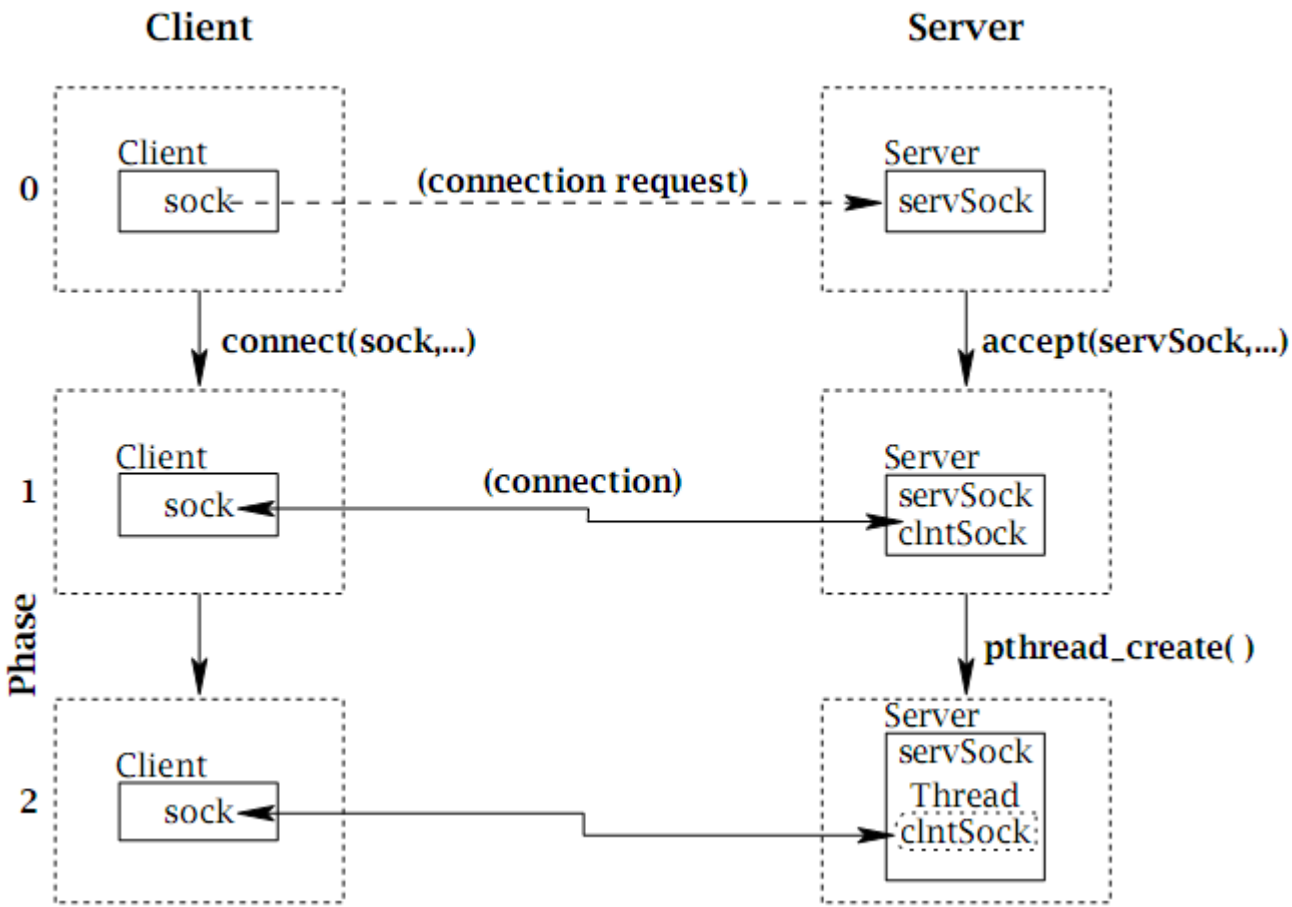
pthread example()

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5

void *PrintHello(void *threadid)
{
   int tid;
   tid = (int)threadid;
   printf("Hello World! It's me, thread #%d!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])
{
    pthread_t threads[NUM_THREADS];
    int rc, t;
    for(t=0;t<NUM_THREADS;t++){
          printf("In main: creating thread %d\n", t);
          rc = pthread_create(&threads[t], NULL,
                              PrintHello, (void *)t);
          if (rc){
                    printf("ERROR; return code from pthread_create() is %d\n", rc);
                    exit(-1);
          }
    }
    pthread_exit(NULL);
}
```

Output

```
In main: creating thread 0
In main: creating thread 1
Hello World! It's me, thread #0!
In main: creating thread 2
Hello World! It's me, thread #1!
Hello World! It's me, thread #2!
In main: creating thread 3
In main: creating thread 4
Hello World! It's me, thread #3!
Hello World! It's me, thread #4!
```

pthread()를 이용한 다중 클라이언트의 처리



pthread()를 이용한 다중 클라이언트의 처리

```
#include <pthread.h>

pthread_t tid;

void *do_thread(void *arg);

for (;;) {
    if(clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr, &clntLen))
                  < 0)
          DieWithError("accept() failed");
    if(pthread_create(&tid, NULL, do_thread, (void *)clntSock) < 0 )
          DieWithError("thread create() failed");
}
…

void *do_thread(void *arg)
{
    int csock;
    csock=(int)arg;
    HandleTcpClient(csock);
    pthread_exit(NULL);
}
```

브로드 캐스팅

- 브로드캐스트란?
  - LAN전체에 데이터를 뿌리는 전송 방식
    - 네트워크 부하를 줄이기 위해 브로드캐스팅은 LAN으로 제한
- 브로드캐스팅 전송 방식
  - 서브넷 직접 전송
    - 특정 서브넷의 모든 호스트에 전송
    - e.g.) 203.252.153.255 // 203.252.153.0 네트워크의 모든 호스트에게 전송
  - 제한된 브로드캐스팅
    - 전송 호스트가 속한 LAN의 모든 호스트에게 전송
    - e.g) 255.255.255.255
    - 라우터는 해당 패킷을 전달하지 않음

브로드캐스팅 방법

- 브로드캐스팅은 UDP만 지원
- 수신자는 수정사항 없으며 송신자는 아래와 같은 약간의 수정내용 필요

```
/* 서버 주소 구조체 초기화 시 주소를 브로드캐스팅 주소로 설정 */
SOCKADDR_IN serv;
memset(&remoteaddr, 0, sizeof(remoteaddr));
serv.sin_family = AF_INET;
serv.sin_port = htons(9000);
serv.sin_addr.s_addr = htonl(INADDR_BROADCAST);
…
…

/* 소켓을 브로드캐스팅이 가능토록 설정 */
int broadcastPerm = 1;
if (setsockopt(sock,SOL_SOCKET, SO_BROADCAST, $broadcastPerm,
        sizeof(broadcastPerm)) < 0)
```

응용과제

- 다자간 채팅 프로그램
  - 클라이언트는 시작할 때 자신의 별명과 서버 이름, 포트번호를 전달받아 시작한다.
  - 서버는 접속한 클라이언트의 명단(별명, IP 주소, 포트 번호 등)을 유지해야 한다.
  - 사용자가 새로 들어오는 경우 해당 클라이언트에게 환영 메시지와 현재 접속자 명단을 보내고, 나머지 클라이언트에게는 새로 사용자(별명)가 들어왔음을 통보한다.
  - 사용자가 입력한 메시지는 그 사용자를 제외한 나머지 사용자들에게 전달되어야 한다.

응용과제

- 다자간 채팅 프로그램 (계속)
  - 서버는 사용자가 없는 경우에만 종료할 수 있다. (시그널 처리)
  - 접속을 종료할 클라이언트는 서버에게 그 사실을 알려야 한다. (leave)

응용과제

- 다자간 채팅 프로그램 (계속)
  - 서버는 주기적으로 각 클라이언트가 살아있음을 확인하기 위한 메시지(alive_req)를 보내고, 각 클라이언트는 해당 메시지에 응답(alive_resp)해야 한다.
  - 서버의 alive_req에 주어진 시간(설정 값)동안 응답하지 않은 사용자는 명단에서 제거한다.
  - 서버는 주기적으로 사용자 수와 명단을 서버 화면에 출력한다.

응용과제

- 다자간 채팅 프로그램 (계속)
  - 다음 설명 반드시 포함
    - 서버와 클라이언트가 교환할 명령 정의 및 설명
      - 메시지 전송 (data), 종료(leave), alive_req, alive_resp 등
    - 서버에서 클라이언트 관련 자료 유지 방법
  - 클라이언트 5개 이상 실행하여 각 단계(명령) 수행 화면 캡처
    - 초기 접속화면
      - 환영 메시지 / 신규 접속 알림 메시지
    - 채팅 화면
      - 송신자 / 수신자
    - 서버의 상태(사용자 수, 명단) 출력 화면
    - 종료 화면