Επικοινωνία μεταξύ διεργασιών *a la* System V

- Πραγματικά ασύγχρονη επικοινωνία στον ίδιο υπολογιστή
- Καθολικά προσπελάσιμοι πόροι μέσω του πυρήνα του Unix
  - Ουρές μηνυμάτων
  - Κοινή μνήμη
  - Σηματοφόροι
- Ταυτοποίηση πόρων μέσω κλειδιών (τύπου `key_t`) και πρόσβαση στις εσωτερικές δομές τους (`struct msqid_ds`, `struct shmid_ds` και `struct semid_ds`) μέσω προσδιοριστών που επιστρέφονται και χρησιμοποιούνται από κατάλληλες κλήσεις συστήματος (οι οποίες, ως συνήθως, επιστρέφουν -1 σε περίπτωση λάθους)
- Δικαιώματα προστασίας πόρων (read/write) που καταχωρούνται σε δομές `struct ipc_perm`
- Μοναδικά κλειδιά (τύπου `key_t`) μπορούν να παραχθούν καλώντας τη συνάρτηση

  ```
  key_t ftok(const char *pathname, int proj_id)
  ```

  δίνοντάς της ένα όνομα-μονοπάτι `pathname` και έναν ακέραιο `proj_id`
- Απαιτήσεις
  - `#include <sys/types.h>`
  - `#include <sys/ipc.h>`

Συνάρτηση ftok

```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```

Τιμή επιστροφής σε λάθος.
Θέτει και το errno

```
if ((thekey = ftok("tmp/trouble.c", 1)) == (key_t)-1))
    perror("Failed to derive key from /tmp/trouble.c");
```

Πρέπει να είναι προσπελάσιμο
το αρχείο στη διαδικασία

Ουρές Μηνυμάτων

- Χρησιμεύουν στην ανταλλαγή μηνυμάτων μεταξύ διεργασιών
- Η αποστέλλουσα διεργασία επισυνάπτει έναν τύπο στο μήνυμα που στέλνει και η παραλαμβάνουσα διεργασία μπορεί να ζητήσει την παραλαβή μηνύματος συγκεκριμένου τύπου
- Πριν το σώμα ενός μηνύματος πρέπει να προηγείται ο τύπος του (long) σε bytes, δηλαδή στις κλήσεις συστήματος για αποστολή και παραλαβή μηνυμάτων χρειάζεται να δίνεται ένας δείκτης σε δομή σαν την

```
struct message {
    long mtype;
    char mtext[MSGSIZE]; };
```

- Απαίτηση: `#include <sys/msg.h>`

Κλήση msgget

- `int msgget(key_t key, int msgflag)`
- Επιστρέφει έναν προσδιοριστή για την ουρά μηνυμάτων που αντιστοιχεί στο κλειδί `key`
- Το `msgflag` είναι ένας οδέραιος όπου τίθενται τα επιθυμητά δικαιώματα προστασίας της ουράς μηνυμάτων, καθώς επίσης και πρόσθετες απαιτήσεις (υπό τη μορφή διάζευξης συμβολικών ονομάτων) σχετικές με τη δημιουργία της ουράς μηνυμάτων, όπως:

  `IPC_CREAT`: Αν δεν υπάρχει πόρος (ουρά μηνυμάτων) που αντιστοιχεί στο `key` να δημιουργηθεί νέος (αντί να επιστραφεί λάθος), ενώ αν υπάρχει πόρος, να προσπελασθεί αυτός

  Θυμίζουν δημιουργία αρχείων

  `IPC_EXCL`: Σε συνδυασμό με το προηγούμενο, αν δεν υπάρχει πόρος να δημιουργηθεί, αλλά αν υπάρχει να επιστραφεί λάθος

- Περιπτωσιολογία για το `msgflag`

	PERMS	PERMS|IPC_CREAT	PERMS|IPC_CREAT|IPC_EXCL
υπάρχει πόρος	χρήση του πόρου	χρήση του πόρου	λάθος
δεν υπάρχει πόρος	λάθος	δημιουργία και χρήση νέου πόρου	δημιουργία και χρήση νέου πόρου

Κλήσεις msgrcv, msgsnd

- `int msgrcv(int msqid, void *ptr, int len, long type, int flag)`
- `int msgsnd(int msqid, void *ptr, int len, int flag)`
- Με την msgrcv παραλαμβάνεται ένα μήνυμα τύπου type, επιστρέφοντας το μέγεθος του μηνύματος, από την ουρά μηνυμάτων με προσδιοριστή msqid (αν το type είναι 0, παραλαμβάνεται το πρώτο μήνυμα, ανεξαρτήτως τύπου) και με την msgsnd αποστέλλεται ένα μήνυμα στην ουρά
- Ο τύπος και το σώμα του μηνύματος βρίσκονται σε μία δομή στην οποία δείχνει το ptr και το len είναι το μέγεθος του σώματος του μηνύματος για την msgsnd ή το μέγιστο του μεγέθους αυτού για την msgrcv
- Στο flag τίθεται το 0, εκτός ειδικών περιπτώσεων

Παράδειγμα Δομής για msgrcv, msgsnd

```
struct mymsg {
  long mtype;
  char text[MAXLEN];
} mymsg_t;
```

Το μήνυμα γράφεται στο text

Το mtype δηλώνει τον τύπο του μηνύματος

Η msgrcv μπορεί να δηλώνει τι τύπο μηνύματος περιμένει να λάβει

Κλήση msgctl

- int msgctl(int msqid, int cmd,
  struct msqid_ds *buff)
- Εκτελεί την ενέργεια cmd επάνω στην ουρά μηνυμάτων που αντιστοιχεί στον προσδιοριστή msqid
- Μία ενέργεια είναι η IPC_STAT με την οποία συμπληρώνονται τα πεδία της δομής *buff με τα χαρακτηριστικά της ουράς μηνυμάτων που ενδιαφέρει
- Η συνηθέστερη ενέργεια είναι η IPC_RMID που καταστρέφει την ουρά μηνυμάτων (στο buff αρκεί να τεθεί 0 αφού προσαρμοσθεί σε (struct msqid_ds *))

Δομή msqid_ds

```
struct ipc_perm msg_perm; /* operation permission structure */
msgqnum_t msg_qnum;       /* number of messages currently in queue */
msglen_t msg_qbytes;      /* maximum bytes allowed in queue */
pid_t msg_lspid;          /* process ID of msgsnd */
pid_t msg_lrpid;          /* process ID of msgrcv */
time_t msg_stime;         /* time of last msgsnd */
time_t msg_rtime;         /* time of last msgrcv */
time_t msg_ctime;         /* time of last msgctl */
```

- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μεταξύ τους μέσω ουρών μηνυμάτων.*

```
/* File: msgq_server.c */
#include <sys/types.h>                          /* For System V IPC */
#include <sys/ipc.h>                            /* For System V IPC */
#include <sys/msg.h>                          /* For message queues */
#include <stdio.h>                                      /* For I/O */
#include <stdlib.h>                                    /* For exit */
#include <string.h>                          /* For strcpy, strlen */
#define MSGSIZE 256          /* Maximum size of message to communicate */
#define KEY (key_t)5678                 /* Key value of message queue */
#define PERMS 0666                    /* Permissions of message queue */
#define SERVER_MTYPE 27L            /* Type of message sent by server */
#define CLIENT_MTYPE 42L            /* Type of message sent by client */

struct message {                                /* Message structure */
   long mtype;
   char mtext[MSGSIZE]; };

main()
{  int qid;
   struct message sbuf, rbuf;
   if ((qid = msgget(KEY, PERMS | IPC_CREAT)) < 0) {
      perror("msgget"); exit(1); }             /* Create message queue */
   printf("Created message queue with identifier %d\n", qid);
   sbuf.mtype = SERVER_MTYPE;                 /* Server's message type */
   strcpy(sbuf.mtext, "A message from server");     /* Server's message */
   if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0) {
      perror("msgsnd"); exit(1); }                     /* Send message */
   printf("Sent message:     %s\n", sbuf.mtext);
   if (msgrcv(qid, &rbuf, MSGSIZE, CLIENT_MTYPE, 0) < 0) {
      perror("msgrcv"); exit(1); }                  /* Receive message */
   printf("Received message: %s\n", rbuf.mtext);
   if (msgctl(qid, IPC_RMID, (struct msqid_ds *) 0) < 0) {
      perror("msgctl"); exit(1); }            /* Destroy message queue */
   printf("Removed message queue with identifier %d\n", qid); }
```

```
/* File: msgq_client.c */
#include <sys/types.h>                          /* For System V IPC */
#include <sys/ipc.h>                            /* For System V IPC */
#include <sys/msg.h>                          /* For message queues */
#include <stdio.h>                                      /* For I/O */
#include <stdlib.h>                                    /* For exit */
#include <string.h>                          /* For strcpy, strlen */
#define MSGSIZE 256          /* Maximum size of message to communicate */
#define KEY (key_t)5678                /* Key value of message queue */
#define PERMS 0666                   /* Permissions of message queue */
#define SERVER_MTYPE 27L           /* Type of message sent by server */
#define CLIENT_MTYPE 42L           /* Type of message sent by client */

struct message {                                 /* Message structure */
   long mtype;
   char mtext[MSGSIZE]; };

main()
{  int qid;
   struct message sbuf, rbuf;
   if ((qid = msgget(KEY, PERMS)) < 0) {
      perror("msgget"); exit(1); }            /* Access message queue */
   printf("Accessing message queue with identifier %d\n", qid);
   if (msgrcv(qid, &rbuf, MSGSIZE, SERVER_MTYPE, 0) < 0) {
      perror("msgrcv"); exit(1); }               /* Receive message */
   printf("Received message: %s\n", rbuf.mtext);
   sbuf.mtype = CLIENT_MTYPE;                /* Client's message type */
   strcpy(sbuf.mtext, "A message from client");    /* Client's message */
   if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0) {
      perror("msgsnd"); exit(1); }                   /* Send message */
   printf("Sent message:     %s\n", sbuf.mtext); }
```

Εκτέλεση (πρώτα ο server)

```
$ ./msgq_server
Created message queue with identifier 98304
Sent message:     A message from server
Received message: A message from client
Removed message queue with identifier 98304
$
```

```
$ ./msgq_client
Accessing message queue with identifier 98304
Received message: A message from server
Sent message:     A message from client
$
```

Κοινή Μνήμη

- Δυνατότητα επικοινωνίας μεταξύ διεργασιών μέσω καταχώρησης και ανάγνωσης πληροφοριών σε περιοχή μνήμης που είναι προσπελάσιμη από όλες τις διεργασίες
- Ανάγκη συγχρονισμού των διεργασιών, συνήθως μέσω σηματοφόρων
- Απαίτηση: `#include <sys/shm.h>`
- Κλήση συστήματος `shmget`
  - `int shmget(key_t key, int size, int shmflag)`
  - Επιστρέφει έναν προσδιοριστή για την κοινή μνήμη μεγέθους `size` που αντιστοιχεί στο κλειδί `key`
  - Στο `shmflag` τίθενται τα επιθυμητά δικαιώματα προστασίας καθώς και πρόσθετες απαιτήσεις (`IPC_CREAT` και `IPC_EXCL`, με την ίδια σημασία που έχουν και στις ουρές μηνυμάτων) σχετικές με τη δημιουργία της κοινής μνήμης

Επιστρέφει -1 σε αποτυχία

Κλήσεις shmat, shmdt

- Κλήση συστήματος shmat
  - char *shmat(int shmid, char *addr, int flag)
  - Προσαρτά την κοινή μνήμη που αντιστοιχεί στον προσδιοριστή shmid στην περιοχή μνήμης που έχει πρόσβαση η καλούσα διεργασία και επιστρέφει την κατάλληλη διεύθυνση
  - Μέσω των addr και flag μπορεί να ζητηθεί η προσάρτηση σε συγκεκριμένη περιοχή μνήμης, αλλά η συνηθισμένη χρήση της shmat είναι να αφεθεί η επιλογή αυτή στον πυρήνα θέτοντας 0 στα addr και flag (το πρώτο προσαρμοσμένο σε (char *))
- Κλήση συστήματος shmdt
  - int shmdt(char *addr)
  - Αποσπά την προσαρτημένη στο addr κοινή μνήμη

Κλήση shmctl

- int shmctl(int shmid, int cmd,
  struct shmid_ds *buff)
- Εκτελεί την ενέργεια cmd στην κοινή μνήμη που αντιστοιχεί στον προσδιοριστή shmid
- Αντίστοιχα με τις δυνατότητες που υπάρχουν για τις ουρές μηνυμάτων (μέσω της msgctl), με την ενέργεια IPC_STAT συμπληρώνονται τα πεδία της δομής *buff με τα χαρακτηριστικά της κοινής μνήμης, ενώ η συνηθέστερη ενέργεια είναι η IPC_RMID που καταστρέφει την κοινή μνήμη (στο buff αρκεί να τεθεί 0 αφού προσαρμοσθεί σε (struct shmid_ds *))

Δομή shmid_ds

```
struct ipc_perm shm_perm; /* operation permission structure */
size_t shm_segsz;         /* size of segment in bytes */
pid_t shm_lpid;           /* process ID of last operation */
pid_t shm_cpid;           /* process ID of creator */
shmatt_t shm_nattch;      /* number of current attaches */
time_t shm_atime;         /* time of last shmat */
time_t shm_dtime;         /* time of last shmdt */
time_t shm_ctime;         /* time of last shctl */
```

Παραδείγματα Χρήσης

Ένας μόνο τη δημιουργεί

```
int id;
id = shmget(IPC_PRIVATE, 10, 0666);
if ( id==-1 ) perror("CREATION");
```

Όλοι την προσαρτούν

```
int *mem;
mem = (int *)shmat(id, (void *)0, 0);
if ((int)mem == -1 ) perror("Attachment.");
```

Όλοι την απομακρύνουν

```
int err;
err = shmdt((void *)mem);
if (err == -1 ) perror("Detachment");
```

Ένας μόνο την αποδεσμεύει

```
int err;
err = shmctl(id, IPC_RMID, 0);
if ( err == -1 ) perror("Removal");
```

Παράδειγμα 1

```
/* shareMem1.c -- Y. Alagiannis-A.Delis */
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char **argv){
  int id=0, err=0;
  int *mem;
  /* Make shared memroy segment */
  id = shmget(IPC_PRIVATE,10,0666);
  if (id == -1)  perror ("Creation");
  else   printf("Allocated. %d\n",(int)id);
  /* Attach the segment */
  mem = (int *) shmat(id, (void*)0, 0);
  if ((int) mem == -1)  perror("Attachment.");
  else  printf("Attached. Mem contents %d\n",*mem);

  /* Give it initial value */
  *mem=1;
  printf("Start other process. >"); getchar();
  /* Print out new value */
  printf("mem is now %d\n", *mem);
  /* Remove segment */
  err = shmctl(id, IPC_RMID, 0);
  if (err == -1)  perror ("Removal.");
  else  printf("Removed. %d\n", (int)(err));
  return 0;
}
```

Παράδειγμα 2

```
/* shareMem2.c: Y. Alagiannis-A.Delis */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(int argc, char **argv) {
  int id, err;
  int *mem;
  if (argc <= 1) {printf("Need shmem id. \n"); exit(1); }
  /* Get id from command line. */
  sscanf(argv[1], "%d", &id);
  printf("Id is %d\n", id);

  /* Attach the segment */
  mem = (int *) shmat(id, (void*) 0,0);
  if ((int) mem == -1) perror("Attachment.");
  else printf("Attached. Mem contents %d\n",*mem);

  /* Give it a different value */
  *mem=2;
  printf("Changed mem is now %d\n", *mem);
  /* Detach segment */
  err = shmdt((void *) mem);
  if (err == -1) perror ("Detachment.");
  else printf("Detachment %d\n", err);
  return 0;
}
```

Χρήση

Output from shareMem1.c

Output from shareMem2.c

./out1

Allocated. 38731782

Attached. Mem contents 0

Start other process. >

./out2 38731782

Id is 38731782

Attached. Mem contents 1

Changed mem is now 2

Detachment 0

Start other process. >s

mem is now 2

Removed. 0

Σηματοφόροι

- Μηχανισμός συγχρονισμού διεργασιών για την αποκλειστική διαχείριση κοινών πόρων (π.χ. κοινής μνήμης)
- Πριν την είσοδο σε κρίσιμο τμήμα του προγράμματός της, μία διεργασία ζητά την απαιτούμενη άδεια από έναν ελεγκτή σηματοφόρο (αναμένοντας, αν χρειάζεται, μέχρι να της δοθεί, οπότε δεσμεύει το απαιτούμενο μέρος του πόρου που ελέγχει ο σηματοφόρος) και μετά την έξοδο από το κρίσιμο τμήμα αποδεσμεύει το δεσμευμένο μέρος του πόρου
- Η δέσμευση γίνεται με κατάλληλη μείωση (DOWN ή P λειτουργία κατά Dijkstra) της τιμής του σηματοφόρου, εφ' όσον μετά τη μείωση η νέα τιμή θα είναι ≥ 0, και η αποδέσμευση με κατάλληλη αύξηση (UP ή V λειτουργία κατά Dijkstra) της τιμής του σηματοφόρου

Σηματοφόροι (συν)

- Με τις κλήσεις συστήματος που θα ακολουθήσουν δημιουργούνται και χρησιμοποιούνται σύνολα από σηματοφόρους και ο πυρήνας εγγυάται ότι ένα σύνολο λειτουργιών επάνω σε ένα τέτοιο σύνολο σηματοφόρων είναι ατομική διαδικασία Αν έχουμε > 1 προστατευόμενους πόρους μπορούμε να τους "κλειδώνουμε" ταυτόχρονα όλους
- Απαίτηση: `#include <sys/sem.h>`
- Κλήση συστήματος `semget`
  - `int semget(key_t key, int nsems, int semflag)`
  - Επιστρέφει έναν προσδιοριστή για ένα σύνολο από `nsems` σηματοφόρους που αντιστοιχεί στο κλειδί `key`
  - Στο `semflag` τίθενται τα επιθυμητά δικαιώματα προστασίας καθώς και πρόσθετες απαιτήσεις (`IPC_CREAT` και `IPC_EXCL`, με την ίδια σημασία που έχουν στις ουρές μηνυμάτων και στην κοινή μνήμη) σχετικές με τη δημιουργία του συνόλου σημαφόρων

Κλήση συστήματος semop

- `int semop(int semid, struct sembuf *opstr, int nops)`
- Εκτελεί στο σύνολο σημαφόρων που προσδιορίζονται από το `semid` τις λειτουργίες που καθορίζονται σε ένα πίνακα μεγέθους `nops` από δομές `struct sembuf` το πρώτο στοιχείο του οποίου δείχνει το `opstr`
- Η δομή `struct sembuf` ορίζεται σαν

```
struct sembuf {
    short sem_num;
    short sem_op;
    short sem_flg; };
```

και περιγράφει τη λειτουργία μεταβολής της τιμής του υπ' αριθμόν `sem_num` σημαφόρου του συνόλου (από 0 έως `nsems-1`) κατά `sem_op` (<0 για δέσμευση και >0 για αποδέσμευση), ενώ το `sem_flg` τίθεται συνήθως 0, εκτός ειδικών περιπτώσεων

Παράδειγμα Πρόσβασης σε 2 Ταινίες

```
#include <sys/sem.h>

void setsembuf(struct sembuf *s, int num, int op, int flg) {
   s->sem_num = (short)num;
   s->sem_op = (short)op;
   s->sem_flg = (short)flg;
   return;
}

struct sembuf get_tapes[2];
struct sembuf release_tapes[2];

setsembuf(&(get_tapes[0]), 0, -1, 0);
setsembuf(&(get_tapes[1]), 1, -1, 0);
setsembuf(&(release_tapes[0]), 0, 1, 0);
setsembuf(&(release_tapes[1]), 1, 1, 0);

Process 1:    semop(S, get_tapes, 1);
        <use tape A>
        semop(S, release_tapes, 1);

Process 2: semop(S, get_tapes, 2);
        <use tapes A and B>
        semop(S, release_tapes, 2);

Process 3: semop(S, get_tapes + 1, 1);
        <use tape B>
        semop(S, release_tapes + 1, 1);
```

- Πρώτο Σηματοφόρο (0)
- Ενέργεια = -1
- Σημαία (flag) = 0

Προσπάθησε να αφαιρέσεις 1 (P) από τον πρώτο σηματοφόρο

Προσπάθησε να αφαιρέσεις 1 (P) από τους 2 πρώτους σηματοφόρους

Προσπάθησε να αφαιρέσεις 1 (P) από το δεύτερο σηματοφόρο

Προσπάθησε να προσθέσεις 1 (V) στο δεύτερο σηματοφόρο

Κλήση Συστήματος semctl

- `int semctl(int semid, int semnum, int cmd, union semun arg)`
- Εκτελεί την ενέργεια cmd στον semnum σηματοφόρο του συνόλου σηματοφόρων (ή, ανάλογα με την ενέργεια, σε ολόκληρο το σύνολο) που αντιστοιχεί στον προσδιοριστή semid
- Η ένωση union semun είναι ορισμένη σαν

```
union semun {
    int              val;
    struct semid_ds *buff;
    unsigned short  *array; };
```

και χρησιμεύει για να καλύψει όλες τις δυνατότητες που υπάρχουν για έλεγχο συνόλων σηματοφόρων
- Με την ενέργεια IPC_STAT συμπληρώνονται τα πεδία της δομής *(arg.buff) με τα χαρακτηριστικά του συνόλου σηματοφόρων
- Με την ενέργεια SETVAL τίθεται σαν τιμή ενός σηματοφόρου το arg.val και με την ενέργεια GETVAL επιστρέφει η semctl την τιμή του σηματοφόρου
- Με τις ενέργειες SETALL και GETALL τίθενται τιμές στους σηματοφόρους του συνόλου (από τον πίνακα στην αρχή του οποίου δείχνει το arg.array) ή επιστρέφονται οι τιμές των σηματοφόρων (στον πίνακα arg.array)
- Με την ενέργεια IPC_RMID καταστρέφεται το σύνολο των σηματοφόρων

Δομή semid_ds

```
struct ipc_perm sem_perm; /* operation permission structure */
unsigned short sem_nsems; /* number of semaphores in the set */
time_t sem_otime;         /* time of last semop */
time_t sem_ctime;         /* time of last semctl */
```

- *Να γραφούν client και server προγράμματα C που να επιδεικνύουν την επικοινωνία μεταξύ τους μέσω κοινής μνήμης και το συγχρονισμό τους μέσω σηματοφόρων.*

```
/* File: shm_sem_server.c */
#include <sys/types.h>                                  /* For System V IPC */
#include <sys/ipc.h>                                    /* For System V IPC */
#include <sys/shm.h>                                   /* For shared memory */
#include <sys/sem.h>                                     /* For semaphores */
#include <stdio.h>                                              /* For I/O */
#include <stdlib.h>                                            /* For exit */
#include <string.h>                                 /* For strcpy, strlen */
#define SHMKEY (key_t)4321                  /* Key value of shared memory */
#define SEMKEY (key_t)9876                  /* Key value of semaphore set */
#define SHMSIZ 256                               /* Size of shared memory */
#define PERMS 0600    /* Permissions of shared memory and semaphore set */
union semun {                                    /* Union for semaphores */
   int val; struct semid_ds *buff; unsigned short *array; };
main()
{  int shmid, semid; char line[128], *shmem;
   struct sembuf oper[1] = {0, 1, 0}; union semun arg;
   if ((shmid = shmget(SHMKEY, SHMSIZ, PERMS | IPC_CREAT)) < 0) {
      perror("shmget"); exit(1); }              /* Create shared memory */
   printf("Created shared memory region with identifier %d\n", shmid);
   if ((semid = semget(SEMKEY, 1, PERMS | IPC_CREAT)) < 0) {
      perror("semget"); exit(1); }  /* Create semaphore set with 1 item */
   printf("Created semaphore with identifier %d\n", semid);
   arg.val=0;
   if (semctl(semid, 0, SETVAL, arg) < 0) {
      perror("semctl"); exit(1); }  /* Initialize semaphore for locking */
   printf("Initialized semaphore to lock shared memory region\n");
   if ((shmem = shmat(shmid, (char *) 0, 0)) == (char *) -1) {
      perror("shmat"); exit(1); }/* Attach shared memory region locally */
   printf("Attached shared memory region\nGive input line: ");
   fgets(line, sizeof line, stdin); line[strlen(line)-1] = '\0';
   strcpy(shmem, line);               /* Write message to shared memory */
   printf("Wrote to shared memory region: %s\n", line);
   if (semop(semid, &oper[0], 1) < 0) {
      perror("semop"); exit(1); }       /* Make shared memory available */
   shmdt(shmem);                          /* Detach shared memory region */
   printf("Released shared memory region\n"); }
```

```
/* File: shm_sem_client.c */
#include <sys/types.h>                          /* For System V IPC */
#include <sys/ipc.h>                            /* For System V IPC */
#include <sys/shm.h>                           /* For shared memory */
#include <sys/sem.h>                             /* For semaphores */
#include <stdio.h>                                       /* For I/O */
#include <stdlib.h>                                     /* For exit */
#define SHMKEY (key_t)4321                /* Key value of shared memory */
#define SEMKEY (key_t)9876                /* Key value of semaphore set */
#define SHMSIZ 256                             /* Size of shared memory */
#define PERMS 0600    /* Permissions of shared memory and semaphore set */
main()
{  int shmid, semid;
   char *shmem;
   struct sembuf oper[1] = {0, -1, 0};
   if ((shmid = shmget(SHMKEY, SHMSIZ, PERMS)) < 0) {
      perror("shmget"); exit(1); }              /* Access shared memory */
   printf("Accessing shared memory region with identifier %d\n", shmid);
   if ((semid = semget(SEMKEY, 1, PERMS)) < 0) {
      perror("semget"); exit(1); }              /* Access semaphore set */
   printf("Accessing semaphore with identifier %d\n", semid);
   if ((shmem = shmat(shmid, (char *) 0, 0)) == (char *) -1) {
      perror("shmat"); exit(1); }/* Attach shared memory region locally */
   printf("Attached shared memory region\n");
   printf("Asking for access to shared memory region\n");
   if (semop(semid, &oper[0], 1) < 0) {
      perror("semop"); exit(1); }/* Ask if you may access shared memory */
   printf("Read from shared memory region: %s\n", shmem);  /* Accessing */
   shmdt(shmem);                           /* Detach shared memory region */
   if (shmctl(shmid, IPC_RMID, (struct shmid_ds *) 0) < 0) {
      perror("shmctl"); exit(1); }             /* Destroy shared memory */
   printf("Removed shared memory region with identifier %d\n", shmid);
   if (semctl(semid, 0, IPC_RMID, 0) < 0) {
      perror("semctl"); exit(1); }             /* Destroy semaphore set */
   printf("Removed semaphore with identifier %d\n", semid); }
```

Εκτέλεση Προγράμματος

```
$ ./shm_sem_server
Created shared memory region with identifier 163840
Created semaphore with identifier 131072
Initialized semaphore to lock shared memory region
Attached shared memory region
Give input line: To be saved in shared memory
Wrote to shared memory region: To be saved in shared memory
Released shared memory region
$
```

```
$ ./shm_sem_client
Accessing shared memory region with identifier 163840
Accessing semaphore with identifier 131072
Attached shared memory region
Asking for access to shared memory region
Read from shared memory region: To be saved in shared memory
Removed shared memory region with identifier 163840
Removed semaphore with identifier 131072
$
```