JEGYZŐKÖNYV

Operációs rendszerek BSc

2022. tavasz féléves feladat

Készítette: **Salamon István**

Neptunkód: **FA6VDV**

1. feladat: IPC mechanizmus

A feladat leírása:

21. Írjon egy C programot, ami
egy másodfokú egyenlet megoldóképletét reprezentálja message queue(üzenetsoros) IPC mechanizmus segítségével.
A műveletvégzéshez szükséges adatokat egy bemeneti fájlból olvassa be, majd az adatokat és az eredményt adja vissza egy kimeneti fájlba.
A Bemeneti ill. kimeneti fájl struktúrája kötött!

Példa a bemeneti és kimeneti fájl struktúrájára:

Bemeneti fájl:
i (A megoldani kívánt egyenletek száma)
a b c

Kimeneti fájl:
a b c x y (Az a,b,c jelzi a bemeneti adatokat, az x,y pedig a kimeneti eredményeket)

A feladat elkészítésének lépései:

Üzenet soros kommunikáció

- Az egyik processz üzenetsort készít, más processzek

kapcsolódnak rá, és a sorba betehető, kivehető információ.

- Az üzenetsor processzektől függetlenül létezik.

Rendszerhívásai:

msgget(); // készítés, azonosítás
msgctl(); // kontroll, jellemzők // lekérdezése + megszüntetés
msgsnd(); // üzenet betétele a sorba
msgrcv(); // üzenet kivétele a sorból

send.c fájl:

```c
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    fgets(message.mesg_text,MAX,stdin);

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

    // display the message
    printf("Data send is : %s \n", message.mesg_text);

    return 0;
}
```

a send.c

- mesg_buffer struktúrában tárolja az adatokat
- Szabványos bemenetről kér be adatokat
- elküldi a msgsnd() rendszerhívással a unique key -el azonosított sorba.
- msgget() üzenetsor készítés, azonosítás
- msgid az azonosító

rcv.c fájl:

```c
#include <unistd.h>
#include <stdlib.h>
#include <math.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
    int a, b, c;
    double x1,x2;
} message, *ms;

double main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

    // msgrcv to receive message
    msgrcv(msgid, &message, sizeof(message), 1, 0);
    // read message convert to int
    sscanf(message.mesg_text,"%d:%d:%d",&message.a,&message.b, &message.c);
    //message.a=atoi(message.mesg_text);
    printf("Data Received is : %s \n",
                    message.mesg_text);
    double d; //temporary file sgrt
    d = message.b * message.b - 4.0 * message.a * message.c;
    //printf("number d: %lf \n",d);
    //result
    message.x1 = (-message.b + sqrt(d)) / (2.0 * message.a);
    message.x2 = (-message.b - sqrt(d)) / (2.0 * message.a);
    printf("number x1: %lf \n",message.x1);
    printf("number x2: %lf \n",message.x2);
    FILE* fp;
    fp = fopen("kimenet.txt", "w");  /* file-open, w = write*/
    if (fp != NULL) {
        fprintf(fp,"equals: %lf %lf \n",message.x1,message.x2);

        fclose(fp);                 /* file-close*/
    }
    else {
        perror("Nem sikerült megnyitni a fájlt");
    }

    // to dstroy the message queue
    msgctl(msgid, IPC_RMID, NULL);

    return 0;
```

A rcv.c

msgrcv() rendszerhívással kiolvassa az egyedi kulccsal azonosított üzenet sort

a sscanf() kiolvasom a sorokat átalakítom integer-re

d átmeneti változót használok a négyzetgyök alatti rész kiszámítására

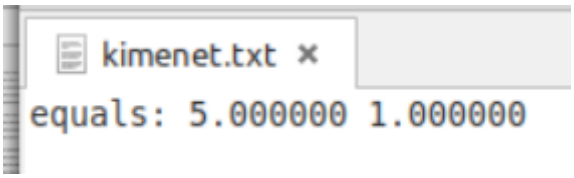
```
d = message.b * message.b - 4.0 * message.a * message.c;
```

majd alkalmazom a másodfokú egyenlet képletét

```
message.x1 = (-message.b + sqrt(d)) / (2.0 * message.a);
```

msgctl() rendszerhívással törlöm az üzenetet

kimeneti.txt



A futtatás eredménye:

először a send.c futtatom elküldöm az egyenlet együtthatóit 2:-12:10

másodszor a rcv.c futtatom megkapja a számokat kiírja az egyenlet eredményét x1, x2.

```
istvan@istvan-VirtualBox:~/hazi/beadando$ gcc rcv.c -o rcv -lm
istvan@istvan-VirtualBox:~/hazi/beadando$ ./send
Write Data : 2:-12:10
Data send is : 2:-12:10

istvan@istvan-VirtualBox:~/hazi/beadando$ ./rcv
Data Received is : 2:-12:10

number x1: 5.000000
number x2: 1.000000
```

mellékletek: send.c, rcv.c, kimenet.txt

2. feladat: OS algoritmusok

A feladat leírása:

19. Adott az alábbi terhelés esetén a rendszer. Határozza meg az *indulás, befejezés, várakozás/átlagos várakozás és körülfordulás/átlagos körülfordulás, válasz/átlagos válaszidő* és a *CPU kihasználtság* értékeket az FCFS ütemezési algoritmusok mellett! (cs: 0,1ms; sch: 0,1ms)

	P1	P2	P3	P4	P5
Érkezés	0	1	3	9	12
CPU idő	3	5	2	5	5
Indulás					
Befejezés					
Várakozás					

Ábrázolja Gantt diagram segítségével az *aktív/várakozó processzek* futásának menetét.

A feladat elkészítésének lépései:

FCFS (First Come, First Served)

A ready állapotban levő processzek a beérkezési sorrendjük szerint kapnak CPU-t.

várakozás	indulás	-	Érkezés
befejezés	befejezés	+	cpu idő
válasz idő	befejezés	-	indulás
cpu kihasznátság:	sch+szum cpu idő - context switch / szum cpu idő + cs		
körülfordulási idő	cpu idő	+	várakozás

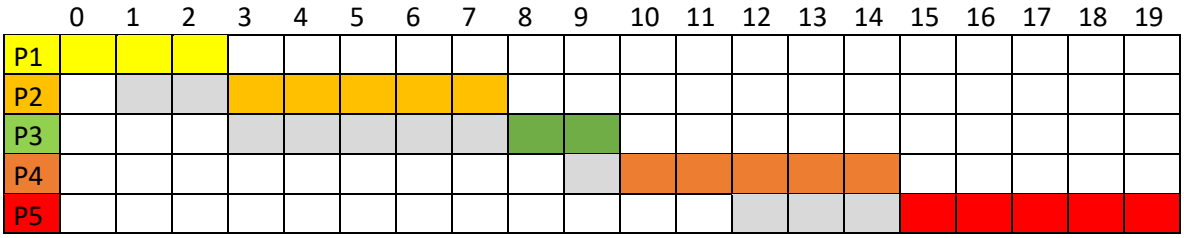
A futtatás eredménye

	Érkezés	Cpu idő
P1	0	3
P2	1	5
P3	3	2
P4	9	5
P5	12	5
	szum:	20

FcFs	Érkezés	Cpu idő	indulás	befejezés	várakozás	válasz idő	körülfordulási idő
P1	0	3	0	3	0	3	3
P2	1	5	3	8	2	5	7
P3	3	2	8	10	5	2	7
P4	9	5	10	15	1	5	6
P5	12	5	15	20	3	5	8
				szum	11	20	31

Átlagos várakozási idő:	(0+2+5+1+3)/5 =	2,20
átlagos körülfordulási idő:	(3+7+7+6+8)/5 =	6,2
átlagos válasz idő:	(3+5+2+5+5)/5 =	4
cpu kihasználtság:	5 context switch, (20,5-0,5/20,5) =	98%
cs	0,1 ms	
sch	0,1 ms	

Gantt diagramm:



melléklet: feleves feladat_fa6vdv.xlsx