Основы построения файловых систем

Разное о C++

```
char *tmp_proc=malloc(sizeof(char)*sizeof(directory_proc));
char *tmp_stat=malloc(sizeof(char)*sizeof(stat));

strcpy(tmp_proc,directory_proc);
strcpy(tmp_stat,stat);

char *partly_way = strcat(tmp_proc, entry->d_name);
char *way_to_stat = strcat(partly_way, tmp_stat);

file_status = fopen(way_to_stat, "r");
```

1. Функция strcat() считает, что за первой строкой достаточно места, чтобы дописать вторую.

Разное о C++

```
char *tmp_proc=malloc(sizeof(char)*sizeof(directory_proc));
char *tmp_stat=malloc(sizeof(char)*sizeof(stat));

strcpy(tmp_proc,directory_proc);
strcpy(tmp_stat,stat);

char *partly_way = strcat(tmp_proc, entry->d_name);
char *way_to_stat = strcat(partly_way, tmp_stat);

file_status = fopen(way_to_stat, "r");
```

1. Функция strcat() считает, что за первой строкой достаточно места, чтобы дописать вторую.
2. sizeof(char) = 1 согласно C99, section 6.5.3.4

Разное о C++

```
char *tmp_proc=malloc(sizeof(char)*sizeof(directory_proc));
char *tmp_stat=malloc(sizeof(char)*sizeof(stat));

strcpy(tmp_proc,directory_proc);
strcpy(tmp_stat,stat);

char *partly_way = strcat(tmp_proc, entry->d_name);
char *way_to_stat = strcat(partly_way, tmp_stat);

file_status = fopen(way_to_stat, "r");
```

Правильный и простой способ собрать строку:

```
char path[64];
snprintf(path, sizeof(path), “/proc/%u/stat”, pid);
```

Разное о C++

```
...

char tmp_proc [256];

snprintf(tmp_proc, 256, "%s%s%s",
         directory_proc, entry->d_name, stat);

...
```

Правильный и простой способ собрать строку:

```
char path[64];
snprintf(path, sizeof(path), “/proc/%u/stat”, pid);
```

Разное о C++

```
char *tmp_proc=malloc(sizeof(char)*sizeof(directory_proc));
char *tmp_stat=malloc(sizeof(char)*sizeof(stat));

strcpy(tmp_proc,directory_proc);
strcpy(tmp_stat,stat);

char *partly_way = strcat(tmp_proc, entry->d_name);
char *way_to_stat = strcat(partly_way, tmp_stat);

file_status = fopen(way_to_stat, "r");
```

Правильный и простой способ собрать строку:

```
char path[64];
snprintf(path, sizeof(path), “/proc/%u/stat”, pid);
```

Что делать, если длина строки наперёд неизвестна?

```
va_list va0;
int len;
char *res;

va_copy(va0, va);
len = vsnprintf(NULL, 0, fmt, va0);
va_end(va0);

res = malloc(len + 1);

va_copy(va0, va);
vsnprintf(res, len, fmt, va0);
va_end(va0);
```

Разное о C++

```
struct stat_file {
    int pid;
    char name[30];
    char status;
};

...

struct stat_file statFile;
fscanf(file_status, "%d %s %c",
       &statFile.pid, statFile.name, &statFile.status);
```

Что будет, если имя процесса в файле /proc/PID/stat окажется длиннее 30 символов?

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        try{
                std::ifstream file(fPath);
                file >> Name;
                file.close();
        }
        catch ( const std::ifstream::failure& error){
           throw error;
        }
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

1. Блок catch() без какой-либо обработки ошибки лишён смысла.
2. “throw error”, в отличие от ”throw”, не прокинет исключение error дальше, а сделает его копию и выше по стеку выбросит именно копию.

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

Сколько вызовов malloc() спрятано в этой строке?

```
char path[64];
snprintf(path, “/proc/%u/comm”, pid);
```

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

IO в конструкторе:

1. Может зависать на длительное время (как отменить вызов конструктора?)

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

IO в конструкторе:

1. Может зависать на длительное время (как отменить вызов конструктора?),
2. Нет разумного способа сообщить об ошибке (нет, исключения не годятся, поскольку конструкторы могут неявно зваться из многих мест, которые попробуй отследить),

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

IO в конструкторе:

1. Может зависать на длительное время (как отменить вызов конструктора?),
2. Нет разумного способа сообщить об ошибке (нет, исключения не годятся, поскольку конструкторы могут неявно зваться из многих мест, которые попробуй отследить),
3. Нет разумного способа сообщить об ошибке вида “IO выполнилось частично”,

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

IO в конструкторе:

1. Может зависать на длительное время (как отменить вызов конструктора?),
2. Нет разумного способа сообщить об ошибке (нет, исключения не годятся, поскольку конструкторы могут неявно зваться из многих мест, которые попробуй отследить),
3. Нет разумного способа сообщить об ошибке вида “IO выполнилось частично”,
4. Проблемы для компилятора: конструктор не noexcept, такой код тяжелее оптимизировать.

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

IO в деструкторе: эта затея ещё хуже, чем IO в конструкторе, поскольку из деструктора бросать исключения нельзя.

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        ...
    }

    void print_pid_name(){...}

private:
    int Pid;
    std::string Name;
};

...

int main()
{
    std::vector<Process> processes;
    ...
}
```

Что делает конструктор по умолчанию?

Что произойдёт после вызова processes.resize(10)?

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        ...
    }

    void print_pid_name(){...}

private:
    int Pid;
    std::string Name;
};

...

int main()
{
    std::vector<Process> processes;
    ...
}
```

Что делает конструктор по умолчанию?

Что произойдёт после вызова
processes.resize(10)?

Эксперимент:

```
std::aligned_storage_t<sizeof(Process)> buf;
memset(&buf, 0x55, sizeof(buf));

Process *p = new (&buf) Process();
p->print_pid_name();
```

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        ...
    }

    void print_pid_name(){...}

private:
    int Pid;
    std::string Name;
};

...

int main()
{
    std::vector<Process> processes;
    ...
}
```

Что делает конструктор по умолчанию?

Что произойдёт после вызова
processes.resize(10)?

Эксперимент:

```
std::aligned_storage_t<sizeof(Process)> buf;
memset(&buf, 0x55, sizeof(buf));

Process *p = new (&buf) Process();
p->print_pid_name();

~~~~

p->Pid = 0x55555555;
```

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

Const-correctness: методы, которые не меняют состояние объекта, должны быть помечены как const:

```
void print_pid_name() const {
    ...
}
```

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

Разное о C++

```
class Process {
public:

    explicit Process(int pid) :
            Pid(pid)

    {
        std::string fPath = "/proc/" + std::to_string(pid) + "/comm";

        std::ifstream file(fPath);
        file >> Name;
        file.close();
    }

    void print_pid_name(){
        std::cout << Pid << "\t\t\t" << Name << std::endl;
        }

private:
    int Pid;
    std::string Name;
};
```

Не надо смешивать в одном классе модель данных (в данном случае, сведения о PID и имени процесса) и способ их представления пользователю.

См. Model-View-Controller.

Разное о C++

```
class Process {
public:

    explicit Process(int pid);
    ...
};

...

int main()
{
    std::vector<Process> processes;
    ...
    while ((ent = readdir (dir)) != nullptr) {
        pid = ent->d_name;
        int pid_int = std::stoi(pid);
        processes.emplace_back(Process(pid_int));
    }
}
```

В C++11 у контейнеров появились методы emplace_*(). Их цель – создавать объекты-элементы контейнера сразу в том месте памяти, где контейнер их будет хранить, а не копировать эти объекты в контейнер.

```
while (( ... )) {
    processes.emplace_back(pid_int);
}
```

Разное о C++

```
int main(int argc, char** argv)
{
    DIR*           proc_dir = opendir(proc_dir_name);
    struct dirent* dir_entry = NULL;

    while((dir_entry = readdir(proc_dir)) != NULL)
    {
        if(check_if_name_is_pid(dir_entry->d_name))
            print_openned_files_info_for_pid(dir_entry->d_name);
    }
}
```

Ввод-вывод может завершаться неудачей. Следить за этим надо обязательно.

Разное о C++

```
const size_t fd_file_path_size = 1024;
const size_t fd_name_size = 16;

struct fd_info
{
        struct stat_info* pid_stat_info;
        char* file_path;
        char* fd;
};

struct fd_info* allocate_fd_info()
{
    struct fd_info* fd_info = (struct fd_info*)calloc(1, sizeof(struct fd_info));

    fd_info->file_path = (char*)calloc(fd_file_path_size, sizeof(char));
    fd_info->fd = (char*)calloc(fd_name_size, sizeof(char));

    return fd_info;
}

void free_fd_info()
{
    ...
}

void fill_fd_info(struct fd_info* fd_info, ...)
{
    ...
}
```

```
struct fd_info* allocate_fd_info()
{
  struct fd_info *i = malloc(sizeof(*i));
  i->file_path = malloc(fd_file_path_size);
  i->fd = malloc(fd_name_size);

  return i;
}
```

В C, в отличие от C++, в присваиваниях void * автоматически преобразуется к любому указателю.

Вместо calloc(), если уж требуется заполнить память нулями, заведите zmalloc() или, что лучше, xzmalloc(), который ещё и будет убивать программу в случае неудачи.

Разное о C++

```
const size_t fd_file_path_size = 1024;
const size_t fd_name_size = 16;

struct fd_info
{
        struct stat_info* pid_stat_info;
        char* file_path;
        char* fd;
};

struct fd_info* allocate_fd_info()
{
    struct fd_info* fd_info = (struct fd_info*)calloc(1, sizeof(struct fd_info));

    fd_info->file_path = (char*)calloc(fd_file_path_size, sizeof(char));
    fd_info->fd = (char*)calloc(fd_name_size, sizeof(char));

    return fd_info;
}

void free_fd_info(struct fd_info *i)
{
    ...
}

void fill_fd_info(struct fd_info* fd_info, ...)
{
    ...
}
```

```
struct fd_info
{
  struct stat_info *pid_stat_info;
  char file_path[1024];
  char fd[16];
};

struct fd_info* allocate_fd_info()
{
  /* no longer needed */
  /* let us keep it just for now */
  struct fd_info *i = malloc(sizeof(*i));
  return i;
}

void free_fd_info(struct fd_info *i)
{
  /* no longer needed */
}
```

Разное о C++

```
struct fd_info {...};

struct fd_info* allocate_fd_info() {...}
void free_fd_info(struct fd_info *i) {...}
void fill_fd_info(struct fd_info* fd_info, ...) {...}

void print_openned_files_info_for_pid(const char* pid)
{
    struct fd_info* fd_info = allocate_fd_info();

    ...

    while((e = readdir(fd_dir)) != NULL)
    {
        if((strcmp(".", e->d_name) && strcmp("..", e->d_name)))
        {
            fill_fd_info(fd_info, e, fd_dir_path);
            print_fd_info(fd_info);
        }
    }

    ...
}
```

```
void print_opened_file(const char *pid)
{
    ...

    while((e = readdir(fd_dir)) != NULL) {
        struct fd_info i;
        fill_fd_info(&i, e, fd_dir_path);
        print_fd_info(&i);
    }

    ...
}
```

Разное о C++

```
while ((entry = readdir(dir_ptr)) != NULL) {
    lstat(entry->d_name, &status_buffer);
    if(S_ISDIR(status_buffer.st_mode)) {
        ...
        if (status_buffer.st_uid == getuid()) {
            strcpy(path, PROC_DIR);
            strcat(path, entry->d_name);
            strcat(path, "/status");

            file_ptr = fopen(path, "r");
            if(file_ptr == NULL)
            {
                printf("Error opening file %s\n", path);
                return ERROR;
            }

            ...
        }
    }
}
```

1. Проверка ошибок.
2. Утечка открытого dir_ptr.

Разное о C++

```
while ((entry = readdir(dir_ptr)) != NULL) {
    lstat(entry->d_name, &status_buffer);
    if(S_ISDIR(status_buffer.st_mode)) {
        ...
        if (status_buffer.st_uid == getuid()) {
            ...
        }
    }
}
```

```
while ((e = readdir(dir_ptr)) != NULL) {
    int r = lstat(e->d_name, &stat);
    if (r < 0) {
        fprintf(stderr, “failed to stat %s”, e->d_name);
        continue;
    }

    if (!S_ISDIR(stat.st_mode))
        continue;
    if (stat.st_uid != getuid())
        continue;

    ...

    if (file_ptr == NULL) {
        fprintf(stderr, “blah-blah-blah”);
        goto out;
    }
}

out:
    closedir(dir_ptr);
```

Разное о C++

```
char path[sizeof("/proc/") + 5 + //pid max -- 5-digit number,
          sizeof("/fd/") + 10];  //inode max number is 10-digit
                                 //number on ext 2/3/4
snprintf(path, sizeof(path), "/proc/%s/fd/%s",
         proc_entry->d_name, fd_entry->d_name);

lstat(path, &st);
unsigned int buffer_size = st.st_size + 1;

char *buffer = calloc(buffer_size, sizeof(char));
readlink(path, buffer, buffer_size);
printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
free(buffer);
```

1. PID может быть больше 65535. Это настраивается с помощью sysctl ”kernel.pid_max”.

Разное о C++

```
char path[sizeof("/proc/") + 5 + //pid max -- 5-digit number,
          sizeof("/fd/") + 10];  //inode max number is 10-digit
                                 //number on ext 2/3/4
snprintf(path, sizeof(path), "/proc/%s/fd/%s",
         proc_entry->d_name, fd_entry->d_name);

lstat(path, &st);
unsigned int buffer_size = st.st_size + 1;

char *buffer = calloc(buffer_size, sizeof(char));
readlink(path, buffer, buffer_size);
printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
free(buffer);
```

1. PID может быть больше 65535. Это настраивается с помощью sysctl ”kernel.pid_max”.
2. Слишком сложно, хватит так:
   char path[64];

Разное о C++

```
char path[sizeof("/proc/") + 5 + //pid max -- 5-digit number,
          sizeof("/fd/") + 10];  //inode max number is 10-digit
                                 //number on ext 2/3/4
snprintf(path, sizeof(path), "/proc/%s/fd/%s",
         proc_entry->d_name, fd_entry->d_name);

lstat(path, &st);
unsigned int buffer_size = st.st_size + 1;

char *buffer = calloc(buffer_size, sizeof(char));
readlink(path, buffer, buffer_size);
printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
free(buffer);
```

1. PID может быть больше 65535. Это настраивается с помощью sysctl ”kernel.pid_max”.
2. Слишком сложно, хватит так:
   char path[64];
3. Здесь есть “race condition”

Разное о C++

```
char path[sizeof("/proc/") + 5 + //pid max -- 5-digit number,
          sizeof("/fd/") + 10];  //inode max number is 10-digit
                                 //number on ext 2/3/4
snprintf(path, sizeof(path), "/proc/%s/fd/%s",
         proc_entry->d_name, fd_entry->d_name);

lstat(path, &st);
unsigned int buffer_size = st.st_size + 1;

char *buffer = calloc(buffer_size, sizeof(char));
readlink(path, buffer, buffer_size);
printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
free(buffer);
```

1. PID может быть больше 65535. Это настраивается с помощью sysctl ”kernel.pid_max”.
2. Слишком сложно, хватит так:
   char path[64];
3. Здесь есть “race condition”:
   1. lsof обнаруживает символическую ссылку на открытый файл и узнаёт её длину,
   2. процесс, состояние которого изучает lsof, закрывает файл и открывает другой, а номер файлового дескриптора переиспользуется,
   3. lsof делает readlink() на ссылку, длина которой поменялась.

Разное о C++

```
char path[sizeof("/proc/") + 5 + //pid max -- 5-digit number,
          sizeof("/fd/") + 10];  //inode max number is 10-digit
                                 //number on ext 2/3/4
snprintf(path, sizeof(path), "/proc/%s/fd/%s",
         proc_entry->d_name, fd_entry->d_name);

lstat(path, &st);
unsigned int buffer_size = st.st_size + 1;

char *buffer = calloc(buffer_size, sizeof(char));
readlink(path, buffer, buffer_size);
printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
free(buffer);
```

1. PID может быть больше 65535. Это настраивается с помощью sysctl ”kernel.pid_max”.
2. Слишком сложно, хватит так:
   char path[64];
3. На самом деле, тут другая проблема: размер символических ссылок в /proc не имеет никакого отношения к их содержимому. Он всегда равен 64.

Пример использования valgrind:

```
$ valgrind --tool=memcheck --leak-check=full --show-leak-kinds=all --track-origins=yes --num-callers=32 ./a.out

==2628== Memcheck, a memory error detector
==2628== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2628== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==2628== Command: ./a.out
==2628==
==2628== Invalid read of size 1
==2628==    at 0x4E82EA9: vfprintf (vfprintf.c:1635)
==2628==    by 0x4E892C8: printf (printf.c:34)
==2628==    by 0x4009E1: main (main.c:45)
==2628==  Address 0x5213129 is 0 bytes after a block of size 9 alloc'd
==2628==    at 0x4C2B955: calloc (vg_replace_malloc.c:711)
==2628==    by 0x4009A6: main (main.c:43)
==2628==

~~~~

42                unsigned int buffer_size = status_buffer.st_size + 1; // one for null-terminate symbol
43                char *buffer = calloc(buffer_size, sizeof(char));//calloc makes string null-terminated
44                readlink(directory_fd, buffer, buffer_size);
45                printf("[pid] - %s, opened file - %s\n", proc_entry->d_name, buffer);
```