Introduction/Tutorial on the Linux Ecosystem

Bash scripting, SSH, POSIX Threads, BSD Sockets (part 2)

Alexandru Iulian Orhean
aorhean@hawk.iit.edu

Illinois Institute of Technology
Computer Science Department
Data-Intensive Distributed Systems Laboratory

Table of contents

Introduction

The Shell

Bash Scripting

Secure Shell

SSH

- cryptographic network protocol;
- secure channel over unsecured network;
- client-server architecture (SSH client and SSH server);
- OpenSSH is the most popular implementation;

Uses:

- login remote host;
- execute command on remote host;
- automatic login (passwordless login) to remote server;
- secure file transfer;
- forwarding and tunneling ports;
- forwarding X from a remote host;

SSH to Hyperion SLURM cluster

Credentials:

- username = IIT email/username (without domain name)
- password = $tudent<last 4 digits of your CWID>

Example:

- alex5@hawk.iit.edu -> alex5
- CWID 12345678 => $tudent5678

Linux and MacOS

```
localhost$ ssh <username>@129.114.33.105
hyperion$ passwd
```

Windows

- install Putty;
- or Linux Subsystem for Windows;
- or Cygwin;

Public-Private Key Setup

Linux, MacOS and Linux Subsystem for Windows;

```
localhost$ ssh-keygen
localhost$ ls -l ~/.ssh
-rw------- id_rsa
-rw-r--r-- id_rsa.pub

locahost$ ssh-copy-id <username>@129.114.33.105
locahost$ ssh -i ~/.ssh/id_rsa <username>@129.114.33.105

localhost$ scp file1.txt <username>@129.114.33.105:file1.txt
localhost$ scp <username>@129.114.33.105:file2.txt file2.txt
```

Windows

- Putty - specify path to private key;
- Cygwin - install openssh-client package;

POSIX Threads

Processes and Threads

Process

- program in execution, the unit of work in a computer;
- encapsulates the necessary resources that a program needs: CPU time, memory, files, I/O devices;

Threads

- basic unit of CPU utilization;
- comprises a thread ID, program counter, register set and a stack;
- kernel-level vs user-lever thread libraries;
- Linux uses lightweight processes (LWP) (hardware threads?);

POSIX Threads (Pthreads)

- standardized C language programming interface;

POSIX Threads Programming LLNL - `https://computing.llnl.gov/tutorials/pthreads/`

POSIX Threads Example

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 32
pthread_mutex_t mutex;
long shared_var;

void *work(void *args)
{
    long tid, local_var;

    tid = (long) args;
    pthread_mutex_lock(&mutex);
    local_var = shared_var++;
    pthread_mutex_unlock(&mutex);
    printf("Thread %ld got %ld\n", tid, local_var);

    pthread_exit(NULL);
}
```

POSIX Threads Example

```
int main(int argc, char **argv)
{
    pthread_t threads[NUM_THREADS]; long tid; int rc;
    pthread_mutex_init(&mutex, NULL); shared_var = 0;

    for (tid = 0; tid < NUM_THREADS; tid++) {
        rc = pthread_create(&threads[tid], NULL, work, (void *) tid);
        if (rc) {
            printf("Could not create thread %ld\n", tid);
        }
    }

    for (tid = 0; tid < NUM_THREADS; tid++) {
        rc = pthread_join(threads[tid], NULL);
        if (rc) {
            printf("Could not join thread %ld\n", tid);
        }
    }
    pthread_mutex_destroy(&mutex);
    pthread_exit(NULL);
}
```

POSIX Threads Example

```
$ gcc -Wall -o main.exe main.c -lpthread

run.slurm
----------------------------
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --output=main.out

./main.exe

$ sbatch run.slurm
$ squeue
```

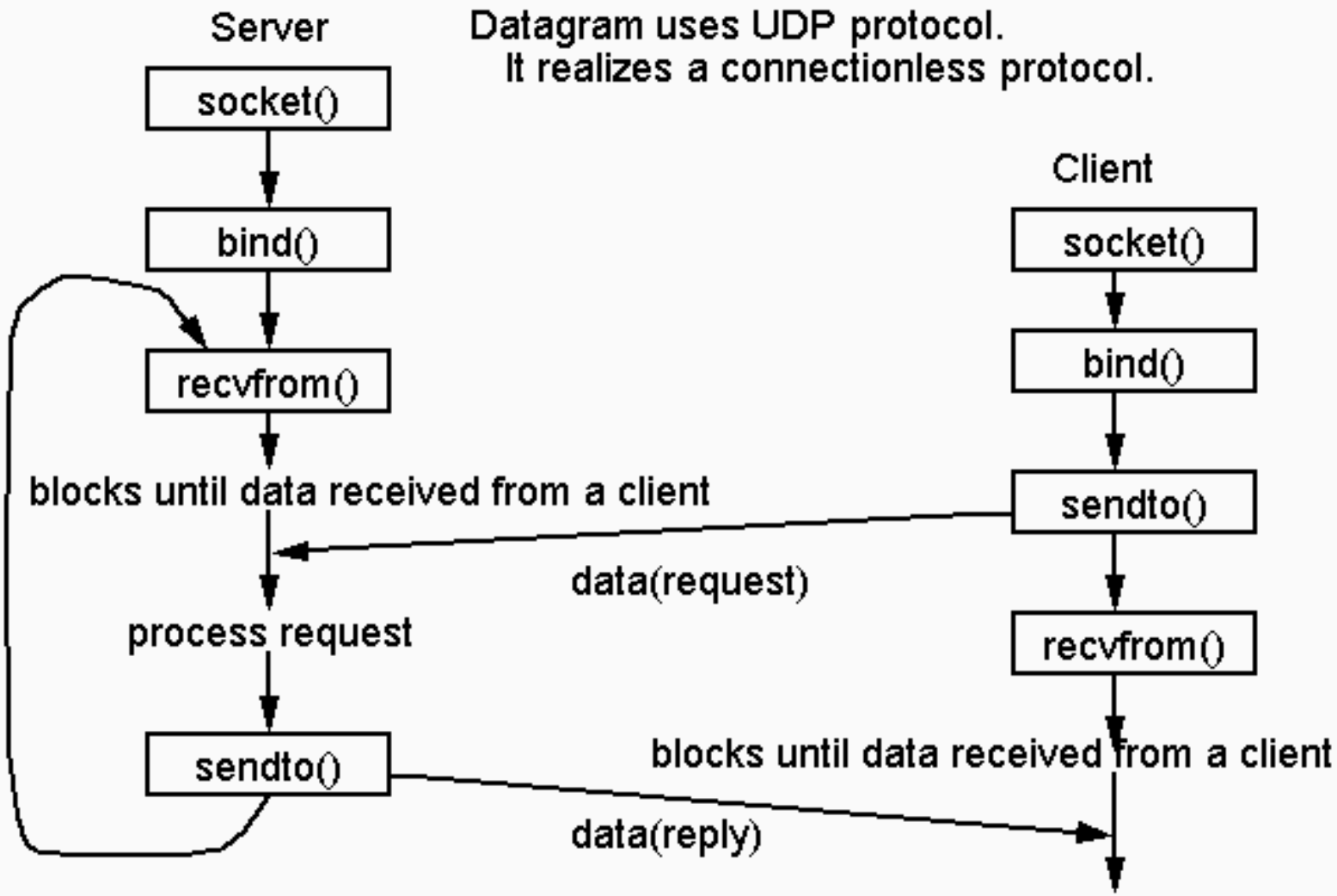
BSD Sockets

BSD Sockets

- generalized interprocess communication channel;
- represented by a file descriptor;
- supports communication between unrelated processes, even running on different machines that communicate over the network;
- select the communication style and protocol to use (SOCK_STREAM / SOCK_DGRAM);

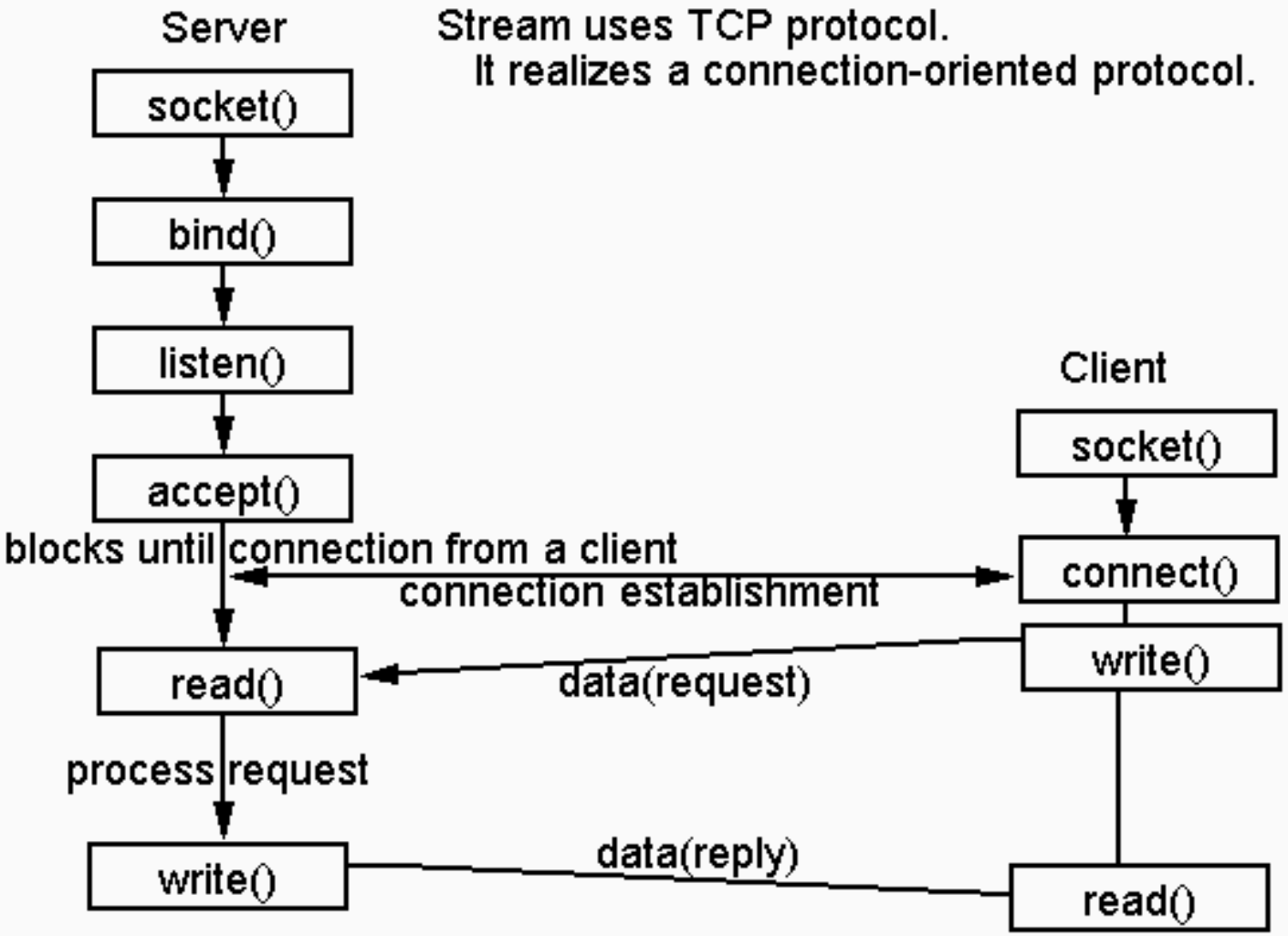
Pragmatic Up-to-Date Tutorial -
`https://beej.us/guide/bgnet/html/single/bgnet.html`

UDP Flowchart



http://cs.uccs.edu/~cs522/pp/f99pp-3.gif

TCP Flowchart



http://cs.uccs.edu/~cs522/pp/f99pp-4.gif

Sockets TCP example

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <string.h>

#define BUFF_SIZE 64
#define ERROR -1
```

Sockets TCP example (server)

```
int sockfd, newfd, rc;
char buffer[BUFF_SIZE];
struct addrinfo hints, *res;
socklen_t addrlen;
struct sockaddr_storage clt;

hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;

rc = getaddrinfo(NULL, "11155", &hints, &res);
if (rc != 0) {
    printf("Could not get address information!\n");
    return ERROR;
}

sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
if (sockfd < 0) {
    printf("Could not create socket!\n");
    return ERROR;
}
```

Sockets TCP example (server)

```
if (bind(sockfd, res->ai_addr, res->ai_addrlen) < 0) {
    printf("Could not bind socked!");
    close(sockfd);
    return ERROR;
}

if (listen(sockfd, 10) == -1) {
    printf("Could not listen to socket!");
    close(sockfd);
    return ERROR;
}

newfd = accept(sockfd, (struct sockaddr *) &clt, &addrlen);
if (newfd < 0) {
    printf("Could not accept client!\n");
    close(sockfd);
    return ERROR;
}
```

Sockets TCP example (server)

```
memset(buffer, 0, BUFF_SIZE);
rc = recv(newfd, &buffer[0], BUFF_SIZE, 0);

if (rc == 0) {
    printf("Connection closed at receive!\n");
    break;
}

if (rc < 0) {
    printf("Could not receive package!\n");
    break;
}
printf("%s", buffer);
```

Sockets TCP example (server)

```
strcpy(buffer, "Absolutely!");
rc = send(newfd, &buffer[0], BUFF_SIZE, 0);

if (rc < 0) {
    printf("Could not send package!\n");
}

freeaddrinfo(res);
close(newfd);
close(sockfd);
```

Sockets TCP example (client)

```
int sockfd, rc;
char buffer[BUFF_SIZE];
struct addrinfo hints, *res;

hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;

rc = getaddrinfo("<server-ip>", "11155", &hints, &res);
if (rc != 0) {
    printf("Could not get address information!\n");
    return ERROR;
}
```

Sockets TCP example (client)

```
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
if (sockfd < 0) {
    printf("Could not create socket!\n");
    return ERROR;
}

rc = connect(sockfd, res->ai_addr, res->ai_addrlen);
if (rc < 0) {
    printf("Could not connect to server!\n");
    return ERROR;
}
```

Sockets TCP example (client)

```
strcpy(buffer, "Can you hear me?");
rc = send(sockfd, &buffer[0], BUFF_SIZE, 0);

if (rc < 0) {
    printf("Could not send package!\n");
}
```

Sockets TCP example (client)

```
memset(buffer, 0, BUFF_SIZE);
rc = recv(sockfd, &buffer[0], BUFF_SIZE, 0);

if (rc == 0) {
    printf("Connection closed at receive!\n");
    break;
}

if (rc < 0) {
    printf("Could not receive package!\n");
    break;
}
printf("%s", buffer);

freeaddrinfo(res);
close(sockfd);
```

Last remarks

Where there is a shell, there is a way.