ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

КУРСОВОЙ ПРОЕКТ

по дисциплине “Операционные системы” на тему

LiteSH

Выполнил студент Шиндель Э. Д.
Ф.И.О.

Группы ИВ-823

Работу принял ассистент Бочкарёв Б.В.
подпись

Защищена Оценка

Новосибирск – 2020

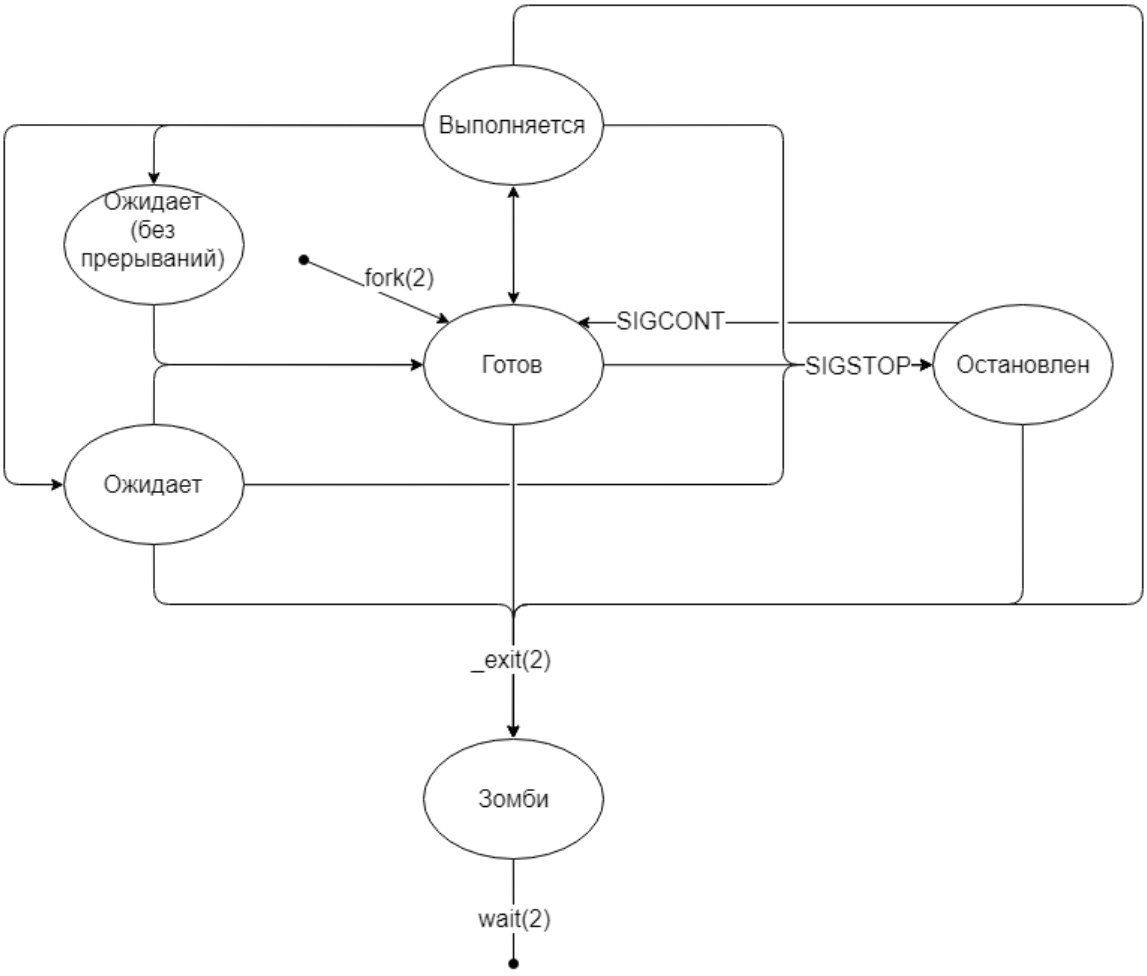
Оглавление

ВВЕДЕНИЕ

Цель курсовой работы – доработать проект LiteSH с поддержкой CI для новых версий. В проекте LiteSH реализованы следующие возможности: возможность порождать процессы и переводить их в фоновый режим, получать и обрабатывать сигналы от внешних программ. Так же в проекте присутствуют клиент-серверная структура и дополнительный функционал при помощи динамической библиотеки.

1. ПРОЦЕССЫ И ДЕМОНЫ

Процесс в ядре представляется просто как структура с множеством полей [1]. В этой структуре находятся следующие поля [2]: идентификационная информация о процессе, статус процесса, информация для планировщика, информация для организации межпроцессорного взаимодействия, ссылки и связи процесса, информация о времени исполнения и таймеры, информация об используемых процессом ресурсах файловой системы, информация о выделенном процессу адресном пространстве, контекст процесса – информация о состоянии регистров процессора, стеке и тд.



Жизненный цикл процесса

Демоны отличаются от обычных процессов только тем, что работают в не интерактивном режиме. Демоны обычно используются для выполнения сервисных функций, обслуживания запросов от других процессов.

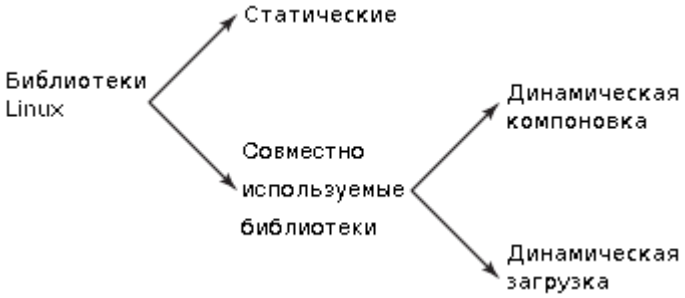
2. СОКЕТЫ

Сокет – комбинация IP адреса и номера порта, которая однозначно определяет отдельный сетевой процесс во всей глобальной сети Internet. Сокеты используются для обеспечения сетевых коммуникаций. Два сокета, один для хоста-получателя, другой для хоста-отправителя, определяют соединение для протоколов, ориентированных на установление связи, таких, как TCP [3].

Для создания сокета типа stream с протоколом TCP используется команда *s = socket(AF_INET, SOCK_STREAM, 0);*

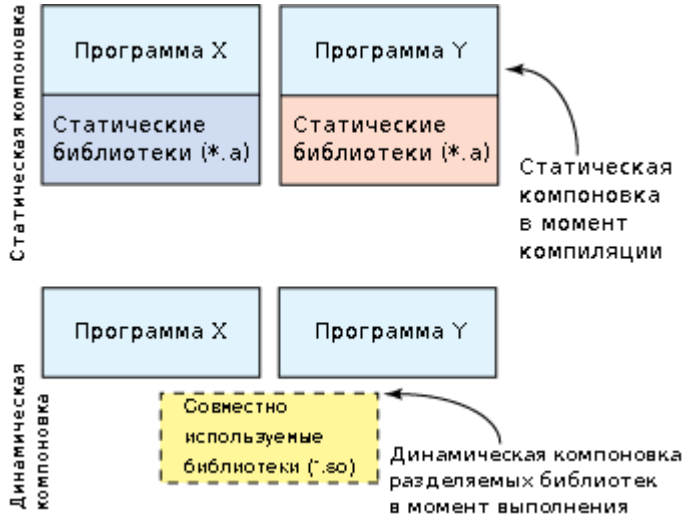
3. ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ

Динамические библиотеки загружаются после запуска приложения, а связывание происходит на этапе выполнения [4].



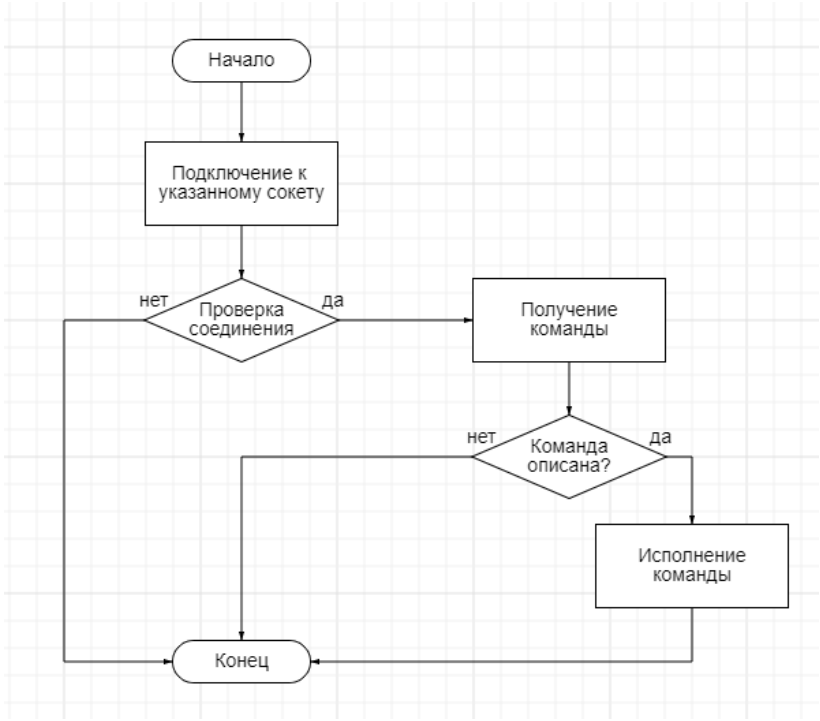
Иерархия библиотек

В программах с обширным функционалом и большим количеством библиотек рекомендуется использовать динамические библиотеки. Это позволяет снизить количество используемой памяти во время работы приложения.

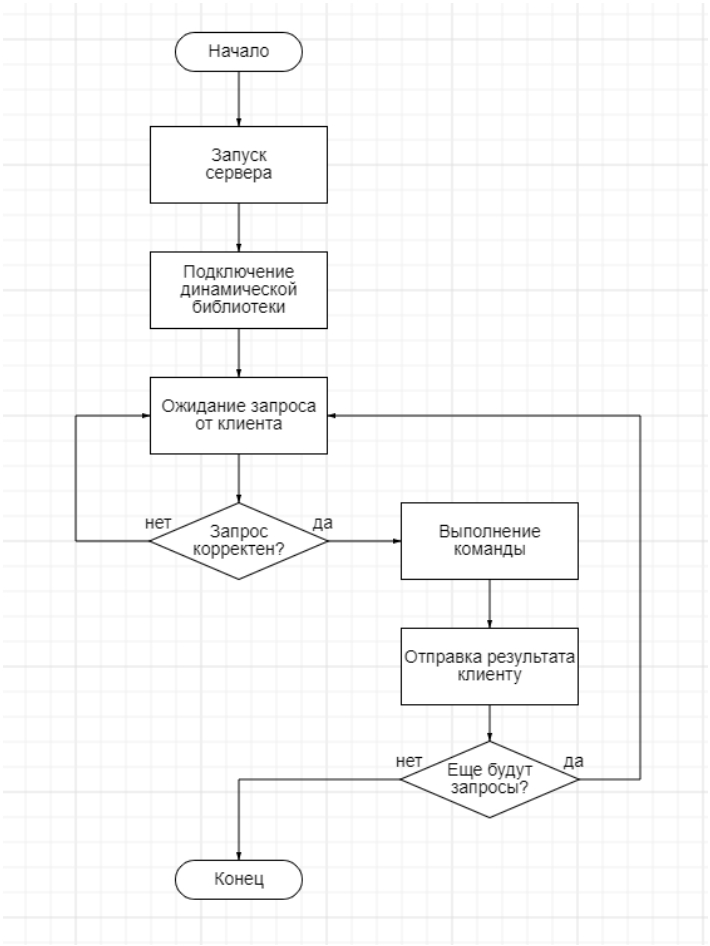


Сравнение статической и динамической компоновки

4. БЛОК-СХЕМЫ ПРОЕКТА



Блок-схема подключения сервера и клиента



Блок-схема всего алгоритма

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. Процессы в Linux [Электронный ресурс] // Хабр: Сообщество IT URL https://habr.com/ru/post/423049/ (дата обращения – 25.12.2020).
2. Процессы и демоны в Linux [Электронный ресурс] // Linux по-русски: виртуальная энциклопедия URL http://rus-linux.net/kos.php?name=/papers/proc/proc_lin.html (дата обращения – 25.12.2020)
3. Программирование сокетов [Электронный ресурс] // CodeNet: всё для программиста URL http://www.codenet.ru/progr/cpp/Sockets.php (дата обращения – 25.12.2020)
4. Анатомия динамических библиотек Linux [Электронный ресурс] // IBM URL https://www.ibm.com/developerworks/ru/library/l-dynamic-libraries/

Приложение

h.c

```
#include
<stdio.h>

            void info(char *msg) {
                        printf ("Авторы: Мяконьких Дмитрий, Шиндель Эдуард, Полищук
            Никита\n-fm - Вызов программы для работы с файловой системой\n  Пример запуска -
            ./litesh -fm\n-cp - Порождение процесса\nПример запуска - ./litesh -cp
            proc_name\n-cbp - Перевод процесса в фоновый режим\nПример запуска - ./litesh -cbp
            proc_name\n-rs - Получение сигнала от процесса\nПример запуска - ./litesh -rs\n");
                        return;
            }


            int main() {
                return 0;
            }
```

client.c

```
#include
<arpa/inet.h>
                #include <stdlib.h>
                #include <string.h>
                #include <sys/socket.h>
                #include <netinet/in.h>
                #include <unistd.h>
                #include <stdio.h>
                int main(int argc, char *argv[]) {
                    struct sockaddr_in server;
                    char ans[256], command[256], arg[256];
                    server.sin family = AF INET;
                    server.sin port = htons(2019);
                    server.sin_addr.s_addr = inet_addr("127.0.0.1");
                    int sock = socket(AF_INET,SOCK_STREAM,0);

                    if (connect(sock, (struct  sockaddr*)&server, sizeof(server)) < 0) {
                        printf("Connection failed\n");
                        return 1;
                    } else {
                        printf("Connection established\n");
                    }
                    while(strcmp(command, "exit")){
                      printf("enter command\n>");
                      scanf("%s", command);
```

```
            send(sock, command, 256,0);
            if(strcmp("-h", command) == 0){
              recv(sock, ans, 256,0);
              printf("Server send: \n%s\n", ans);
            } else if((strcmp(command, "-rs") == 0)){
              recv(sock, ans, 256,0);
              printf("Server send: \n%s\n", ans);
            } else if((strcmp(command, "-cbp") == 0) || (strcmp(command, "-cp") ==
        0)){
              recv(sock, ans, 256,0);
              printf("Server send: \n%s\n", ans);
              scanf("%s", arg);
              send(sock, arg, 256,0);
              recv(sock, ans, 256,0);
              printf("Server send: \n%s\n", ans);
            } else{
              printf("looks like a wrong command, try another one\nServer also
        said\n");
              recv(sock, ans, 256,0);
              printf("%s\n", ans);
            }
          }
          close(sock);
        }
```

main.c

```
#include
<stdio.h>
            #include <string.h>
            #include <stdlib.h>
            #include <unistd.h>
            #include <signal.h>
            #include <wait.h>

            #define SHELL "/bin/sh"

            int create_process(const char *command) {
                int status;
                pid_t pid;

                pid = fork();
                if (pid == 0) {
                    execl(SHELL, SHELL, "-c", command, NULL);
                     exit(EXIT FAILURE);
                } else if (pid < 0) {
                    status = -1;
```

```
    } else {
        if (waitpid(pid, &status, 0) != pid) {
            status = -1;
        }
    }

    return status;
}

int create_background_process(const char *command) {
    int status = 0;
    pid_t pid;

    pid = fork();
    if (pid == 0) {
        setsid();
        fclose(stdin);
        fclose(stdout);
        fclose(stderr);
        execl(SHELL, SHELL, "-c", command, NULL);
        _exit(EXIT_FAILURE);
    } else if (pid < 0) {
        status = -1;
    }

    return status;
}

int send_signal(pid_t pid, int signum) {
    if (kill(pid, signum) == -1) {
        return 1;
    }

    return 0;
}

void signal handler(int signumber) {
    printf("\nSignal - %d received successfully (Press Enter...)\n", signumber);
}

int receive_signal(int signumber) {
    if (signal(signumber, signal_handler) == SIG_ERR) {
        return 1;
    }

    return 0;
}
```

```
int main() {
    char *line = NULL;
    char *lin1 = NULL, *lin2 = NULL;
    size_t len = 0;
    char *choice = NULL;

    int exit = 0;
    while (!exit) {
        printf("<=>Welcome to LiteSH<=>\n"
               "1 create process\n"
               "2 create background process\n"
               "3 send signal\n"
               "4 receive signal\n"
               "5 help\n"
               "6 exit\n");
        printf("Enter key: ");
        getline(&choice, &len, stdin);
        choice[strlen(choice) - 1] = '\0';
        if (!strcmp(choice, "1")) {
            printf("Enter name of process: ");
            getline(&line, &len, stdin);
            line[strlen(line) - 1] = '\0';
            create_process(line);
            printf(">->->Procces over<-<-<\n");
        } else if (!strcmp(choice, "2")) {
            printf("Enter name of process: ");
            getline(&line, &len, stdin);
            line[strlen(line) - 1] = '\0';
            create_background_process(line);
        } else if (!strcmp(choice, "3")) {
            printf("Enter PID: ");
            getline(&lin1, &len, stdin);
            lin1[strlen(lin1) - 1] = '\0';
            printf("Enter Signal Number: ");
            getline(&lin2, &len, stdin);
            lin2[strlen(lin2) - 1] = '\0';
            send signal(atoi(lin1), atoi(lin2));
        } else if (!strcmp(choice, "4")) {
            printf("Enter Signal Number: ");
            getline(&line, &len, stdin);
            line[strlen(line) - 1] = '\0';
            if (receive_signal(atoi(line)) == 1) {
                printf("Failed to receive signal\n");
            }
        } else if (!strcmp(choice, "5")) {
            printf("/This programm can send and receive signals to/from
processes\n"
                   "/and create processes.\n"
```

```
                        ".To create process enter 1, next type the name of process\n"
                        " programm create process with this command.\n"
                        " (Enter name of programm on your computer)\n"
                        ".To create background process enter 2, type name\n"
                        " programm do the same things as create ordionary process,\n"
                        " but created process will running on another thread.\n"
                        " (Enter name of programm on your computer)\n"
                        ".To send signal enter 3, type Process ID, and number of\n"
                        " signal.\n"
                        " (PID: 777777, Signal: 15)\n"
                        ".To receive signal enter 4, type number of signal, when\n"
                        " process receive this signal, you will see message -\n"
                        " \"Signal - {number of signal} received successfully\"\n"
                        ".To get help enter 5.\n"
                        ".To exit enter 6.\n"
                        "Авторы: Полищук Никита, Дмитрий Мяконьких, Шиндель Эдуард\n");
                printf("Press Enter...");
                getchar();
            } else if (!strcmp(choice, "6")) {
                exit++;
            } else {
                printf("Wrong key\n");
            }
        }
        free(line);
        free(lin1);
        free(lin2);
        return 0;
    }
```

server.c

```
#include
<dirent.h>
            #include <string.h>
            #include <sys/stat.h>
            #include <sys/types.h>
            #include <sys/stat.h>
            #include <unistd.h>
            #include <sys/wait.h>
            #include <signal.h>
            #include <dlfcn.h>
            #include <arpa/inet.h>
            #include <stdlib.h>
            #include <sys/socket.h>
            #include <netinet/in.h>
            #include <stdio.h>
```

```
void sigHandler(int signum) {
  printf("Signal %d was handled successfully\n", signum);
  _exit(EXIT_FAILURE);
}

int main(int argc, char** argv) {
  int pid, ppid, rez;
  char *error;
  void *handle = dlopen("/home/_chariot/os/lab4/dlib/libhelp.so", RTLD_NOW);
    if (!handle) {
        fputs (dlerror(), stderr);
        exit (-1);
    }

    if ((error = dlerror()) != NULL) {
        fprintf (stderr, "%s\n", error);
        exit (-1);;
    }
    typedef void (*func_info)();
    func_info info = (func_info)dlsym(handle,"info");
    if ((error = dlerror()) != NULL) {
        fprintf (stderr, "%s\n", error);
        exit(-1);
    }
  char command[256], arg[256], ans[256];
    struct sockaddr_in server, client;
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    int enable = 1;
    if (setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)
        perror("setsockopt(SO_REUSEADDR) failed");
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(2019);
    bind(sock, (struct sockaddr *) &server, sizeof(server));
    listen(sock, 5);
    int newsock, size;
      socklen_t clnlen;
      int numh,num;
      newsock = accept(sock, (struct sockaddr*)&client, &clnlen);
      printf("New client: %s\n",inet_ntoa(client.sin_addr));
    while(1){
        recv(newsock, command, 256, 0);

  if (strcmp("-cp", command) == 0) {

    int status;
    pid_t pidt;
    char invite[30] = "Proc name?\n>";
```

```
    send(newsock, invite, 30, 0);
    recv(newsock, arg, 256, 0);
    pidt = fork();
    if (pidt == -1) {
      strcpy(ans, "Process creation failure\n");
      send(newsock, ans, 256, 0);
    } else if (pidt == 0) {
      pid_t pid, ppid;
      pid = getpid();
      ppid = getppid();
      sprintf(
          ans, "Process-child was created successfully\n %d - child's number\n
%d parents number\n",
          pid, ppid);
          send(newsock, ans, 256, 0);
      if (execl(arg, arg, NULL) == -1) {
              perror("exec");
               _exit(EXIT_FAILURE);
      }
    }
  } else if (strcmp("-cbp", command) == 0) {
    int ind = 0;
    pid_t pidt;
    char invite[40] = "Proc name?\n>";
    send(newsock, invite, 30, 0);
    recv(newsock, arg, 256, 0);
    pidt = fork();
    if (pidt == -1) {
      strcpy(ans, "Process creation failure\n");
      send(newsock, ans, 256, 0);
    } else if (pidt == 0) {
      setsid();
      pid = getpid();
      ppid = getppid();
      chdir("/");
      execl(arg, arg, NULL);
      wait(&ind);
      sprintf(ans, "Background process was created \npid = %d\nppid = %d",
pid, ppid);
      send(newsock, ans, 256, 0);
    }

  } else if (strcmp("-rs", command) == 0) {
    signal(SIGINT , sigHandler);
    pid_t pidt;
    int pid;
    pidt = fork();
   if (pidt == 0) {
```

```
        pid_t pid, ppid;
        pid = getpid();
        ppid = getppid();
        kill(pid, SIGINT);
      }
      strcpy(ans, "Signal was catched\n");
      send(newsock, ans, 256, 0);
    } else if(strcmp("-h", command) == 0){
      info(ans);
      strcpy(ans, "You can check help on serverside\n");
      send(newsock, ans, 256, 0);
    } else{
      strcpy(ans, "Wrong command\n");
      send(newsock, ans, 256, 0);
    }

  }
  }
```

Makefile

```
LDFLAGS=-
ldl

.PHONY: clean

all: serv client libhelp.so

#serv: builds/serv
	#gcc builds/serv -I/libhelp.so -o serv

builds/serv.o: src/server.c
	gcc -c src/server.c -ldl -o builds/serv.o

client: src/client.c
	gcc src/client.c -o client

libhelp.so: dlib/help.o
	gcc ./dlib/help.o -shared -o ./dlib/libhelp.so

dlib/help.o: dlib/h.c
	gcc -c -fPIC ./dlib/h.c -o ./dlib/help.o

clean:
	rm serv client

serv: builds/serv
```

```
gcc src/server.c -L./dlib/libhelp.so -o serv -ldl
```

litesh.service

```
[Unit]
        Description=LiteSH_daemon


        [Service]
        Type=simple
        User=root
        Group=root
        ExecStart=/home/_chariot/os/lab4/serv
        ExecReload=/home/_chariot/os/lab4/serv
        KillMode=control-group
        TimeoutStartSec=120
        OOMScoreAdjust=-1000


        [Install]
        WantedBy=multi-user.target
```