Operációs rendszerek BSc

5. Gyak.

2022. 03. 07.

Készítette:

Martinák Mátyás Bsc

Programtervező informatikus

KLNSPG

Miskolc, 2022

1. A system() rendszerhívással hajtson végre létező és nem létező parancsot, és vizsgálja a visszatérési érteket.

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    int status = system("ls");;
    printf(status);
}
```

2. Írjon programot, amely billentyűzetről bekér Unix parancsokat és végrehajtja őket, majd kiírja a szabványos kimenetre. (pl.: amit bekér: date, pwd, who etc.; kilépés: CTRL-\)

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    char input[100];
    printf("Adjon meg egy parancsot:");
    scanf("%s", input);
    system(input);
    return 0;
}
```

3. Készítsen egy XY_parent.c és a XY_child.c programokat. A XY_parent.c elindít egy gyermek processzt, ami különbözik a szülőtől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre (10-ször) (pl. a hallgató neve és a neptunkód)!

parent.c:

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

int main (void) {
    pid_t pid;

    if ((pid = fork()) < 0){
        perror("Process error");
    }
    else if (pid == 0){
        if(execl("./child", "child", (char *) NULL) < 0){
            perror("Execl error");
        }
    }
    if (waitpid(pid, NULL, 0) < 0){
        perror("Wait error");
    }
    return 0;
}
```

child.c:

```c
#include <stdio.h>
#include <unistd.h>

int main(void) {
    for (int i = 0; i < 10; i++)
    {
        printf("Martinak Matyas KLNSPG\n");
        sleep(1);
    }
    return 0;
}
```

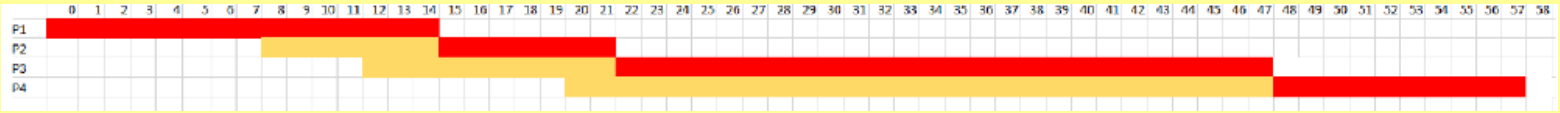
4. A fork() rendszerhívással hozzon létre egy gyerek processzt-t és abban hívjon meg egy exec családbeli rendszerhívást (pl. execlp). A szülő várja meg a gyerek futását!
5. A fork() rendszerhívással hozzon létre gyerekeket, várja meg és vizsgálja a befejeződési állapotokat (gyerekben: exit, abort, nullával való osztás)!

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

int main(void)
{
    int pid;
    int status;

    if ((pid = fork()) < 0) {
        perror("Hiba a forkban");
        exit(7);
    }
    else if (pid == 0)
       abort();
    if(wait(&status)!=pid) {
        perror("Hiba a wait-el");
    }
    if(WIFEXITED(status))
        printf("Sikeres");

    return 0;
}
```

6. **I.** Határozza meg FCFS és SJF esetén
   a.) A befejezési időt?
   b.) A várakozási/átlagos várakozási időt?
   c.) Ábrázolja Gantt diagram segítségével az aktív/várakozó processzek futásának menetét

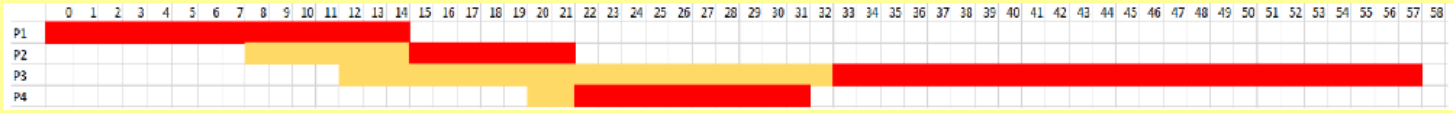
FCFS	*P1*	*P2*	*P3*	*P4*
Érkezés	*0*	*8*	*12*	*20*
CPU idő	*15*	*7*	*26*	*10*
Indulás	*0*	*15*	*22*	*48*
Befejezés	*15*	*22*	*48*	*58*
Várakozás	*0*	*7*	*10*	*28*

Átlagos várakozási idő: 45/4 – 11.25



SJF	*P1*	*P2*	*P3*	*P4*
Érkezés	*0*	*8*	*12*	*20*
CPU idő	*15*	*7*	*26*	*10*
Indulás	*0*	*15*	*32*	*22*
Befejezés	*15*	*22*	*58*	*32*
Várakozás	*0*	*7*	*20*	*2*

Átlagos várakozási idő: 29/4 – 7,25



II. Round Robin (RR) esetén

a.) Ütemezze az adott időszelet (5ms) alapján az egyes processzek (befejezési és várakozási/átlagos
várakozási idő) paramétereit (ms)!
b.) A rendszerben lévő processzek végrehajtásának sorrendjét?
c.) Ábrázolja Gantt diagram segítségével az aktív/várakozó processzek futásának menetét!”

RR: 5ms	***Érkezés***	***CPU igény***	***Indulás***	***Befejezés***	***Váró processz***	***Várakozás***	***Marad idő***
P1	*0*	*3*	*0*	*3*	*P2*	*0*	-
P2	*1*	*8*	*3*	*8*	*P2, P3*	*2*	*3*
P3	*3*	*2*	*8*	*10*	*P2, P4*	*5*	-
*P2**	*(8)*	*3*	*10*	*13*	*P4, P5*	*2*	-
P4	*9*	*20*	*13*	*18*	*P4, P5*	*4*	*15*
P5	*12*	*5*	*18*	*23*	*P4*	*6*	-
*P4**	*(18)*	*15*	*23*	*28*	*P4*	*5*	*10*
*P4**	*(28)*	*10*	*28*	*33*	*P4*	*0*	*5*
*P4**	*(33)*	*5*	*33*	*38*	-	*0*	-