Universiteti i Prishtinës “Hasan Prishtina”

Fakulteti Inxhinierisë Elektrike dhe Kompjuterike

Dokumentim teknik

Lënda: Sistemet Operative

Titulli i projektit: Komunikimi në mes dy proceseve

Emri profesorit/Asistentit	**Emri & mbiemri studentëve / email adresa**	
Prof. Dr. Artan MAZREKAJ Msc. Dalinë VRANOVCI	1. Arlinda Kastrati	arlinda.kastrati4@student.uni-pr.edu
	2. Alberiana Tofaj	alberiana.tofaj@student.uni-pr.edu
	3. Fortesa Mujaj	fortesa.mujaj@student.uni-pr.edu
	4. Veranda Blakaj	veranda.blakaj@student.uni-pr.edu

Prishtinë, 2022

Përmbajtja

Abstrakt

Ky raport paraqet një përmbledhje të detajeve në lidhje me përshkrimin, dizajnimin, implementimi dhe testimi i një programi i cili mundëson komunikimin mes dy proceseve prind/fëmije. Ky projekt është implementuar duke përdorur sistemin operativ të Ubuntu dhe gjuhën programuese C.

Objektivi kryesor i këtij raporti është të paraqesë parimet që qëndrojnë prapa programimit të sistemeve operative gjegjësisht proceseve dhe sinkronizimit të tyre në sistemin operativ që i takon familjes së Linuxit.

Hyrje

Sistemet operative janë programe softuerike të cilat përdoren për të menaxhuar pajisjet kompjuterike siç janë: telefonat inteligjentë, tabletët, kompjuterët, superkompjuterët, serverët në internet, makinat, orët inteligjente, etj. Sistemet operativ janë ato të cilat eliminojnë nevojën për të njohur gjuhën e kodimit për të ndërveprojnë me pajisjet kompjuterike. Sistemet operative janë një shtresë e ndërfaqes grafike të përdoruesit (GUI), e cila vepron si një platformë midis përdoruesit dhe harduerit të kompjuterit. Për më tepër, sistemi operativ menaxhon anën e softuerit të një kompjuteri dhe kontrollon ekzekutimin e programeve.

Pa një sistem operativ, çdo aplikacion do të duhet të përfshijë UI-në e vet, si dhe kodin gjithëpërfshirës të nevojshëm për të trajtuar të gjitha funksionet e nivelit të ulët të kompjuterit themelor, të tilla si ruajtja e diskut, ndërfaqet e rrjetit etj. Duke marrë parasysh gamën e madhe të pajisjeve themelore të disponueshme, kjo do të rriste shumë madhësinë e çdo aplikacioni dhe do ta bënte zhvillimin e softuerit jopraktik.

Sistemi operativ, sot ofron një platformë gjithëpërfshirëse që identifikon, konfiguron dhe menaxhon një seri harduerësh, duke përfshirë procesorët, pajisjet e memories dhe menaxhimin e tyre, chipsets, ruajtje, rrjetëzimi, komunikimi mes porteve si Video Graphics Array (VGA), High-Definition Multimedia Interface (HDMI), Universal Serial Bus (USB) dhe ndërfaqet e nënsistemeve të tilla si Peripheral Component Interconnect Express (PCIe).

Pesë sistemet operative më të zakonshme janë: Microsoft Windows, Apple macOS, Linux, Android dhe iOS i Apple.

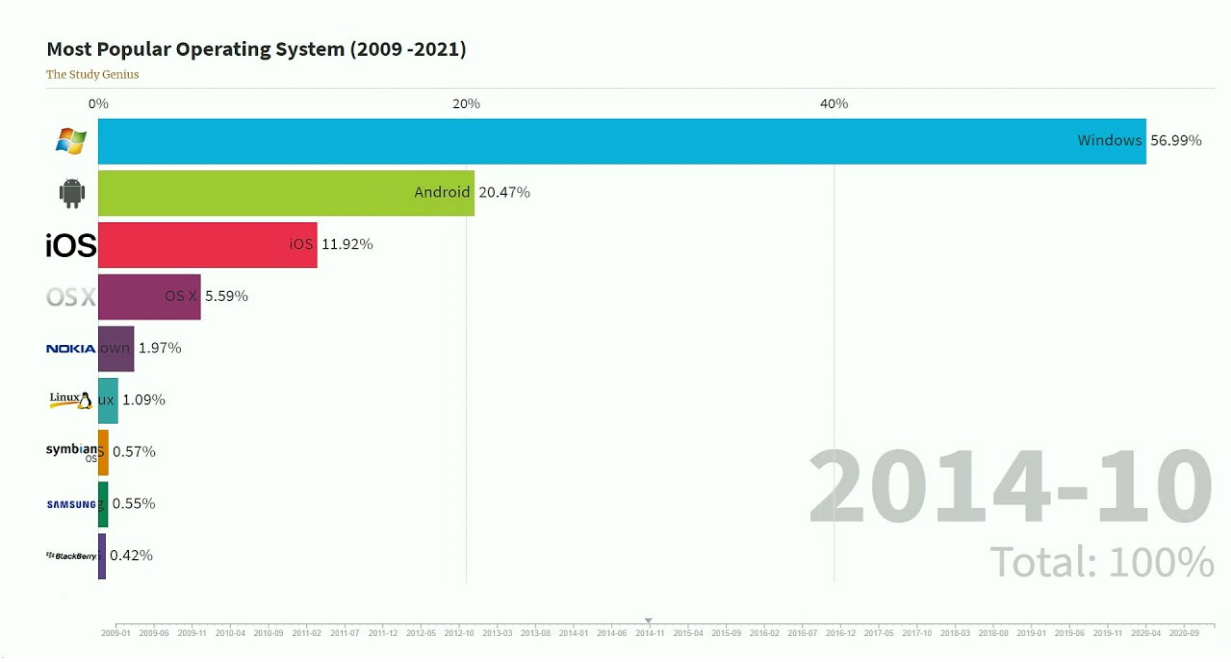

Figura 1 Sistemet operative më të përdoruarat(2009-2021)

1. Qëllimi i punimit

Komunikimi ndërprocesorik (IPC) është një mekanizëm i cili lejon proceset të komunikojnë me njëri-tjetrin dhe të sinkronizojnë veprimet e tyre. Poashtu, mundëson shkëmbimin e burimeve dhe të dhënave ndërmjet proceseve pa ndërhyrje.

Proceset që ekzekutohen njëkohësisht në sistemin operativ mund të jenë:

- procese të pavarura
- procese bashkëpunuese

Një proces është i pavarur dhe mund ose nuk mund të ndikohet nga procese të tjera që ekzekutohen në sistem. Çdo proces që nuk ndan të dhëna me asnjë proces tjetër është i pavarur.

Supozoni nëse një proces po bashkëpunon atëherë, ai mund të ndikohet nga procese të tjera që po ekzekutohen në sistem. Çdo proces që ndan të dhënat me një proces tjetër quhet proces bashkëpunues.

Për realizimin e komunikimit ne mes te dy proceseve (prind/fëmijë) në atë mënyrë qe procesi prind shkruan në një fajll dhe procesi fëmijë lexon përmbajtjen e fajllit mund te përdorim pipe dhe fork.

1.1 Fork

Thirrja sistemore fork() përdoret për te krijuar një kopje te procesit aktual. Procesi i ri i krijuar është identik me procesin nga i cili është bere kopje, me përjashtim qe ka proces id-ne e ndryshme. Procesi nga I cili thërrasim fork() quhet procesi prind ndërsa procesi i krijuar quhet proces fëmije. Për te pare se si funksionon thirrja sistemore fork(), do te krijojmë një program te thjeshte ne C.

1.2 Pipe

Thirrja sistemore pipe() është një teknikë e përdorur për komunikimin ndër-procesor. Një pipe është një mekanizëm me anë të të cilit dalja e një procesi drejtohet në hyrjen e një procesi tjetër. Kështu ai siguron një rrjedhë të njëanshme të të dhënave midis dy proceseve të lidhura. Pipe komunikojnë me metodën FIFO. Nëse pipe është plot, procesi bllokohet derisa të ndryshojë gjendja e pipe. Në mënyrë të ngjashme, një proces leximi bllokohet, nëse tenton të lexojë më shumë bajt që janë aktualisht në pipe, përndryshe procesi i leximit ekzekutohet. Vetëm një proces mund të hyjë në një pipe në të njëjtën kohë.

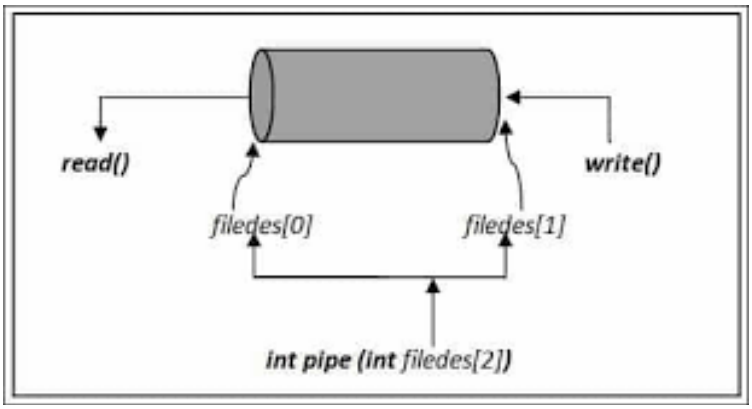

Figura 2 Pipe()

2. Pjesa kryesore

Meqë sistemi operativ **Ubuntu** tashmë është i instaluar në **Virtual Machine** ,përmes terminalit krijojmë një fajll të ri në direktoriumin ku do të ruhet fajlli të cilit i jemi qasur me komandë **cd**.

```
alberiana@alberiana-VirtualBox:~$ cd Desktop
alberiana@alberiana-VirtualBox:~/Desktop$ cd Project-OS
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ gedit project.c
```

Figura 3 Procesi i krijimit të fajllit

2.1 Kodi

Me poshtë kemi paraqitur te gjithë kodin te shënuar ne gjuhen programuese C.

```c
//Has the information for all input, output related functions.
#include <stdio.h>
//Provide exit function
#include <stdlib.h>
/*Provide one variable type, one macro, and various functions  for manipulating arrays of character*/
#include <string.h>
//Provide system calls: read, write and pipe
#include <unistd.h>
//Provide the wait system call
#include <sys/types.h>
//Provide open and creat system calls for files
#include <fcntl.h>
int main(){
        /*The pid with which the process will be identified*/
        pid_t pid;
        /*Declaring pointer of FILE type*/
        FILE *fd;
        /*Two file descriptors used on pipe to read from and write to*/
        int fs[2];
        /*Where data from file will reside temporarily*/
        char buffer[30];
        /*Condition whether the pipe can be created or not*/
        if(pipe(fs) == -1){
                /*The message that will be displayed if the pipe cannot be created*/
                perror("Pipe failed!\n");
                exit(1);
        }
        /*If the fork id is less than 0 the fork can not be created*/
        if((pid = fork()) < 0) {
                perror("Fork failed!\n");
                exit(1);
        }
        /*If the process id is equal to 0, the child process is created*/
        if(pid == 0){
                /*Child process will close write end of the pipe since
                it is only going to use read end of it*/
                close(fs[1]);

                /*Check if the file is provided by the parent process*/
                if(fd == NULL) {
                    printf("Could not read file data.txt");
                return 1;
                }
                /*Child process reads from the pipe's read end in 12 character chunks
                 by     executing a read system call */
                if(read(fs[0], buffer, 12) <= 0 ) {
                        perror("Child read failed!\n");
```

Figura 4 Pjesa e parë e kodit

```
39                /*Check if the file is provided by the parent process*/
40                if(fd == NULL) {
41                    printf("Could not read file data.txt");
42                return 1;
43                }
44                /*Child process reads from the pipe's read end in 12 character chunks
45                 by     executing a read system call */
46                if(read(fs[0], buffer, 12) <= 0 ) {
47                        perror("Child read failed!\n");
48                        exit(1);
49                }
50                /*Child process read data from parent process*/
51                printf("\nChild is reading... \n %s\n", buffer);
52                /*Once the reading is finished,
53                the process will close read end of the pipe and the output file*/
54                close(fs[0]);
55                exit(0);
56        }
57        else{
58                /*Close the read end of the pipe since it will write on the pipe*/
59                close(fs[0]);
60                /*Open file and write in it*/
61                fd = fopen("data.txt", "w");
62                /*If no file is set in the fd variable, the message is displayed*/
63                if(fd == NULL) {
64                    printf("Could not oper file data.txt");
65                    return 1;
66                }
67                printf("Parent is writing...\n");
68                printf("Write something: \n");
69                /*It reads a line from the specified stream and stores it into
70                the string pointed to by str.*/
71                /*The data is taken from stdin up to a maximum of 30 characters and placed in the buffer*/
72                fgets(buffer, 30, stdin);
73                /*The data found in the buffer is placed in the file*/
74                fputs(buffer, fd);
75                /*Parent process writes on pipe what it has read from the input file
76                and if the data in the buffer is less than 120 bytes the message is displayed*/
77                if(write(fs[1], buffer, 30) <= 0) {
78                        perror("Parent write failed!\n");
79                        exit(1);
80                }
81                /*Once the parent is done with copying data from input file to the pipe
82                it will close the write end of the pipe and the input file*/
83                close(fs[1]);
84        }
85
```

Figura 5 Pjesa e dytë e kodit

Me poshtë janë paraqitur libraritë e përdoruara për të mundësuar komunikimin prind fëmijë.

```c
//Has the information for all input, output related functions.
#include <stdio.h>
//Provide exit function
#include <stdlib.h>
/*Provide one variable type, one macro, and various functions  for manipulating arrays of character*/
#include <string.h>
//Provide system calls: read, write and pipe
#include <unistd.h>
//Provide the wait system call
#include <sys/types.h>
//Provide open and creat system calls for files
#include <fcntl.h>
```

Figura 6 Libraritë

Deklarimi i variablave që do të përdoren më vonë ne program.

```c
/*The pid with which the process will be identified*/
pid_t pid;
/*Declaring pointer of FILE type*/
FILE *fd;
/*Two file descriptors used on pipe to read from and write to*/
int fs[2];
/*Where data from file will reside temporarily*/
char buffer[30];
```

Figura 7 Variablat e deklaruara

Krijimi i kushteve qe nëse pipe dhe procesi nuk mund te krijohen të shfaqet mesazhi si lajmërim.

```c
/*Condition whether the pipe can be created or not*/
if(pipe(fs) == -1){
        /*The message that will be displayed if the pipe cannot be created*/
        perror("Pipe failed!\n");
        exit(1);
}
/*If the fork id is less than 0 the fork can not be created*/
if((pid = fork()) < 0) {
        perror("Fork failed!\n");
        exit(1);
}
/*If the process id is equal to 0, the child process is created*/
```

Figura 8 Pipe() dhe fork()

Në këtë pjese të kodit është paraqitur procesi fëmijë do të thotë nëse plotësohet kushti për **pid==0** atëherë programi vazhdon me ekzekutimin e kodit për procesin fëmijë. Së pari pipe mbyllet sepse procesi fëmijë vetëm do të lexoj fajlln e dërguar nga procesi prind pastaj kushti që nëse procesi prind nuk ka vendosur fajllin në variabilen e deklaruar shfaqet mesazhi përkatës.

Në kushtin tjetër **if** shikojmë nëse procesi fëmijë mund te lexoj ose jo në fajllin e deklaruar, nëse mundet e shfaq mesazhin që ka lexuar nëse jo atëherë shfaqet mesazhi përkatës. Në fund mbyllet pipe për lexim.

```
if(pid == 0){
        /*Child process will close write end of the pipe since
        it is only going to use read end of it*/
        close(fs[1]);

        /*Check if the file is provided by the parent process*/
        if(fd == NULL) {
            printf("Could not read file data.txt");
        return 1;
        }
        /*Child process reads from the pipe's read end in 12 character chunks
         by     executing a read system call */
        if(read(fs[0], buffer, 12) <= 0 ) {
                perror("Child read failed!\n");
                exit(1);
        }
        /*Child process read data from parent process*/
        printf("\nChild is reading... \n %s\n", buffer);
        /*Once the reading is finished,
        the process will close read end of the pipe and the output file*/
        close(fs[0]);
        exit(0);
}
```

Figura 9 Procesi fëmijë

Në vazhdim është paraqitur pjesa e kodit për procesin prind. Fillimisht pipe për lexim mbyllet pastaj hapet fajlli për shkrim në të.

Kushti nëse fajlli nuk mund të hapet paraqet mesazhin përkatës ndërsa nëse fajlli mund të hapet procesi fillon leximin e te dhënave që shkruhen nga procesi prind në fajll.

Nëse nuk shkruhen te dhëna për fajllin paraqitet mesazhi. Dhe në fund mbyllet pipe për shkrim në të.

```
else{
        /*Close the read end of the pipe since it will write on the pipe*/
        close(fs[0]);
        /*Open file and write in it*/
        fd = fopen("data.txt", "w");
        /*If no file is set in the fd variable, the message is displayed*/
        if(fd == NULL) {
            printf("Could not oper file data.txt");
            return 1;
        }
        printf("Parent is writing...\n");
        printf("Write something: \n");
        /*It reads a line from the specified stream and stores it into
        the string pointed to by str.*/
        /*The data is taken from stdin up to a maximum of 30 characters and placed in the buffer*/
        fgets(buffer, 30, stdin);
        /*The data found in the buffer is placed in the file*/
        fputs(buffer, fd);
        /*Parent process writes on pipe what it has read from the input file
        and if the data in the buffer is less than 120 bytes the message is displayed*/
        if(write(fs[1], buffer, 30) <= 0) {
                perror("Parent write failed!\n");
                exit(1);
        }
        /*Once the parent is done with copying data from input file to the pipe
        it will close the write end of the pipe and the input file*/
        close(fs[1]);
}
```

Figura 10 Procesi prind

2.2 Testimi

Në vazhdim kemi paraqitur të gjithë procesin e testimit të komunikimit **prind-fëmijë** përmes **pipes()**.

Në figurën e mëposhtme janë paraqitur komandat për krijimin e fajllit në gjuhën programuese dhe text fajllin në të cilin do të shënojmë të dhënat.

```
alberiana@alberiana-VirtualBox:~$ cd Desktop
alberiana@alberiana-VirtualBox:~/Desktop$ cd Project-OS
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ gedit project.c
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ gedit data.c
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$
```

Figura 11 Krijimi i fajllave

Të dhënat në fajll shkruhen nga procesi prind. Procesi pret që të dhënat të shënohen në fajllin përkatës të deklaruar.

```
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ gcc -o project project.c
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ ./project
Parent is writing...
Write something:
```

Figura 12 Procesi pret për të dhënat hyrëse

Te dhënat shkruhen nga procesi prind përmes terminalit ne fajllin e deklaruar dhe poashtu të dhënat lexohen nga procesi fëmijë duke u shfaqur ne terminal.

```
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ gcc -o project project.c
alberiana@alberiana-VirtualBox:~/Desktop/Project-OS$ ./project
Parent is writing...
Write something:
Project for OS

Child is reading...
 Project for OS
```

Figura 13 Shënimi i të dhënave në fajll dhe leximi i tyre nga procesi fëmijë

Të dhënat e shënuara në fajllin **data.txt** janë paraqitur me poshtë:

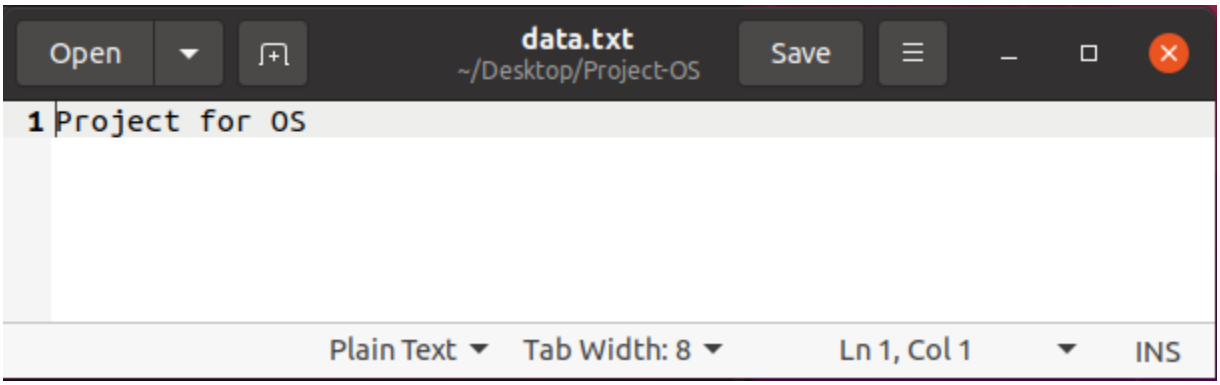

Figura 14 Fajlli data.txt

3. Konkluzione

Realizimi i gjithë projektit ishte mjaft sfidues gjatë gjithë kohës por njëkohësisht edhe argëtues. Ky projekt është projekti i parë që e kemi punuar i kësaj natyre që i takon sistemeve operative gjegjësisht proceseve dhe realizimin e komunikimit të tyre, dhe si student fillestar në këtë lëmi kemi hasur në disa pengesa gjatë realizimit të projektit. Por, me ndihmën e literaturës nga librat e huaj, koncepteve themelore të marra nga ligjëratat dhe ushtrimet e lëndës Sistemet Operative, bashkëpunim, ide të përbashkëta dhe me hulumtime të shumta ne besojmë se kemi arritur me sukses realizimin e komunikimit në mes dy proceseve. E gjithë kjo ka rezultuar edhe në përforcimin me të madh të koncepteve ndaj arkitekturës dhe komunikimit të proceseve në sistemet operative.

Programi i implementuar më lartë realizon komunikimin mes dy proceseve (prind/fëmijë) në atë mënyrë:

- Procesi prind shkruan në një fajll,
- Procesi fëmijë lexon përmbajtjen e fajllit,

dhe funksionon në mënyrën e duhur.

Referencat

[1] "Stack Exchange," 26 May 2021. [Online]. Available: https://unix.stackexchange.com/questions/446131/the-file-used-by-parent-and-child-process..

[2] "Geeksforgeeks," 14 May 2020. [Online]. Available: https://www.geeksforgeeks.org/ipc-technique-pipes/#:~:text=A%20Pipe%20is%20a%20technique,data%20between%20two%20related%20processes...

[3] "Tutorialspoint," [Online]. Available: https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_pipes.htm..

[4] H. Darvis, "Quick Adviser," 4 November 2021. [Online]. Available: https://quick-adviser.com/how-parent-processes-communicate-with-child-process/..