Operációs rendszerek BSc

5.gyak.

2021. 03. 10.

Készítette:

Tóth József BProf
Üzemmérnök-informatikus alapszak
WI2GDP

Miskolc, 2021

1. **feladat -** *A system() rendszerhívással hajtson végre létező és nem létező parancsot, és vizsgálja a visszatérési érteket! Mentés: neptunkodgyak1.c*

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

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

2. **feladat -** *Írjon programot, amely billentyűzetről bekér Unix parancsokat és végrehajtja őket, majd kiírja a szabványos kimenetre. (pl.: amit bekér: date, pwd, who etc.; kilépés: CTRL-\)*
Mentés: neptunkodgyak2.c

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

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

3. **feladat -** *Készítsen egy parent.c és a child.c programokat. A parent.c elindít egy gyermek processzt, ami különbözik a szülőtől. A szülő megvárja a gyermek lefutását. A gyermek szöveget ír a szabványos kimenetre (5-ször) (pl. a hallgató neve és a neptunkód)!*
Mentés: parent.c, ill. child.c

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

int main (void) {
    pid_t pid;

    if ((pid = fork()) < 0){
        perror( s: "process error");
    }
    else if (pid == 0){
        if(execl( path: "./child", arg: "child", (char *) NULL) < 0){
            perror( s: "execl error");
        }
    }
    if (waitpid(pid, stat_loc: NULL, options: 0) < 0){
        perror( s: "wait error");
    }
    return 0;
}
```

parent.c kódja

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

int main(void) {
    for (int i = 0; i < 5; i++)
    {
        printf( format: "Toth Jozsef WI2GDP\n");
        sleep( seconds: 1);
    }
    return 0;
}
```

child.c kódja

4. **feladat -** *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! Mentés: neptunkodgyak4.c*

5. **feladat -** *A fork() rendszerhívással hozzon létre gyerekeket, várja meg és vizsgálja a befejeződési állapotokat (gyerekben: exit, abort, nullával való osztás)! Mentés: neptunkodgyak5.c*

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

    return 0;
}
```

6. **feladat -** *Adott a következő ütemezési feladat, ahol a RR ütemezési algoritmus használatával készítse el:*
Határozza meg a
a.) Ütemezze az adott időszelet alapján az egyes processzek 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*	*-*