Operációs rendszerek BSc

10.gyak.

2021. 04. 13.

Készítette: Nagy Róbert

Programtervező Informatikus
Neptunkód: JMDRGG

Miskolc, 2021

```c
int main(void) {
    int nbytes;
    pid_t childpid;
    char string[] = "XY jmdrgg\n";
    char readbuffer[80];
    int fd[2];
    pipe(fd);

    if((childpid = fork()) == -1) {
        perror("fork");
        exit(1);
    }

    if(childpid == 0) {
        close(fd[0]);
        write(fd[1], string, (strlen(string)+1));
        exit(0);
    } else {
        close(fd[1]);
        nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
        printf("kapott string: %s", readbuffer);
    }
```

A pipehoz használunk egy két elemű tömböt, amelyet fd (file descriptor)-al jelölunk. Az fd[1] írást, az fd[0] olvasást jelent. Ebben a példában a gyerek processz fog irni, ezért bezárja a pipe olvasó végét, mert nincs rá szükség. A szülő processz ennek az ellenkezőjét teszi.

```c
    int nbytes;
    pid_t childpid;
    char string[] = "Nagy Robert\n";
    char readbuffer[80];

    //nevesitett csovezetek FN neven chmod 666 jogosultsaggal
    mkfifo(FN, S_IFIFO | 0666);

    if ((childpid = fork()) == -1) {
            perror("fork hiba");
            exit(1);
    }

    if (childpid == 0) {
        int fd = open(FN, O_WRONLY);
        write(fd, string, sizeof(string));
        close(fd);
        unlink(FN);
    } else {
        int fd2 = open(FN, O_RDONLY);
        nbytes = read(fd2, &readbuffer, sizeof(readbuffer));
        close(fd2);
        unlink(FN);
```

A nevesített pipe alapvetően egy file, ami pipe funkcionalitással rendelkezik. Linux alatt mkfifo parancsal lehet nevesített pipokat létrehozni. A nevesített pipe lényege, hogy több processz is tudjon rajta keresztül kommunikálni és hogy ne kelljen a file descriptort ismernie minden processznek. Létrehozása c nyelven az mkfifo hívással történik (az mkfifo hívás nagyon hasonló a mknod híváshoz, mert alapvetően azt hívja meg csak kevesebb argumentummal.). A mkfifo hívásnak meg kell adni a pipe nevét és a hozzáférési jogokat. (linuxbol a chmod megfelelője). A nevesített pipe olvasásakor blokkolódik, azaz vár, amíg valamelyik processz rá nem csatlakozik a pipera, majd bele nem ír. Ezt ki lehet kapcsolni az O_NONBLOCK flag használatával. Egy nevesített pipeot úgy szüntetünk meg, hogy minden processzt unlinkelünk róla.

```c
void main(int argc, char const *argv[]) {
    if (argc < 1) {
        perror("nincs argumentum");
        exit(-1);
    }
    pid_t pid = atoi(argv[1]);
    if (kill(pid, SIGALRM) == -1) {
        perror("sikertelen signalozas");
    }
}
```

```c
void main() {
    printf("A program pidje : %d\n",getpid());
    signal(SIGALRM, handleSigalarm);
    pause();
    printf("kibillent");
    exit(0);
}

void handleSigalarm() {
    printf("\njmdrgg");

}
```

Egy program képes szignált küldeni egy másik program számára a kill parancs segítségével. Ismernie kell hozzá a program pid-jét. A másik program eközben hívja a signal függvényt, amellyel lekezeli a sigalarm signált.

Futás eredmények:

```
→  JMDRGG_0413 git:(main) ✗ ./gyak3_masik
A program pidje : 14285
jmdrgg
kibillent
→  JMDRGG_0413 git:(main) ✗
```

```
  JMDRGG_0413 git:(main) ✗ ./gyak3 14285
  JMDRGG_0413 git:(main) ✗
```

```
→  JMDRGG_0413 git:(main) ✗ gcc jmdrgg_unnamed.c -o unnamed
→  JMDRGG_0413 git:(main) ✗ ./unnamed
kapott string: XY jmdrgg
→  JMDRGG_0413 git:(main) ✗
```

```
→  JMDRGG_0413 git:(main) ✗ gcc jmdrgg_named.c -o named
→  JMDRGG_0413 git:(main) ✗ ./named
Nagy Robert
→  JMDRGG_0413 git:(main) ✗
```