Betriebs- und Kommunikationssysteme | Zettel 2

Studenten: Evghenii Orenciuc, Jonathan Rex
Tutor: Abraham Söyler

Aufgabe 1

Begriffe

Interrupts:

```
An interrupt is a hardware mechanism that enables CPU to detect
that a device needs its attention.

Interrupt becomes inefficient when devices keep on interrupting the CPU repeatedly.
```

Polling:

```
Polling is a protocol that notifies CPU that a device needs its attention.
It's asking the I/O device whether it needs CPU processing.

Polling becomes inefficient when CPU rarely finds a device ready for service.
```

Source:
https://techdifferences.com/difference-between-interrupt-and-polling-in-os.html

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
I0	R	R	R	R													
I1		w	w	w	w	w	w	w	w	w	w	w	w	w	R		
I2		w	w	w	w	w	w	w	w	R	R						
I3			w	w	R	R	R										
I4					w	w	w	w	w	w	w	R	R	R			
I5						w	w	R	R								
I6							w	w	w	w	w	w	w	w	w	R	

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
I0	R	w	w	R	w	w	w	w	w	w	w	R	R				
I1		w	w	w	w	w	w	w	w	w	w	w	w	R			
I2		R	w	w	w	w	w	R									
I3			R	R	R												
I4					w	w	w	w	R	R	R						
I5						R	R										
I6							w	w	w	w	w	w	w	w	R		

Aufgabe 2

Fehlermeldungen werden in die Konsole ausgegeben. Da die Anwendung von <stdio.h> ist nicht moeglich, deswegen werden alle Nachrichten mit Hilfe von der Funktion

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

in die Konsole mit standard Konsole-File-Descriptor: **2** geschrieben.

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <dirent.h>

#define BUFFER_SIZE 1024
#define CONSOLE_FILE_DESCRIPTOR 2
#define TRASHCAN_DIRECTORY_NAME ".ti3_trashcan/"

int move_file_to_trash(char *file_name);
int recover_file_from_trash(char *file_name);
int delete_file_from_trash(char *file_name);
int list_files_in_trash();

int contains_move_to_trash_argument(int argc, char **argv) {
    return argv[1][0] == '-' && argv[1][1] == 't' && argc == 3 && strlen(argv[2]) > 0;
}

int contains_recover_file_from_trash_argument(int argc, char **argv) {
    return argv[1][0] == '-' && argv[1][1] == 'r' && argc == 3 && strlen(argv[2]) > 0;
}

int contains_list_files_in_trash_argument(int argc, char **argv) {
    return argv[1][0] == '-' && argv[1][1] == 'l' && argc == 2;
}

int containst_delete_file_from_trash_argument(int argc, char **argv) {
    return argv[1][0] == '-' && argv[1][1] == 'f' && argc == 3 && strlen(argv[2]) > 0;
}

int print_err_and_return_err_code(char *err) {
    write(CONSOLE_FILE_DESCRIPTOR, "Error: ", 7);
    write(CONSOLE_FILE_DESCRIPTOR, err, strlen(err));
    write(CONSOLE_FILE_DESCRIPTOR, "\n", 1);
    return -1;
}

int main(int argc, char **argv) {
    if (argc == 1) return print_err_and_return_err_code("Not enough arguments!");

    mkdir(TRASHCAN_DIRECTORY_NAME, 0777);

    if (contains_move_to_trash_argument(argc, argv)) {
        return move_file_to_trash(argv[2]);
    } else if (contains_list_files_in_trash_argument(argc, argv)){
        return list_files_in_trash();
    } else if (contains_recover_file_from_trash_argument(argc, argv)) {
        return recover_file_from_trash(argv[2]);
    } else if (containst_delete_file_from_trash_argument(argc, argv)) {
        return delete_file_from_trash(argv[2]);
    } else {
        return print_err_and_return_err_code("Invalid arguments passed!");
    }
}

int copy(char *sourcename, char *targetname) {
    int source_fd = open(sourcename, O_RDONLY);
    if (source_fd == -1) return print_err_and_return_err_code(strerror(errno));

    int target_fd = open(targetname, O_CREAT | O_EXCL | O_WRONLY, 0644);
    if (target_fd == -1) return print_err_and_return_err_code(strerror(errno));

    char buffer[BUFFER_SIZE];
    ssize_t chars_read = 0;

    while((chars_read = read(source_fd, buffer, BUFFER_SIZE)) > 0) {
        chars_read = write(target_fd, buffer, chars_read);
        if (chars_read == -1) return print_err_and_return_err_code(strerror(errno));
    }

    if (close(source_fd) == -1) return print_err_and_return_err_code(strerror(errno));
    if (close(target_fd) == -1) return print_err_and_return_err_code(strerror(errno));

    return 0;
}

char *get_trashcan_relative_file_path(char *file_name) {
    char *trash_file_path = malloc(strlen(TRASHCAN_DIRECTORY_NAME) + strlen(file_name) + 1);
    strcpy(trash_file_path, TRASHCAN_DIRECTORY_NAME);
    strcat(trash_file_path, file_name);

    return trash_file_path;
}

int move_file_to_trash(char *file_name) {
    char *trash_file_path = get_trashcan_relative_file_path(file_name);

    int result = copy(file_name, trash_file_path);
    if (result != -1) {
        result = unlink(file_name);
    }

    free(trash_file_path);

    return (result == -1) ? print_err_and_return_err_code(strerror(errno)) : 0;
}

int list_files_in_trash() {
    DIR *dir_fd = opendir(TRASHCAN_DIRECTORY_NAME);
    if (!dir_fd) return print_err_and_return_err_code(strerror(errno));

    struct dirent *dir;

    while ((dir = readdir(dir_fd)) != NULL) {
        write(CONSOLE_FILE_DESCRIPTOR, dir->d_name, strlen(dir->d_name));
        write(CONSOLE_FILE_DESCRIPTOR, "\n", 1);
    }

    closedir(dir_fd);
    return 0;
}

int recover_file_from_trash(char *file_name) {
    char *trash_file_path = get_trashcan_relative_file_path(file_name);

    int result = copy(trash_file_path, file_name);
    if (result != -1) {
        result = unlink(trash_file_path);
    }

    free(trash_file_path);
    return (result == -1) ? print_err_and_return_err_code(strerror(errno)) : 0;
}

int delete_file_from_trash(char *file_name) {
    char *trash_file_path = get_trashcan_relative_file_path(file_name);
    int result = unlink(trash_file_path);
    free(trash_file_path);

    return (result == -1) ? print_err_and_return_err_code(strerror(errno)) : 0;
}
```