Distributed Programming

Go vs C

Agenda

- Definition: Distributed Programming
- Cross Platform Deployment
- Network Support
  - IPv4/6
  - TLS
  - Code Maintenance
- Language Features
  - Type concept
  - Error handling
- Concurrent Connection Handling
- Portability

Distributed Programming

- No shared Memory
- Asynchronous
- Not Parallel Programming

- Requires:
  - Exchange of information
  - Exchange of commands

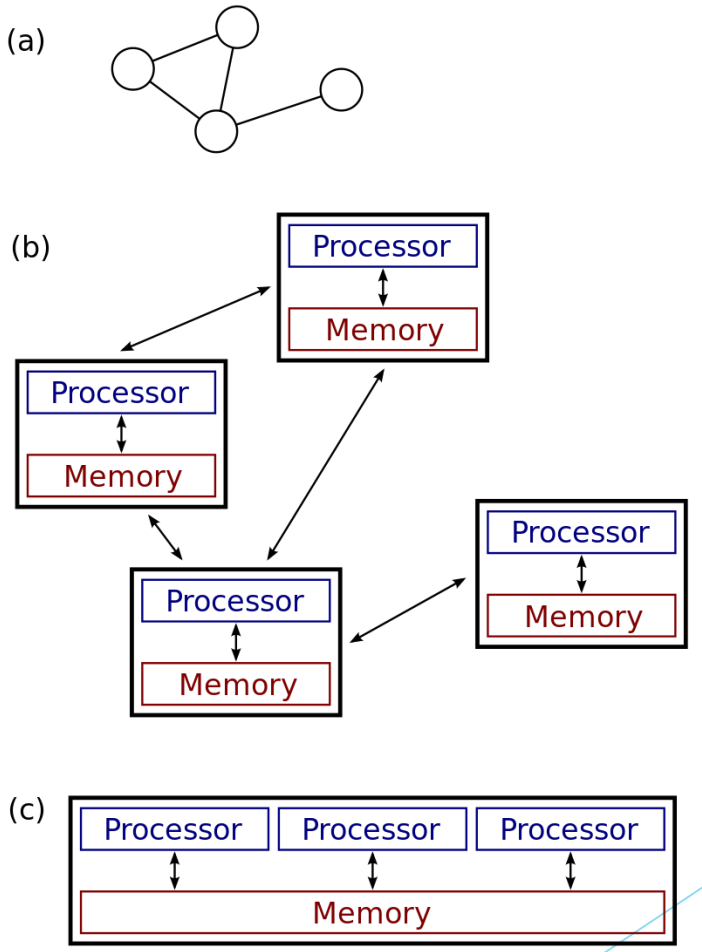

Image: CC BY-SA 3.0: Miym

Distributed Programming

- Telephone and computer networks
- Web applications
- Games with network or multiplayer options
- Redundant fault tolerant real time systems, i.e. 2oo3
- Automated industrial production lines
- Supercomputer clusters and volunteer computing, i.e. BOINC

Conclusion – Cross Platform Development

Go

- Net Package is platform independent
- gRPC is platform independent

C

- Libraries depend on OS
- Code requires OS differentiation
- Inconsistent data types across OS

Sockets - Libraries

Go

```go
import (
  "bufio"
  "fmt"
  "math/rand"
  "net"
  "os"
  "strconv"
  "strings"
  "time"
)
```

C

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef _WIN32
/* Header files for Windows */
#include <winsock2.h>

#else
/* Header files for UNIX/Linux */
#include <netinet/in.h>
#include <netdb.h>
#endif
```

Cross-Plattform C

```
#ifdef _WIN32
    SOCKET sockfd;
#else
    int sockfd;
#endif
#ifdef _WIN32
    /* TCP socket initialization with winsock library */
    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD (1, 1);
    if (WSAStartup (wVersionRequested, &wsaData) != 0){
      perror("ERROR on initialization");
      exit(1);
      }
#endif
#ifdef _WIN32
        closesocket(sockfd);
#else
        close(sockfd);
#endif
```

Conclusion – Network support

Go

- Support for IPv4 and IPv6
- Crypto/tls package shares interfaces with net package
- Clear and concise names

  „tcp“, „tcp4“, „tcp6“

C

- IPv4 and IPv6, but complicated
- TLS requires 4 libraries, new and changed functions
- Cryptic legacy names
  - AF_INET
  - AF_INET6
  - Sockaddr_in
  - Sin_addr

C Legacy structures

```c
struct sockaddr_in {
    short int          sin_family;  // Address family, AF_INET
    unsigned short int sin_port;    // Port number
    struct in_addr     sin_addr;    // Internet address
    unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
/* IPv4 implementation */
struct in_addr {
    uint32_t s_addr; // 32bit or 4byte
};
/* IPv6 implementation */
struct in6_addr {
    unsigned char   s6_addr[16];   // IPv6 address
};
```

Conclusion – Concise Syntax and strict typing reduces Errors

Go

- Strict type concept
  - TCPConn
  - Listener
  - TCPListener
  - Error
- Two return values
- Higher level of abstraction -> Parameters and Settings in one location

C

- Socket and error of type int
- One return value for both
- Parameters must be set in multiple locations

Sockets - Initialization

Go

```go
arguments := os.Args
PORT := ":" + arguments[1]
l, err := net.Listen("tcp4", PORT)
if err != nil {
    fmt.Println(err)
    return
}
```

C

```c
int sockfd, newsockfd, portno, clilen;
char buffer[256];
struct sockaddr_in serv_addr, cli_addr;
int n, pid;

/* Socket creation */
sockfd = socket(AF_INET, SOCK_STREAM, 0);

if (sockfd < 0) {
      perror("ERROR opening socket");
      exit(1);
}

/* Socket initialization */
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = 5001;
```

Sockets - Initialization

Go

```go
arguments := os.Args
PORT := ":" + arguments[1]
l, err := net.Listen("tcp4", PORT)
if err != nil {
    fmt.Println(err)
    return
}
```

C

```c
/* Socket initialization */
bzero((char *) &serv_addr, sizeof(serv_addr));
portno = 5001;

serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(portno);
```

Sockets - Initialization

Go

```go
arguments := os.Args
PORT := ":" + arguments[1]
l, err := net.Listen("tcp4", PORT)
if err != nil {
    fmt.Println(err)
    return
}
```

C

```c
/* Socket binding */
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
      perror("ERROR on binding");
      exit(1);
   }
/* Socket listening */
listen(sockfd,5);
```

Concurrent Connection Handling

Go routines

- <= 2kB memory
- No context change
  - Page table
  - Registers
- No OS call required
- Cooperative scheduling

C – fork()

- <= 1MB memory
- Context change
  - Page table copied
  - Registers stored and restored
- OS call required
- Preemptive scheduling

Connection handling

Go

```go
defer l.Close()

  for {
    c, err := l.Accept()
    if err != nil {
      fmt.Println(err)
      return
    }
    go handleConnection(c)
}
```

C

```c
listen(sockfd,5);
clilen = sizeof(cli_addr);
   while (1) {
      newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
      if (newsockfd < 0) {
         perror("ERROR on accept");
         exit(1);}
      pid = fork();
      if (pid < 0) {
         perror("ERROR on fork");
         exit(1);}
      if (pid == 0) {
         close(sockfd);
         doprocessing(newsockfd);
         exit(0);}
      else {
         close(newsockfd);}}
```

Remote Procedure Call

Go

```
service RouteGuide {
  rpc GetFeature(Point) returns (Feature) {}
  ...
}
```

```
type routeGuideServer struct {
        /* server properties */
}
...

func (s *routeGuideServer) GetFeature(ctx context.Context,
                    point *pb.Point) (*pb.Feature, error) {
        /* body of the GetFeature function must be added here */
}
...
```

C

```
 program MESSAGEPROG {
     version PRINTMESSAGEVERS {
         int PRINTMESSAGE(string) = 1;
    } = 1;
} = 0x20000001;
```

```
#include <stdio.h>
#include "msg.h"        /* msg.h generated by rpcgen */

int * printmessage_1(msg, req)
  char **msg;
  struct svc_req *req;    /* details of call */
{
  static int result;      /* must be static! */
  ...
  result = 1;
  return (&result);
}
```

Portability

Go

OS	Architecture
FreeBSD	Amd64, 386
Linux	Amd64, 386, arm, arm64, s390x, ppc64le, mips
macOS	amd64
Windows	Amd64, 386

C

- GCC support for ~50 architectures
- No OS required
- Suitable for Embedded Systems

Final Conclusion
Which one to use

Go

- Server
- High-end Clients

C

- Low-end Clients
  - Embedded Linux
- Embedded Hardware