```c
#include <dirent.h>
#include <linux/limits.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include "const.h"
#define BUF_SIZE 10000
#define NO_ACCESS 1
#define BAD_ARGS 2


void print_cmdline(const int pid) {
    char path[PATH_MAX];
    snprintf(path, PATH_MAX, "/proc/%d/cmdline", pid);
    FILE *file = fopen(path, "r");

    char buf[BUF_SIZE];
    int len = fread(buf, 1, BUF_SIZE, file);
    buf[len - 1] = 0;
    printf("\nCMDLINE:\n%s\n", buf);

    fclose(file);
}

void print_environ(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/environ", pid);
    FILE *file = fopen(pathToOpen, "r");

    int len;
    char buf[BUF_SIZE];
    printf("\nENVIRON:\n");
    while ((len = fread(buf, 1, BUF_SIZE, file)) > 0) {
        for (int i = 0; i < len; i++)
            if (!buf[i])
                buf[i] = '\n';
        buf[len - 1] = '\n';
        printf("%s", buf);
    }

    fclose(file);
}

void print_fd(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/fd/", pid);
    DIR *dir = opendir(pathToOpen);

    printf("\nFD:\n");
```

```
    struct dirent *readDir;
    char string[PATH_MAX];
    char path[BUF_SIZE] = {'\0'};
    while ((readDir = readdir(dir)) != NULL) {
        if ((strcmp(readDir->d_name, ".") != 0) &&
(strcmp(readDir->d_name, "..") != 0)) {
            sprintf(path, "%s%s", pathToOpen, readDir->d_name);
            int _read_len = readlink(path, string, PATH_MAX);
            printf("%s -> %s\n", readDir->d_name, string);
        }
    }

    closedir(dir);
}

void print_stat(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/stat", pid);
    char buf[BUF_SIZE];

    FILE *file = fopen(pathToOpen, "r");
    fread(buf, 1, BUF_SIZE, file);
    char *token = strtok(buf, " ");

    printf("\nSTAT: \n");

    for (int i = 0; token != NULL; i++) {
        printf(NO_DESCR[i], token);
        token = strtok(NULL, " ");
    }

    fclose(file);
}

void print_statm(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/statm", pid);
    FILE *file = fopen(pathToOpen, "r");
    char buf[BUF_SIZE];
    fread(buf, 1, BUF_SIZE, file);

    char *token = strtok(buf, " ");
    printf("\nSTATM: \n");
    for (int i = 0; token != NULL; i++) {
        printf(STATM_PATTERNS[i], token);
        token = strtok(NULL, " ");
    }
    fclose(file);
}

void print_cwd(const int pid) {
    char pathToOpen[PATH_MAX];
```

```
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/cwd", pid);
    char buf[BUF_SIZE] = {'\0'};
    int _read_len = readlink(pathToOpen, buf, BUF_SIZE);
    printf("\nCWD:\n");
    printf("%s\n", buf);
}

void print_io(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/io", pid);
    char buf[BUF_SIZE] = {'\0'};
    FILE *file = fopen(pathToOpen, "r");
    printf("\nIO:\n");
    int lengthOfRead;
    while ((lengthOfRead = fread(buf, 1, BUF_SIZE, file))) {
        buf[lengthOfRead] = '\0';
        printf("%s\n", buf);
    }
    fclose(file);
}

void print_comm(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/comm", pid);
    char buf[BUF_SIZE] = {'\0'};
    FILE *file = fopen(pathToOpen, "r");
    printf("\nCOMM:\n");
    int lengthOfRead;
    while ((lengthOfRead = fread(buf, 1, BUF_SIZE, file))) {
        buf[lengthOfRead] = '\0';
        printf("%s\n", buf);
    }
    fclose(file);
}

void print_exe(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/exe", pid);
    char buf[BUF_SIZE] = {'\0'};
    int _read_len = readlink(pathToOpen, buf, BUF_SIZE);
    printf("\nEXE:\n");
    printf("%s\n", buf);
}

void print_maps(const int pid) {
    char *line;
    int start_addr, end_addr, page_size = 4096;
    size_t line_size;
    ssize_t line_length;

    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/maps", pid);
```

```
    char buf[BUF_SIZE] = {'\0'};
    FILE *file = fopen(pathToOpen, "r");
    printf("\nMAPS:\n");
    int lengthOfRead;
    do
    {
        line_length = getline(&line, &line_size, file);
        if (!feof(file) && line_length == -1)
        {
            perror("getline():");
            fclose(file);
            free(line);
            exit(1);
        }
        sscanf(line, "%x-%x", &start_addr, &end_addr);
        printf("%d\t%s", (end_addr - start_addr) / page_size,
line);
    } while (!feof(file));
    fclose(file);
}

void print_file(const int num) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/maps", num);
    char buf[BUF_SIZE] = {'\0'};
    FILE *file = fopen(pathToOpen, "r");
    int lengthOfRead;
    while ((lengthOfRead = fread(buf, 1, BUF_SIZE, file))) {
        buf[lengthOfRead] = '\0';
        printf("%s\n", buf);
    }
    fclose(file);
}

void print_task(const int pid) {
    DIR *d;
    struct dirent *dir;

    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/task", pid);

    printf("\nTASK:\n");

    d = opendir(pathToOpen);
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            printf("%s\n", dir->d_name);
        }
        closedir(d);
    }
}
```

```c
void print_pagemap(const int pid) {
    DIR *d;
    struct dirent *dir;

    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/pagemap", pid);

    printf("\nPAGEMAP:\n");

    d = opendir(pathToOpen);
    if (d) {
        while ((dir = readdir(d)) != NULL) {
            printf("%s\n", dir->d_name);
        }
        closedir(d);
    }
}

void print_root(const int pid) {
    char pathToOpen[PATH_MAX];
    snprintf(pathToOpen, PATH_MAX, "/proc/%d/root", pid);
    char buf[BUF_SIZE] = {'\0'};
    int _read_len = readlink(pathToOpen, buf, BUF_SIZE);
    printf("\nROOT:\n");
    printf("%s\n", buf);
}

int get_pid(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Введите pid процесса для исследования\n");
        exit(BAD_ARGS);
    }

    int pid = atoi(argv[1]);
    char check_dir[PATH_MAX];
    snprintf(check_dir, PATH_MAX, "/proc/%d", pid);
    if (!pid || access(check_dir, F_OK)) {
        printf("Директории для введенного pid не найдено\n");
        exit(NO_ACCESS);
    }

    return pid;
}


int main(int argc, char *argv[]) {
    int pid = get_pid(argc, argv);

    print_cmdline(pid);
    print_fd(pid);
    print_environ(pid);
    print_stat(pid);
```

```
    print_statm(pid);
    print_cwd(pid);
    print_exe(pid);
    print_maps(pid);
    print_io(pid);
    print_comm(pid);
    print_root(pid);
    print_task(pid);
    print_pagemap(pid);

    return 0;
}
```