dirent și readdir

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

void compari_fis(DIR* prim, DIR* secund, char* dirname, char* compname, bool compar) {
    struct dirent *d1, *d2;
    d1 = readdir(prim);
    //posibil ca directoryul sa fie gol
    while (d1) {
        d2 = readdir(secund);
        while(d2 && strcmp(d1->d_name, d2->d_name))
            d2 = readdir(secund);
        if (!d2)
            printf("Only in %s: %s\n", dirname, d1->d_name);
        else if (compar) {
            //daca au marimi diferite automat nu coincid
            int f1, f2;
            f1 = openat(dirfd(prim), d1->d_name, O_RDWR);
            f2 = openat(dirfd(secund), d2->d_name, O_RDWR);
            struct stat s1, s2;
            if (fstat(f1, &s1) != -1 && fstat(f2, &s2) != -1) {
                if (s1.st_size != s2.st_size)
                    printf("Files %s differ in size.\n", d2->d_name);
                else {
                    //verificam diferente
                    char c1, c2;
                    while (read(f1, &c1, sizeof(char)) < 1 &&
                     read(f2, &c2, sizeof(char)) < 1 &&
                     c1 != c2) {}
                    if (c1 != c2)
                        printf("Files %s differ in content.\n", d2->d_name);
                }
            }
        }
        rewinddir(secund);
        d1 = readdir(prim);
    }
    rewinddir(prim);
    rewinddir(secund);
}
int main(int argc, char** argv) {
    if (argc < 3) {
        printf("Trebuie precizate doua foldere.\n");
        return -1;
    }
    DIR* prim = opendir(argv[1]);
    DIR* secund = opendir(argv[2]);
    if (!prim || !secund) {
        printf("Unul dintre foldere nu exista.\n");
        return -1;
    }
    compari_fis(prim, secund, argv[1], argv[2], true);
    compari_fis(secund, prim, argv[2], argv[1], false);
    return 0;
}
```

Fisiere, Directoare

```
struct stat {
  dev_t st_dev;         /* ID of device containing file */
  ino_t st_ino;         /* Inode number */
  mode_t st_mode;       /* File type and mode */
  nlink_t st_nlink;     /* Number of hard links */
  uid_t st_uid;         /* User ID of owner */
  gid_t st_gid;         /* Group ID of owner */
  dev_t st_rdev;        /* Device ID (if special file) */
  off_t st_size;        /* Total size, in bytes */
  blksize_t st_blksize; /* Block size for filesystem I/O */
  blkcnt_t st_blocks;   /* Number of 512B blocks allocated */
  /* Since Linux 2.6, the kernel supports nanosecond
     precision for the following timestamp fields.
     For the details before Linux 2.6, see NOTES. */

  struct timespec st_atim;      /* Time of last access */
  struct timespec st_mtim;      /* Time of last modification */
  struct timespec st_ctim;      /* Time of last status change */
#define st_atime st_atim.tv_sec /* Backward compatibility */
#define st_mtime st_mtim.tv_sec
#define st_ctime st_ctim.tv_sec
};

struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file; not supported
                                   by all filesystem types */
    char           d_name[256]; /* Null-terminated filename */
};

struct timespec {
    time_t  tv_sec;  /* Seconds */
    long    tv_nsec; /* Nanoseconds */
};

#include <ctype.h>
//contine uint64_t si altele
```

EOF Check descriptor

```
int sfarsit(int d) {
    off_t old_position = lseek(d, 0, SEEK_CUR);
    struct stat currentstat;
    if (fstat(d, &currentstat) == -1)
        return -1;
    return currentstat.st_size == old_position;
}
```

`FTW` pentru marimea unui directory

```
int ftw(const char *dirpath,
               int (*fn) (const char *fpath, const struct stat *sb,
                          int typeflag),
               int nopenfd);
```

Citirea variabilelor de env

```c
#include <stdlib.h>
char *getenv(const char *name);

int main(int argc, char **argv, char **envp) {
    for (char **env = envp; *env != 0; env++)
    {
        char *thisEnv = *env;
        printf("%s\n", thisEnv);
    }
    return 0;
}
```

Sortare prin interclaseare (mergesort)

```c
void merge(int* array, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[n1], R[n2];
    for (int i = 0; i < n1; i++)
        L[i] = array[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = array[m + 1 + j];
    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            array[k++] = L[i++];
        else
            array[k++] = R[j++];
    }
    while (i < n1)
        array[k++] = L[i++];
    while (j < n2)
        array[k++] = R[j++];
}

void mergeSort(int* array, int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(array, l, m);
        mergeSort(array, m + 1, r);
        merge(array, l, m, r);
    }
}
int main() {
    int arr[];//numere
    mergeSort(arr, numbers/2, numbers-1);
}
```

`mv` emulat prin `link symlink unlink`

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <errno.h>

int check_disk(const char *file1, const char *file2) {
    struct stat file1_stat;
    struct stat file2_stat;
    if (stat(file1, &file1_stat) == -1) {
        perror("stat");
        return 2;
    }
    if (stat(file2, &file2_stat) == -1) {
        return 0;
    }
    return file1_stat.st_dev == file2_stat.st_dev;
}
int main(int argc, char *argv[]) {
    int recursive = 0;
    if (argc < 3) {
        printf("Usage: %s [-r] <file1> <file2>\n", argv[0]);
        return 2;
    }
    if (argc == 4 && strcmp(argv[1], "-r") == 0) {
        recursive = 1;
    }
    char *file1 = argv[argc - 2];
    char *file2 = argv[argc - 1];
    if (access(file2, F_OK) == 0) {
        printf("%s already exists\n", file2);
        return 2;
    }
    int disk = check_disk(file1, file2);
    if (disk == 1) {
        if (link(file1, file2) == -1) {
            perror("link");
            return 2;
        }
        if (unlink(file1) == -1) {
            perror("unlink");
            return 2;
        }
    } else {
        if (symlink(file1, file2) == -1) {
            perror("symlink");
            return 2;
        }
    }
    return 0;
}
```

```c
int main(int argc, char** argv) {
    if (argc < 2) {
        fprintf(stderr, "Utilizare: %s com arg...\n", argv[0]);
        return 2;
    }
    //ori fd_pair, ori long, dar in cazul asta merge si int
    int tub_tata[2], cuvinte = 0;
    char buf;
    bool last_space = false, am_citit = false;
    //in acest punct nu stiu daca sunt in child process sau parent
    if (pipe(tub_tata) == -1) {
        perror("pipe");
        return 2;
    }
    //pipe poate da eroare?
    pid_t copid = fork();
    if (copid == -1) {
        //lipsa de permisiuni?
        perror("fork");
        return 2;
    }
    if(copid == 0) {
        //suntem copil

        //duplex pt stdin stdout
        //while ((dup2(tub_tata[1], 1) == -1)) {};
    dup2(tub_tata[1], 1);
        //indici: stdout e 1, stdin e 0 cred
        close(tub_tata[1]);
        close(tub_tata[0]);

        //din ceva motiv primul argv e numele programului ??
        execvp(argv[1], argv+1);
        perror(argv[1]);
        return 2;
    }

    //opusul cand esti parinte
    close(tub_tata[1]);
    while(1) {
        size_t num = read(tub_tata[0], &buf, sizeof(char));
        if (num < sizeof(char)) //EOF sau eroare
            break;
    if(isspace(buf)) {
        if (!am_citit)
            cuvinte--;
        last_space = true;
    }
    else if (last_space) {
            cuvinte++;
        last_space = false;
    }
    am_citit = true;
    }
    wait(0);
    //printf("%d", am_citit);
    printf("Procesul copil a afisat %d cuvinte.\n", !am_citit ? 0 : ++cuvinte);
    return 0;
}
```

Comunicare intre 2 procese folosind `SIGUSR1` pentru 1 si `SIGUSR2` pentru 0

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

pid_t pid;
char buffer;
int current_letter = 0;

void send_bit(int bit) {
    if (bit == 1) {
        kill(pid, SIGUSR1);
    } else {
        kill(pid, SIGUSR2);
    }
}

void receive_letter(int signum) {
    if (signum == SIGUSR1) {
        buffer |= 1 << current_letter;
    }
    current_letter++;
    if (current_letter == 8) {
        putchar(buffer);
        buffer = 0;
        current_letter = 0;
    }
    fflush(stdout);
}

int main(int argc, char* argv[]) {
    printf("Enter pid of other process: ");
    scanf("%d", &pid);
    if (pid == 0) {
        printf("This process will only receive information\n");
        signal(SIGUSR1, receive_letter);
        signal(SIGUSR2, receive_letter);
        while(1) {
            sleep(1);
        }
    } else {
        printf("Enter text to send to other process:\n");
        signal(SIGUSR1, receive_letter);
        signal(SIGUSR2, receive_letter);
        while (scanf("%c", &buffer)) {
            if (buffer == 'P')
                break;
            for (int i = 0; i < 8; i++) {
                send_bit((buffer >> i) & 1);
                sleep(1);
            }
        }
    }
    return 0;
}
```

SIGUSR1

```c
#include <signal.h>

void my_handler(int signum)
{
    if (signum == SIGUSR1)
    {
        printf("Received SIGUSR1!\n");
    }
}

signal(SIGUSR1, my_handler);
```

SIGCHLD

```c
#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/resource.h>
//interceptarea codului returnat de procesul copil
void proc_exit()
{
    int wstat;
    union wait wstat;
    pid_t   pid;

    while (1) {
        pid = wait3 (&wstat, WNOHANG, (struct rusage *)NULL );
        if (pid == 0)
            return;
        else if (pid == -1)
            return;
        else printf ("Return code: %d\n", wstat.w_retcode);
    }
}
int main() {
    signal (SIGCHLD, proc_exit);
    switch (fork()) {
        case -1:
            perror("fork");
            return 0;
        case 0:
            printf("I'm alive (temporarily)\n");
            exit(rand());
        default:
            pause();
    }
}
```

Utilizatori si grupuri

```
#include <pwd.h>
#include <sys/types.h>
struct passwd {
    char   *pw_name;       /* username */
    char   *pw_passwd;     /* user password */
    uid_t   pw_uid;        /* user ID */
    gid_t   pw_gid;        /* group ID */
    char   *pw_gecos;      /* user information */
    char   *pw_dir;        /* home directory */
    char   *pw_shell;      /* shell program */
};
```

Emulare `getpwnam`

```
#include <stdio.h>
#include <pwd.h>
#include <string.h>
//prototype
struct passwd *getpwnam(const char *name);

struct passwd* iaparolanume(const char* nume) {
    static struct passwd* user;
    do {
        user = getpwent();
    } while (strlen(user->pw_name) && strcmp(user->pw_name, nume));
    return user;
}
```

Emulare `whoami`

```
#include <stdlib.h>
#include <pwd.h>
#include <stdio.h>
#include <unistd.h>
//geteuid() returns the effective user ID of the calling process.
int main(int argc, char *argv[]) {
    uid_t uid = geteuid();
    struct passwd *pw = getpwuid(uid);
    printf("%s\n", pw->pw_name);
    return 0;
}
```

Problema 1

Consideram executarea urmatorului cod C:

```
int k = 5;
if(fork() == fork()) ++k; else --k;
printf("%d\n", k);
```

- Cate procese apar in total (incluzand si procesul initial)? Descrieti dependentele acestor procese prin perechi de forma `p1(n1) -> p2(n2)`, insemnand: procesul p1 afisaza n1 si genereaza procesul p2 care afisaza n2. Justificati raspunsurile.
- Completati codul la un program intreg care, in plus, sa afiseze si PID-ul parintelui si PID-ul propriu (pentru a putea dovedi cele afirmate mai devreme). Este permisa doar inserarea de cod si schimbarea indentarii, nu si stergerea sau modificarea codului existent.

Rezolvare:

Patru procese apar in total: - p1(4) -> p2(4) - p1(4) -> p3(4) - p2(4) -> p4(6)
Observatie: Pe un emulator de tty, procesul tata este lansat drept proces copil al terminalului.

Completare:

```
#include <unistd.h>
#include <stdio.h>
int main() {
    int k = 5;
    if(fork() == fork()) ++k; else --k;
    printf("%d->%d(%d)\n", getppid(), getpid(), k);
    return 0;
}
```

Problema 4

Scrieti o functie in limbajul C:

```
int inv(char *numefis);
```

care interschimba prima si a doua jumatate din fisierul specificat de `numefis`. Daca fisierul are un numar impar de caractere, caracterul din mijloc este lasat pe loc. Functia returneaza numarul de caractere din fisier sau -1 in caz de eroare. Memoria folosita va fi limitata de o constanta (deci, nu se va incarca tot continutul fisierului in memorie) iar la iesirea din functie fisierul va fi inchis.

```
int inv(char* numefis) {
    int fd = open(path, O_RDWR);
    if (!fd)
        return -1;
    off_t file_size = lseek(fd, 0, SEEK_END);
    off_t middle = file_size / 2;
    char buff1, buff2;
    for (int i = 0; i < middle; i++) {
        lseek(fd, i, SEEK_SET);
        read(fd, &buff1, 1);

        lseek(fd, file_size%2 ? middle+i+1 : middle+i, SEEK_SET);
        read(fd, &buff2, 1);

        lseek(fd, -1, SEEK_CUR);
        write(fd, &buff1, 1);

        lseek(fd, i, SEEK_SET);
        write(fd, &buff2, 1);
    }
    close(fd);
    return 0;
}
```

Problema 5

```
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>
#include <pwd.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc < 2) {
        printf("Usage: %s <directory>\n", argv[0]);
        return 2;
    }

    DIR *dir = opendir(argv[1]);
    if (dir == NULL) {
        printf("Could not open directory\n");
        return 2;
    }

    struct dirent *entry;
    struct stat stat_buf;
    struct passwd *pwd;
    int total_size = 0;
    uid_t user_id = geteuid();
    char *user_name;
    int fd;
    char newline = '\n';

    while ((entry = readdir(dir)) != NULL) {
        char path[1024];
        strcpy(path, argv[1]);
        strcat(path, entry->d_name);
        if (stat(path, &stat_buf) == -1) {
            perror("stat");
            return 2;
        }

        if (!S_ISREG(stat_buf.st_mode)) {
            continue;
        }

        if (stat_buf.st_uid != user_id) {
            continue;
        }

        total_size += stat_buf.st_size;
        pwd = getpwuid(user_id);
        user_name = pwd->pw_name;
        fd = open(path, O_WRONLY | O_APPEND);
        write(fd, user_name, strlen(user_name));
        write(fd, &newline, 1);
        close(fd);
    }
    printf("%d", total_size);
    closedir(dir);
    return 0;
}
```

Problema 6

Scrieti un program ‘prog’ in limbajul C, care se poate lansa sub forma ‘./prog comanda1 @ comanda2’, unde ‘comanda1’ si ‘comanda2’ sunt comenzi externe shell (specificatorii unor fisiere executabile) care pot avea mai multe cuvinte si care lanseaza cele doua comenzi a.i. procesele sa formeze lantul: prog ——–> comanda1 ——–> comanda2

Procesul din stanga unei sageti este parintele celui din dreapta si comunica cu el printr-un tub, a.i parintele are stdout la tub iar copilul are stdin la tub. Procesele isi vor inchide descriptorii nefolositi pe tuburi, a.i. tuburile sa aiba un singur cititor si un singur scriitor. Dupa crearea copilului, procesul ‘prog’ va copia stdin la stdout, apoi va inchide stdout si va astepta terminarea copilului, apoi va afisa codul de retur al acestuia pe stderr si se va termina.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char *argv[]) {
    if (argc < 4) {
        printf("Usage: %s <com1> @ <com2>\n", argv[0]);
        return 2;
    }
    int tub_tata[2];
    if (pipe(tub_tata) == -1) {
        perror("pipe");
        return 2;
    }
    pid_t copid = fork();
    if (copid == -1) {
        perror("fork");
        return 2;
    }
    if (copid == 0) {
        //child 1
        close(tub_tata[0]);
        dup2(tub_tata[1], 1);
        execlp(argv[1], "");

        copid = fork();
        if (copid == -1) {
            perror("fork");
            return 2;
        }
        return 0;
    }
    copid = fork();
    if (copid == 0) {
        //child 2
        dup2(tub_tata[0], 0);
        execlp(argv[3], "");
        //close(tub_tata[1]);
        return 2;
    }
    return 0;
}
```

Problema 3

Un sistem de programare in timp real are patru evenimente periodice cu perioadele de 50, 100, 200, si respectiv 250msec. Presupunem ca cele patru evenimente au nevoie de 35, 20, 10, si respectiv x msec timp de procesor. De asemenea, presupunem ca supraincarcarea cauzata de comutarea intre procese este neglijabila. Care este valoarea maxima a lui x pentru care sistemul este planificabil ? Justificati raspunsul (aratati calculul).

Pentru a determina dacă sistemul este planificabil, trebuie să verificăm dacă timpul de procesare total al fiecărui eveniment este mai mic sau egal cu perioada sa respectivă.

Timpul de procesare total pentru evenimentul cu perioada de 50msec = 35msec
Timpul de procesare total pentru evenimentul cu perioada de 100msec = 20msec
Timpul de procesare total pentru evenimentul cu perioada de 200msec = 10msec

Pentru a determina valoarea maximă a lui x pentru care sistemul este planificabil, vom utiliza ecuația:

Timpul de procesare total pentru evenimentul cu perioada de 250msec = xmsec

Astfel, trebuie să avem: x <= 250msec

Pentru a justifica raspunsul, se poate vedea ca in cazul in care x este mai mare de 250msec, atunci timpul de procesare total pentru evenimentul cu perioada de 250msec va fi mai mare decat perioada acestuia, si in consecinta sistemul nu va fi planificabil. Astfel, valoarea maxima a lui x pentru care sistemul este planificabil este 250msec

Problema 2

Rutina de tratare a intreruperii de ceas de pe un anumit calculator are nevoie de 2msec (incluzand supraincarcarea pentru comutarea proceselor) per tact de ceas. Ceasul genereaza tacti la frecventa de 70 Hz. Ce procent din capacitatea procesorului este dedicat ceasului ? Justificati raspunsul (aratati calculul).

Pentru a calcula procentul de capacitate a procesorului dedicat ceasului, trebuie sa calculam cat timp din fiecare secunda este dedicat tratarii intreruperii de ceas.

Frecventa ceasului este de 70 Hz, ceea ce inseamna ca el genereaza 70 de tacti pe secunda. Timpul necesar pentru a trata fiecare tact este de 2 msec, deci pentru a trata toate tactile din secunda, se necesita 70 * 2 = 140 msec.

Acest timp poate fi convertit in procente din o secunda, care este de 100%. 140msec / 1000msec = 0.14 sau 14%.

Deci, 14% din capacitatea procesorului este dedicat tratarii intreruperii de ceas.

11. Scrieţi un program care primeşte ca argumente în linia de comandă un fişier şi un director şi determină dacă fişierul se află ca nume şi conţinut în arborescenţa cu originea în director. În caz afirmativ, se va determina numărul de apariţii.

```
#include <stdio.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ftw.h>
#include <string.h>
#include <unistd.h>
char* nume;
int fd;
int aparitii=0;
int ftw_cb(const char *fpath, const struct stat *sb, int typeflag) {
    int dif = 0;
    while(strchr(fpath+dif, '/')) {
        dif = strchr(fpath+dif, '/') - fpath + 1;
    }
    if(strcmp(fpath+dif, nume))
        return 0; //different names
    int f2;
    if ((f2=open(fpath, O_RDONLY)) == -1) {
        perror("open");
        return 0;
    }
    char c1, c2;
    int ok = 1;
    while(read(f2, &c2, 1) && read(fd, &c1, 1) && ok)
        ok = c1 == c2;
    aparitii += ok;
    lseek(fd, 0, SEEK_SET);
    close(f2);
    return 0;
}
int main(int argc, char** argv) {
    if (argc < 3) {
        printf("Usage: %s <fisier> <directory>\n", argv[0]);
        return 2;
    }
    fd = open(argv[1], O_RDONLY);
    nume = argv[1];
    struct stat fis_stat;
    if(stat(argv[1], &fis_stat)==-1) {
        perror("stat");
        return 2;
    }
    if (!S_ISREG(fis_stat.st_mode)) {
        printf("%s nu este un fisier regular.\n", argv[1]);
        return 2;
    }
    if(ftw(argv[2], &ftw_cb, 1)==-1) {
        perror("ftw");
        return 2;
    }
    printf("Aparitii: %d\n", aparitii);
}
```