

Operációs rendszerek BSc

10. gyakorlat

Készítette:

Nemesi Gergely Tibor
Üzemmérnök-informatikus
Neptun: ILZGJC

2022.04.11

Contents

I Bankár algoritmus, IPC mechanizmus	2
1. feladat	2
2. feladat	3
3. feladat	3
4. feladat	4
4c. feladat	7
5. feladat	8
5a. feladat	11

Part I

Bankár algoritmus, IPC
mechanizmus

1. feladat

Az előadáson bemutatott mintaprogram alapján készítse el a következő feladatot. Adott egy rendszerbe az alábbi erőforrások:

R (R1: 10; R2: 5; R3: 7)

A rendszerbe 5 processz van: P0, P1, P2, P3, P4

Kérdés:

Kielégíthető-e P1 (1,0,2), P4 (3,3,0) ill. P0 (0,2,0) kérése úgy, hogy biztonságos legyen, holtponmentesség szempontjából a rendszer - a következő kiinduló állapot alapján.

Külön-külön táblázatba oldja meg a feladatot!

- Határozza meg a processzek által igényelt erőforrások mátrixát?
- Határozza meg pillanatnyilag szabad erőforrások számát?
- Igazolja, magyarázza az egyes processzek végrehajtásának lehetséges sorrendjét - számolással?

MAX. IGÉNY				FOGLALÁS				KIELÉGÍTETLEN IGÉNYEK			
R1	R2	R3		R1	R2	R3		R1	R2	R3	
p0	7	5	3	0	1	0		7	4	3	
p1	3	2	2	2	0	0		1	2	2	
p2	9	0	2	3	0	2		6	0	0	
p3	2	2	2	2	1	1		0	1	1	
p4	4	3	3	0	0	2		4	3	1	
								KÉSZLET-IGÉNY			
				7	2	5		R1	R2	R3	
Foglaltak				10	5	7		-4	-1	-1	p0
Összesen				3	3	2		2	1	0	p1
Szabad erőforrás szám								-3	3	2	p2
								3	2	1	p3
								-1	0	1	p4

2. feladat

Készítsen C nyelvű programot, ahol egy szülő processz létrehoz egy csővezetékét, a gyerek processz beleír egy szöveget a csővezetékbe (A kiírt szöveg: XY neptunkod), a szülő processz ezt kiolvassa, és kiírja a standard kimenetre.

Mentés: neptunkod-unnamed.c

3. feladat

Készítsen C nyelvű programot, ahol egy szülő processz létrehoz egy nevesített csővezetékét (neve: neptunkod), a gyerek processz beleír egy szöveget a csővezetékbe (A hallgató neve: Keserű Ottó), a szülő processz ezt kiolvassa, és kiírja a standard kimenetre.

Mentés: neptunkod-named.c

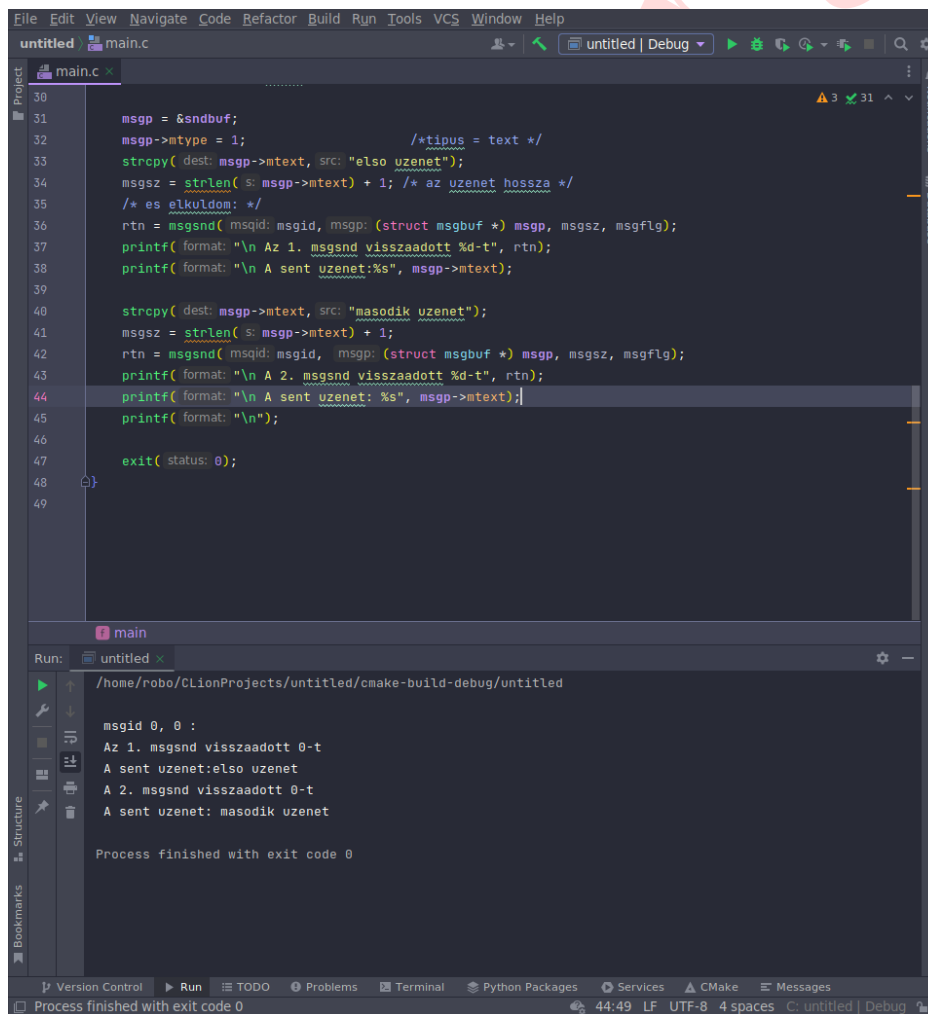
4. feladat

Először tanulmányozzák Vadász Dénes: Operációs rendszer jegyzet, a témához kapcsolódó fejezetét (5.3)., azaz

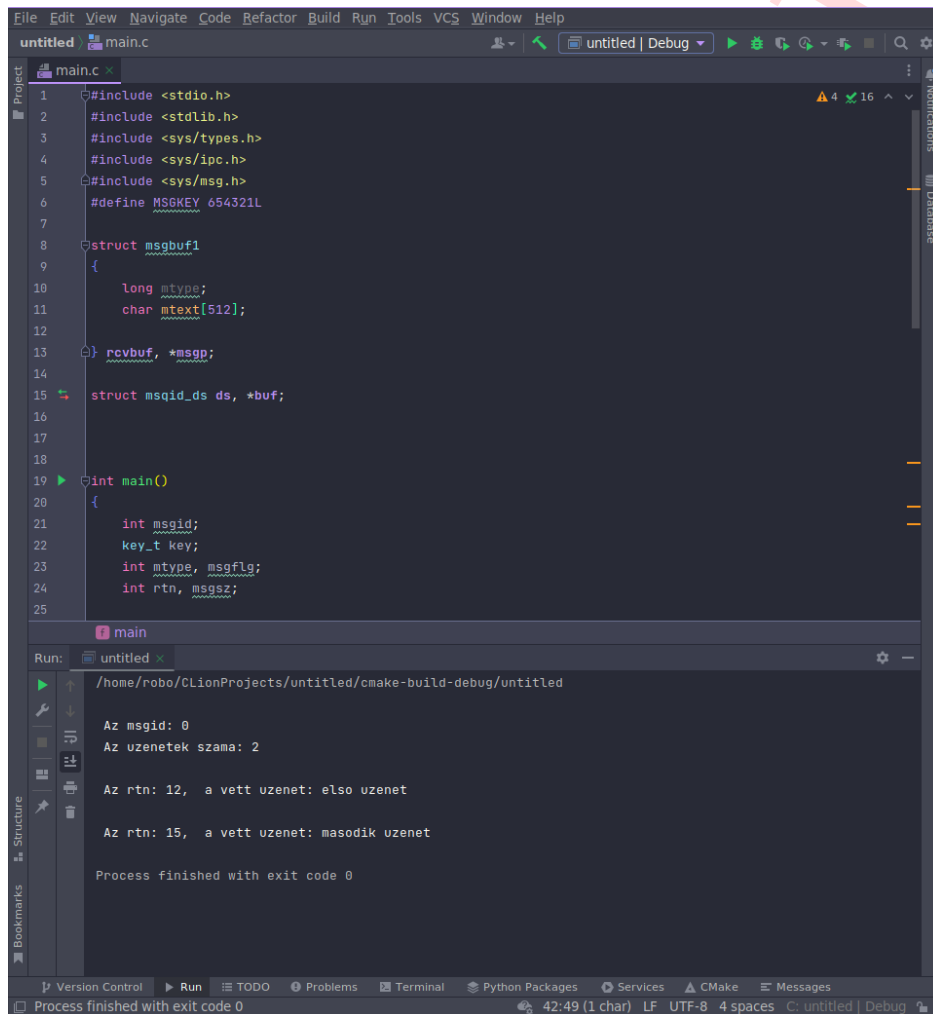
Írjon három C nyelvű programot, ahol készít egy üzenetsort és ebbe két üzenetet tesz bele – msgcreate.c, majd olvassa ki az üzenetet - msgrcv.c, majd szüntesse meg az üzenetsort (takarít) - msgctl.c.

A futtatás eredményét is tartalmazza a jegyzőkönyv.

Mentés: msgcreate.c; msgrcv.c; msgctl.c.



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled main.c
Project: main.c
30
31 msgp = &sndbuf;
32 msgp->mtype = 1; /*tipus = text */
33 strcpy(dest: msgp->mtext, src: "első üzenet");
34 msgsz = strlen(s: msgp->mtext) + 1; /* az üzenet hossza */
35 /* es elkuldöm: */
36 rtn = msgsnd(msgid: msgid, msgp: (struct msgbuf *) msgp, msgsz, msgflg);
37 printf(format: "\n Az 1. msgsnd visszaadott %d-t", rtn);
38 printf(format: "\n A sent üzenet: %s", msgp->mtext);
39
40 strcpy(dest: msgp->mtext, src: "második üzenet");
41 msgsz = strlen(s: msgp->mtext) + 1;
42 rtn = msgsnd(msgid: msgid, msgp: (struct msgbuf *) msgp, msgsz, msgflg);
43 printf(format: "\n A 2. msgsnd visszaadott %d-t", rtn);
44 printf(format: "\n A sent üzenet: %s", msgp->mtext);
45 printf(format: "\n");
46
47 exit(status: 0);
48
49
main
Run: untitled
/home/robo/CLionProjects/untitled/cmake-build-debug/untitled
msgid 0, 0 :
Az 1. msgsnd visszaadott 0-t
A sent üzenet:első üzenet
A 2. msgsnd visszaadott 0-t
A sent üzenet: második üzenet
Process finished with exit code 0
Process finished with exit code 0
44:49 LF UTF-8 4 spaces C: untitled | Debug
```



The screenshot shows an IDE with a C program named `main.c` and its execution output. The code defines a message buffer structure and a `main` function that interacts with a message queue. The output shows the program's execution, including the number of messages received and the content of the messages.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MSGKEY 654321L

struct msgbuf1
{
    long mtype;
    char mtext[512];
};
rcvbuf, *msgp;

struct msqid_ds ds, *buf;

int main()
{
    int msgid;
    key_t key;
    int mtype, msgflg;
    int rtn, msgsz;
```

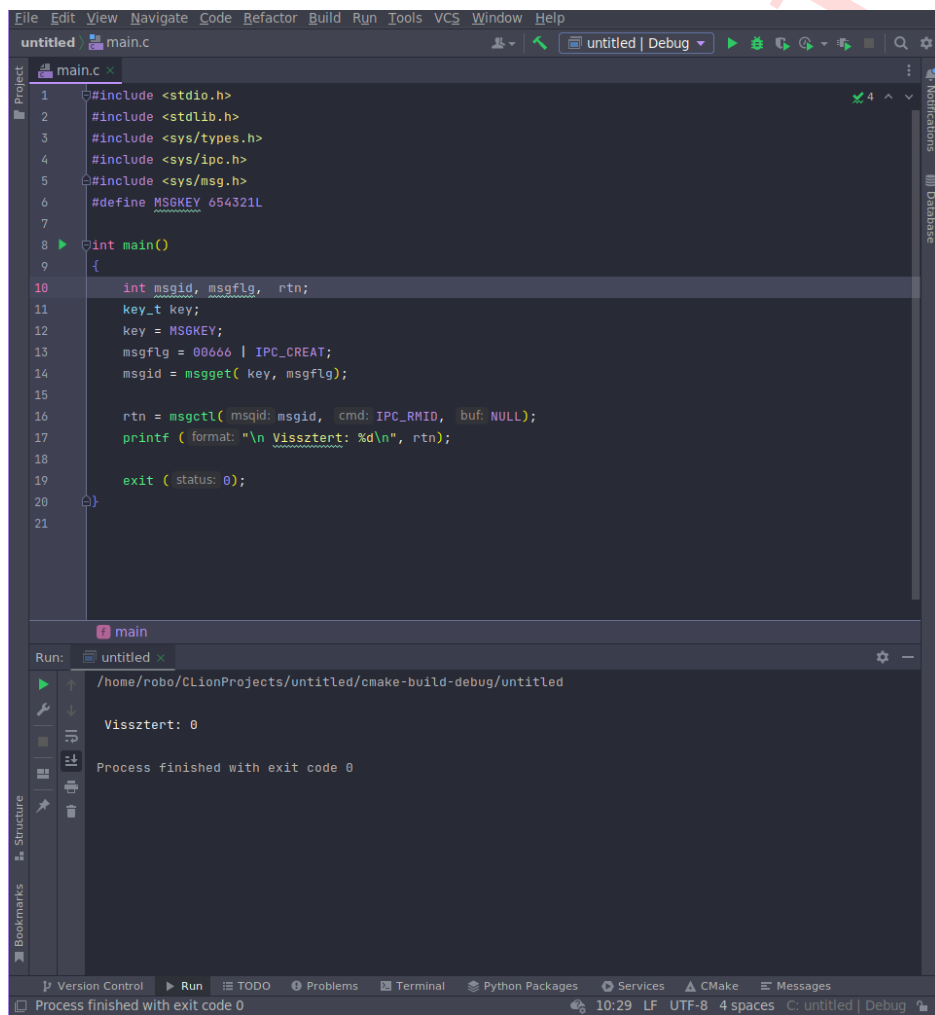
Run: untitled

/home/robo/CLionProjects/untitled/cmake-build-debug/untitled

Az msgid: 0
Az üzenetek száma: 2

Az rtn: 12, a vett üzenet: első üzenet
Az rtn: 15, a vett üzenet: második üzenet

Process finished with exit code 0



The screenshot shows the CLion IDE interface. The main editor displays a C program in `main.c`. The program includes `<stdio.h>`, `<stdlib.h>`, `<sys/types.h>`, `<sys/ipc.h>`, and `<sys/msg.h>`. It defines `MSGKEY` as `054321L`. The `main` function declares `msgid`, `msgflg`, and `rtn`. It sets `key` to `MSGKEY`, `msgflg` to `000000 | IPC_CREAT`, and `msgid` to `msgget(key, msgflg)`. It then calls `msgctl(msgid, msgid, cmd: IPC_RMID, buf: NULL)`, prints `Visszter: 0`, and exits with status 0.

The Run window shows the execution output: `Visszter: 0` and `Process finished with exit code 0`. The status bar at the bottom indicates the file is `untitled` in `Debug` mode, with a timestamp of 10:29, encoding of UTF-8, and 4 spaces.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/msg.h>
6 #define MSGKEY 054321L
7
8 int main()
9 {
10     int msgid, msgflg, rtn;
11     key_t key;
12     key = MSGKEY;
13     msgflg = 000000 | IPC_CREAT;
14     msgid = msgget( key, msgflg);
15
16     rtn = msgctl( msgid: msgid, cmd: IPC_RMID, buf: NULL);
17     printf ( format: "\n Visszter: %d\n", rtn);
18
19     exit ( status: 0);
20 }
21
```

Run: untitled x

/home/robo/CLionProjects/untitled/cmake-build-debug/untitled

Visszter: 0

Process finished with exit code 0

Version Control Run TODO Problems Terminal Python Packages Services CMake Messages

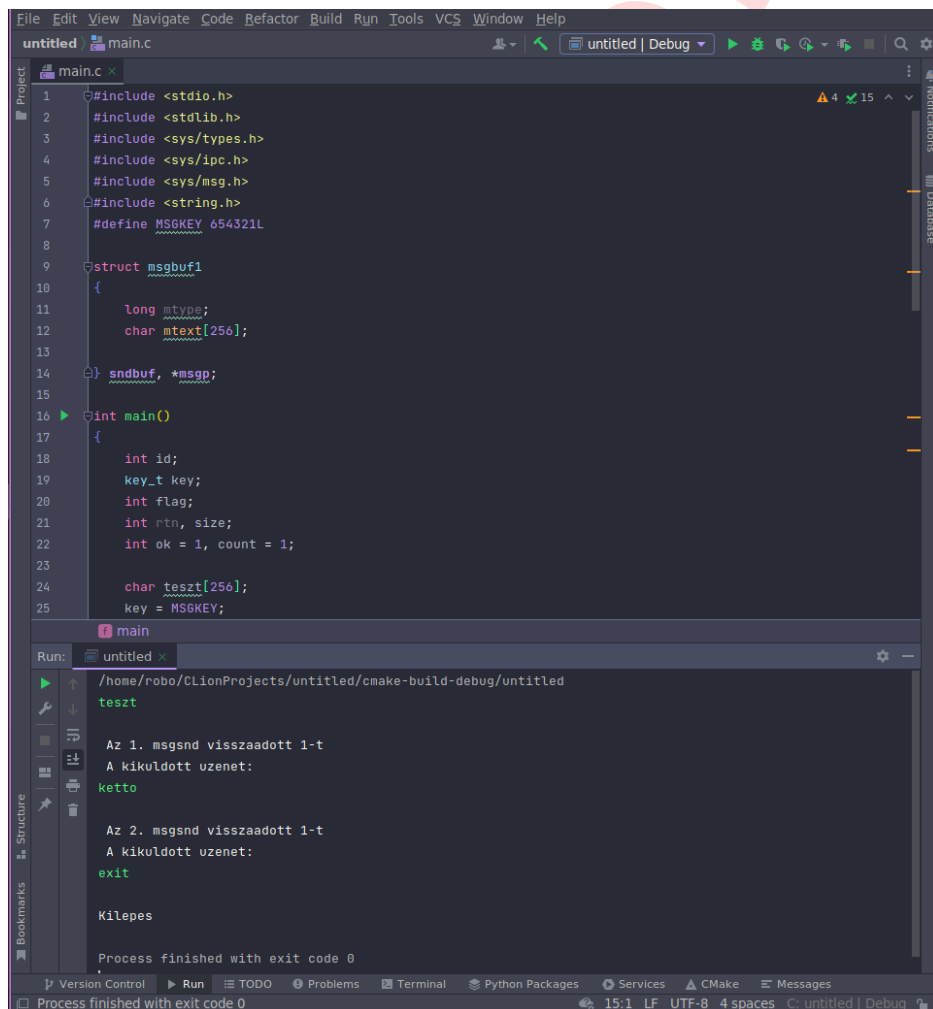
Process finished with exit code 0 10:29 LF UTF-8 4 spaces C: untitled | Debug

4c. feladat

- az egyik processz létrehozza az üzenetsort, és szövegeket küld bele, exit üzenetre kilép
- másik processzben lehet választani a feladatok közül: üzenetek darabszámának lekérdezése, 1 üzenet kiolvasása, összes üzenet kiolvasása, üzenetsor megszüntetése, kilépés

Mentés: gyak10-4.c

A futtatás eredményét is tartalmazza a jegyzőkönyv.



The screenshot shows a C program in a file named `main.c` within an IDE. The program defines a message queue and a structure for messages. It then runs a `main` function that sends two messages and receives two responses.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#define MSGKEY 654321L

struct msgbuf1
{
    long mtype;
    char mtext[256];
} sndbuf, *msgp;

int main()
{
    int id;
    key_t key;
    int flag;
    int rtn, size;
    int ok = 1, count = 1;

    char teszt[256];
    key = MSGKEY;
```

The output window shows the following text:

```
teszt
Az 1. msgsnd visszaadott 1-t
A kiküldött üzenet:
ketto

Az 2. msgsnd visszaadott 1-t
A kiküldött üzenet:
exit

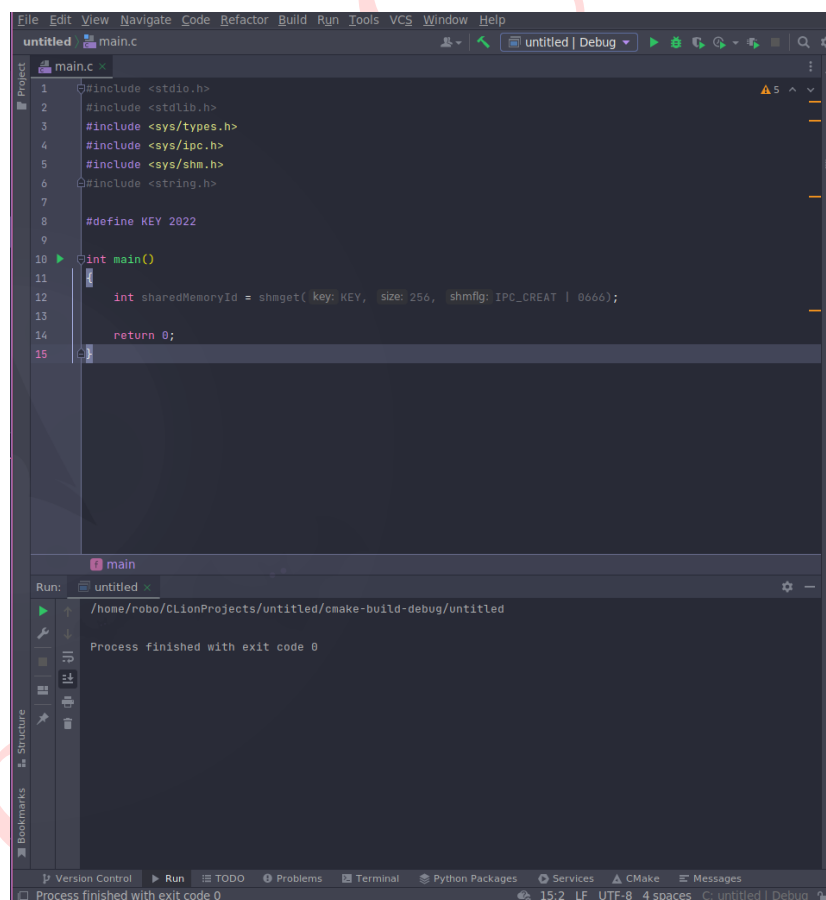
Kilépés

Process finished with exit code 0
```


5. feladat

Először tanulmányozzák Vadász Dénes: Operációs rendszer jegyzetét - a témához kapcsolódó fejezetét (5.3.2), azaz Írjon három C nyelvű programot, ahol

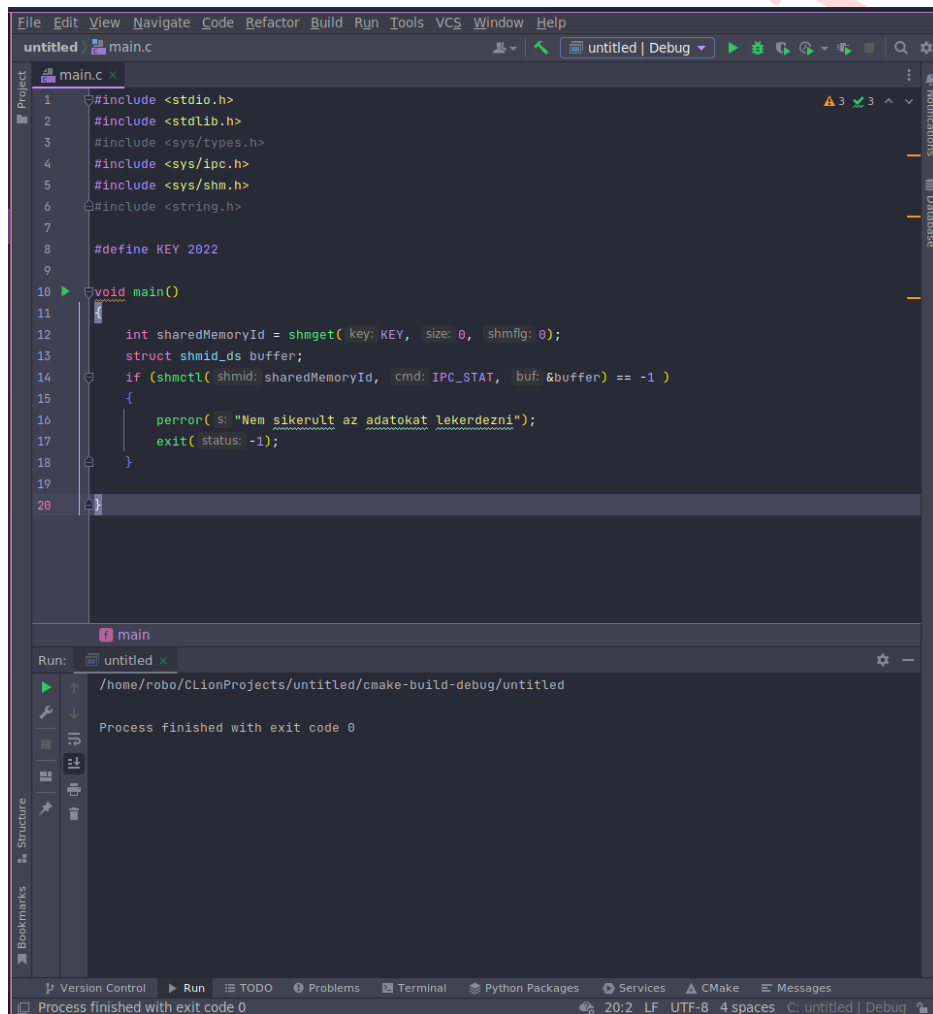
- készít egy osztott memóriát, melyben választott kulccsal kreál/azonosít osztott memória szegmenst - shmcreate.c.
- az shmcreate.c készített osztott memória szegmens státusának lekérdezése – shmctl.c
- opcionális: shmop.c shmctl-del azonosít osztott memória szegmenst. Ezután a segm nevű pointintervál-tozót használva a processz virtuális címtartományába kapcsolja (attach) a szegmenst (shmat()) rendszerhívás). Olvassa, írja ezt a címtartományt, végül lekapcsolja (detach) a shmdt() rendszerhívással).



```
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
untitled | Debug
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <string.h>
7
8 #define KEY 2022
9
10 int main()
11 {
12     int sharedMemoryId = shmget(KEY, 256, IPC_CREAT | 0666);
13
14     return 0;
15 }

Run:
main
/home/robo/CLionProjects/untitled/cmake-build-debug/untitled
Process finished with exit code 0

Version Control Run TODO Problems Terminal Python Packages Services CMake Messages
Process finished with exit code 0 15:2 LF UTF-8 4 spaces C: untitled | Debug
```

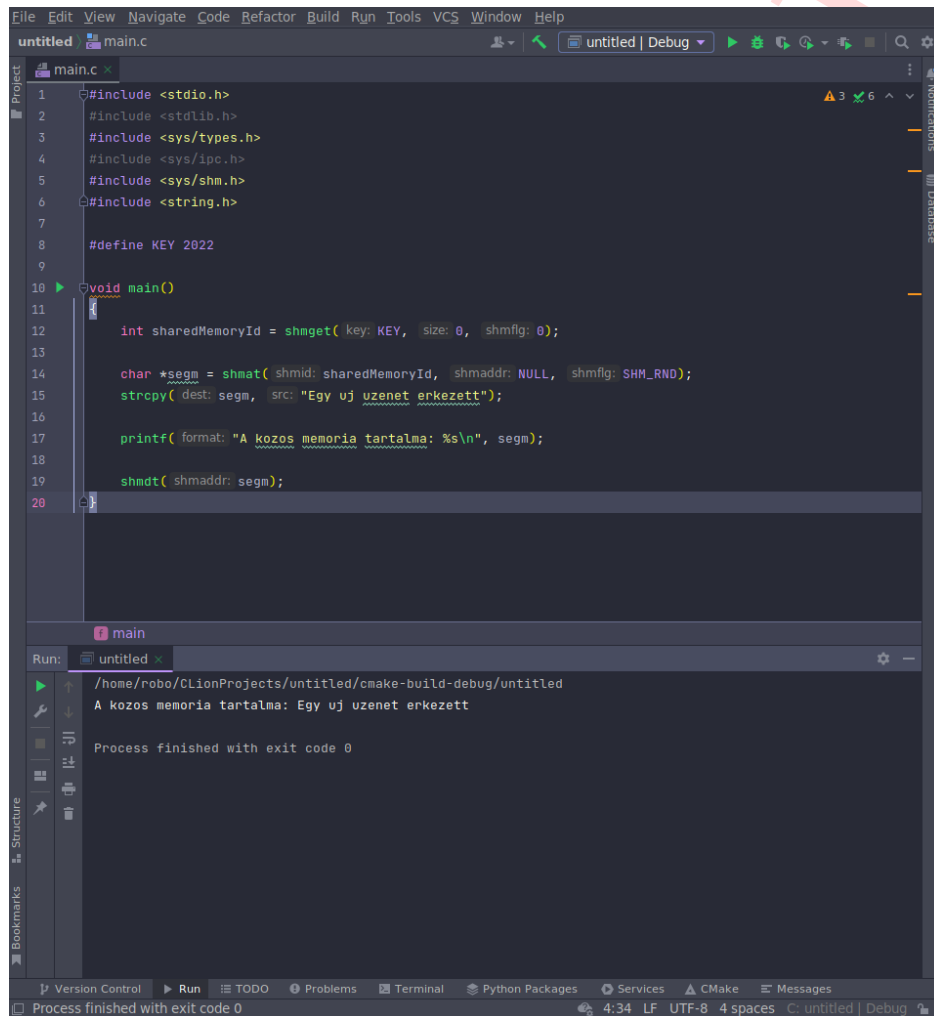


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <string.h>
7
8 #define KEY 2022
9
10 void main()
11 {
12     int sharedMemoryId = shmget( key: KEY, size: 0, shmflg: 0);
13     struct shmid_ds buffer;
14     if (shmctl( shmid: sharedMemoryId, cmd: IPC_STAT, buf: &buffer) == -1 )
15     {
16         perror( s: "Nem sikerult az adatokat lekerdezni");
17         exit( status: -1);
18     }
19 }
20
```

Run: untitled x

/home/robo/CLionProjects/untitled/cmake-build-debug/untitled

Process finished with exit code 0



The screenshot shows an IDE with a C program that demonstrates shared memory usage. The code includes headers for `stdio.h`, `stdlib.h`, `sys/types.h`, `sys/ipc.h`, `sys/shm.h`, and `string.h`. It defines a key `KEY` as 2022. The `main` function calls `shmget` to create a shared memory segment, `shmat` to attach it, `strcpy` to copy a message into it, `printf` to display the message, and `shmdt` to detach it.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <string.h>
7
8 #define KEY 2022
9
10 void main()
11 {
12     int sharedMemoryId = shmget( key: KEY, size: 0, shmflg: 0);
13
14     char *segm = shmat( shmid: sharedMemoryId, shmaddr: NULL, shmflg: SHM_RND);
15     strcpy( dest: segm, src: "Egy új üzenet érkezett");
16
17     printf( format: "A közös memória tartalma: %s\n", segm);
18
19     shmdt( shmaddr: segm);
20 }
```

The Run window shows the output of the program:

```
Run: untitled
/home/robo/CLionProjects/untitled/cmake-build-debug/untitled
A közös memória tartalma: Egy új üzenet érkezett
Process finished with exit code 0
```

5a. feladat

Írjon egy C nyelvű programot, melyben

- egyik processz létrehozza az osztott memóriát
- másik processz rácsatlakozik az osztott memóriára, ha van benne valamilyen szöveg, akkor kiolvassa, majd beleír új üzenetet
- harmadik processznél lehet választani a feladatok közül: státusz lekérése (szegmens mérete, utolsó shmop-os proc. pid-je), osztott memória megszüntetése, kilépés (2. és 3. proc. lehet egyben is)”

A futtatás eredményét is tartalmazza a jegyzőkönyv.

Mentés: gyak10-5.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <sys/ipc.h>
5 #include <sys/shm.h>
6 #include <string.h>
7 #include <unistd.h>
8
9 #define KEY 777777
10
11 void main()
12 {
13     pid_t process1;
14     pid_t process2;
15     pid_t process3;
16
17     process1 = fork();
18     if (process1 == 0)
19     {
20         int sharedMemoryId = shmget( key: KEY, size: 256, shmflg: IPC_CREAT | 0666);
21         if (sharedMemoryId == -1)
22         {
23             perror( s: "Nem sikerült lefoglalni a memoriát\n");
24             exit( status: -1);
25         }
26     }
27 }
```

Run: untitled x

```
/home/robo/CLionProjects/untitled/cmake-build-debug/untitled
Process1 lefoglalta a memoriát!
Process 2 olvas
Nincs benne szöveg
process2 küldte az üzenetet.
process3:
Szegmens mérete: 256
utolsó operációt kiadó processz pidje : 1280

Process finished with exit code 0
```