Operációs rendszerek BSc

5. Gyak.

2022. 03. 07.

Készítette:
Bodnár László Bsc
Mérnökinformatika
D1H8VP

Miskolc, 2022

1, A system() rendszerhívással hajtson végre létező és nem létező parancsot, és vizsgálja a visszatérési értékét, magyarázza 1-1

mondattal

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include < sys/wait.h>

int main(void){
    int status;

    if((status = system("date")) < 0)
        perror("system() error");
    if(WIFEXITED(status))
        printf("Normális befejeződés, visszaadott érték =
                            %d\n",WEXITSTATUS(status));


    exit(0);

}
```

Ha a parancs létezik végrehajtódik, ha nem akkor egy értékkel tér vissza.

2, Írjon programot, amely billenytyűzetről bekér Unix parancsokat és végrehajtja oket, majd kiirja a szabvanyos kimenetelre.

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include < sys/wait.h>

int main(void){
    char input[50];
    printf("Kérem adjon meg egy parancsot!");
    scanf("%s", input);
    system(input);

    exit(0);

}
```

3, Készítsen egy parent.c és egy child.c programot. A parent.c elindit egy gyermek processzt, ami különbözik a szülőitől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre.

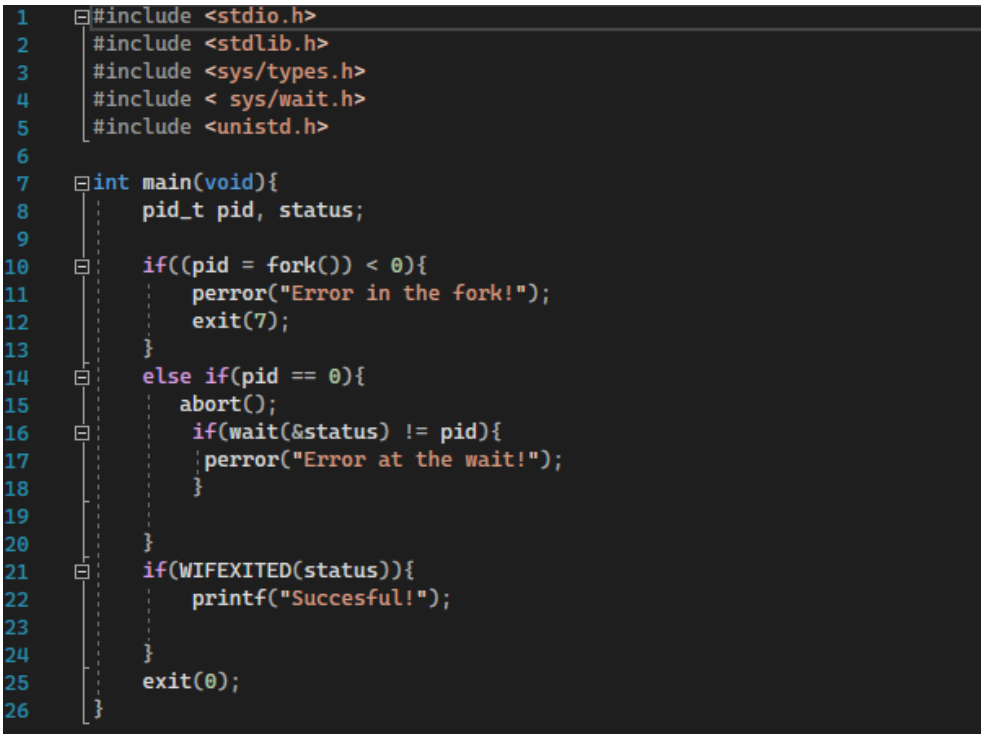
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include < sys/wait.h>
#include <unistd.h>

int main(void){

    for(int i = 0; i<10 ;i++){
        printf("Bodnar Laszlo D1H8VP\n");
        sleep(2);

        }



    exit(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include < sys/wait.h>
pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);


int main(){
    pid_t pid;

    if((pid = fork()) < 0 ){
        perror("fork error");
    }
    else if(pid == 0){
            if(execl("./child", "child",(char *)NULl) < 0)
                perror("execl error");

    }
    if(waitpid(pid, NULL , 0) < 0 )
    {   perror("wait error");

        }

    exit(0);
}
```

4,A fork() rendszerhívással hozzon létre egy gyerek proccesszt és abban hívjon meg egy exec családbeli rendszerhívást (pl execpl). A szülő megvárja a gyerek futását!

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include < sys/wait.h>
#include <unistd.h>

int main(void){
    pid_t pid;

    if((pid = fork()) < 0){
        perror("fork error");
    }
    else if(pid = 0){
            if(execl("ls", "-l","/home/bodnarlaszlo/OSgyak5",NULl) < 0)
                perror("execl error");
    }
    if(waitpid(pid, NULL, 0) < 0){
        perror("wait error");

    }

    exit(0);

}
```

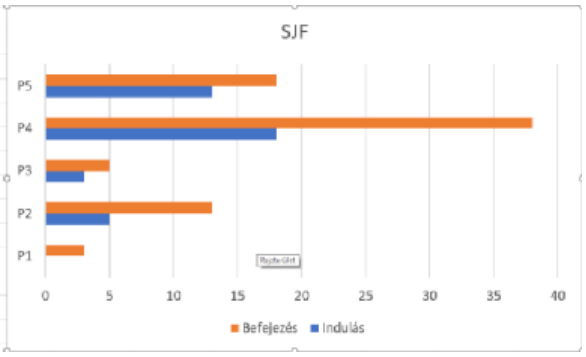
5,A fork() rendszerhívással hozzon létre gyerekeket, várja meg és vizsgálja a befejezési állapotokat



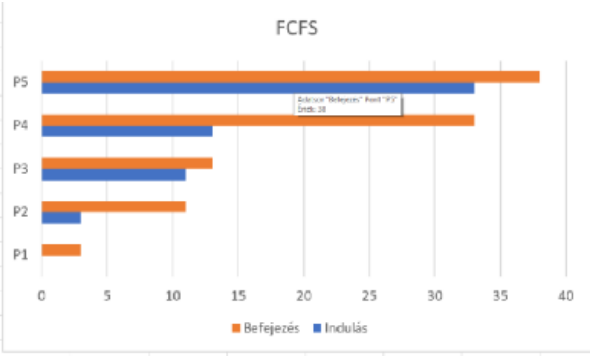
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include < sys/wait.h>
#include <unistd.h>

int main(void){
    pid_t pid, status;

    if((pid = fork()) < 0){
        perror("Error in the fork!");
        exit(7);
    }
    else if(pid == 0){
      abort();
        if(wait(&status) != pid){
        perror("Error at the wait!");
        }

    }
    if(WIFEXITED(status)){
        printf("Succesful!");

    }
    exit(0);
}
```

6,Adott a következő ütemezési feladat, amit a FCFS, SJF és Round Robin ütemezési algoritmus használásával készítsen el.

SJF	Érkezés	CPU idő	Indulás	Befejezés	Várakozás
P1	0	3	0	3	0
P2	1	8	5	13	4
P3	3	2	3	5	0
P4	9	20	18	38	9
P5	12	5	13	18	1



FCFS	Érkezés	CPU idő	Indulás	Befejezés	Várakozás
P1	0	3	0	3	0
P2	1	8	3	11	2
P3	3	2	11	13	8
P4	9	20	13	33	4
P5	12	5	33	38	21



RR: 5ms	Érkezés	CPU idő	Indulás	Befejezés	Várakozás	Várakozó processz
P1	0	3	0	3	0	P2
P2	1	8	3	8	2	P2,P3
P3	3	2	8	10	5	P2,P4
P4	9	20	13	18	4	P4,P5
P5	12	5	18	23	6	P4