ΕΠΛ421 - Προγραμματισμός Συστημάτων

Διάλεξη 16:

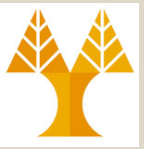
Επικοινωνία μεταξύ Διεργασιών (Inter-Process Communication (IPC))

Ουρές Μηνυμάτων (Message Queues) – Κοινή Μνήμη (Shared Memory) – Σηματοφόροι (Semaphores)

(Κεφάλαιο 15 - Stevens & Rago)

Δημήτρης Ζεϊναλιπούρ

Περιεχόμενα Διάλεξης



- ***Ορισμοί*** *(Προσδιοριστές και Κλειδιά, Διαχείριση IPC πόρων,* ***Διαχείριση*** *IPC με εντολές κελύφους.*
- ***Ουρές Μηνυμάτων (Ο.Μ.):*** *Δομή Ο.Μ., Δημιουργία, Αποστολή/ Παραλαβή Μηνυμάτων, Πλήρες Παράδειγμα, Έλεγχος Ο.Μ..*
- ***Κοινή Μνήμη:*** *Δομή, Δημιουργία, Προσάρτηση / Απόσπαση, Έλεγχος, Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης*
- ***Σηματοφόροι:*** *Δομή, Δημιουργία, Χειρισμός, Έλεγχος, Παράδειγμα Εξυπηρετητή με σηματοφόρους*

Ορισμοί - Προσδιοριστές και Κλειδιά (*Identifiers* και *Keys*)

- Κάθε IPC* δομή (ουρά μηνυμάτων, κοινή μνήμη και σηματοφόροι) στον πυρήνα (*kernel*) αναφέρεται με ένα μη-αρνητικό αριθμό → προσδιοριστής (*identifier*).
- Για παράδειγμα, για να στείλουμε ή να παραλάβουμε ένα μήνυμα σε/από μια ουρά μηνυμάτων, το μόνο που χρειαζόμαστε είναι ο *προσδιοριστής της ουράς* (εσωτερικό IPC όνομα).
- Συνεργαζόμενες διεργασίες χρειάζονται ένα εξωτερικό όνομα για να μπορούν να χρησιμοποιήσουν το ίδιο IPC αντικείμενο.
  - Επομένως, ένα IPC αντικείμενο συσχετίζεται με ένα *κλειδί (key)* που δρα ως ένα εξωτερικό όνομα.
  - Το κλειδί αυτό μετατρέπεται σε ένα προσδιοριστή από τον πυρήνα.

*** όπου IPC δομή θα εννοούμε σ΄ αυτή τη διάλεξη την ουρά μηνυμάτων, κοινή μνήμη και σηματοφόρους)**

Ορισμοί - Διαχείριση IPC Πόρων (συνέχεια)

- Διαχείριση IPC Πόρων
  - Οι IPC πόροι ζουν εκτός της εμβέλειας μιας διεργασίας.
  - Για παράδειγμα, όταν δημιουργούμε μια ουρά μηνυμάτων, τοποθετούμε κάποια μηνύματα στην ουρά και μετά τερματίζουμε, *η ουρά μηνυμάτων και τα περιεχόμενά της* ***δε*** *διαγράφονται*.
  - Παραμένουν στο σύστημα μέχρι **συγκεκριμένα** να διαγραφούν από μια διεργασία (καλώντας συγκεκριμένη IPC λειτουργία → δείτε αργότερα) ή επανεκκινώντας (reboot) το σύστημα
  - Δέστε αντίθεση με pipes: Με το τέλος μιας διεργασίας το Pipe και το περιεχόμενο του χάνεται.
  - Δέστε αντίθεση με FIFO: Με το τέλος όλων των διεργασιών που αναφέρονται στο αρχείο (FIFO), το περιεχόμενο χάνεται, αλλά παραμένει το *όνομα* του στο σύστημα αρχείων, μέχρι να γίνει *unlink*

Ορισμοί - Διαχείριση IPC Πόρων (συνέχεια)

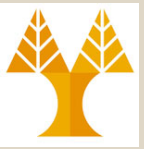
– Οι IPC δομές δεν είναι αναγνωρίσιμες με ονόματα στο σύστημα αρχείων (δε χρησιμοποιούν περιγραφείς αρχείων).

- Δε μπορούμε να έχουμε πρόσβαση σ' αυτές και να αλλάξουμε τις ιδιότητες με κλήσεις συστήματος όπως έχουμε μάθει μέχρι τώρα (open, write, close, …).
- Νέες κλήσεις-λειτουργίες συστήματος έχουν προστεθεί στον πυρήνα για υποστήριξη αυτών των IPC αντικειμένων (→ δείτε αργότερα)

Διαχείριση IPC με εντολές κελύφους

- Ανάγκη για διαχείριση IPC Πόρων
  - Διαγραφή πόρων που έμειναν λόγω διεργασιών που δεν αντιδρούν (*irresponsive ή crashes*)
  - Έλεγχος αριθμού υφιστάμενων πόρων του κάθε τύπου (ειδικά για να βρούμε αν ο καθολικός περιορισμός του συστήματος έχει φθάσει)
  - Δυο utilities συστήματος έχουν δημιουργηθεί για την πιο πάνω διαχείριση: **Δέστε man για λεπτομέρειες**
    - Εντολή **ipcrm** **(όμοιο με *unlink* πάνω σε όνομα αρχείου)**
      - **Δέχεται τον τύπο του πόρου ('msg', 'shm', 'sem') και τον προσδιοριστή ή το κλειδί του πόρου και διαγράφει το συγκεκριμένο πόρο από το σύστημα (→πρέπει να έχουμε τα αναγκαία δικαιώματα πρόσβασης)**

Διαχείριση IPC με εντολές κελύφους (συνέχεια)



- Εντολή **ipcs**
  - **Δείχνει στατιστικές για κάθε τύπο IPC πόρου που υπάρχει στο σύστημα**

```
bash-3.1$ ipcs

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status

------ Semaphore Arrays --------
key        semid      owner      perms      nsems

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

Διαχείριση IPC με εντολές κελύφους (συνέχεια)

- Εντολή **ipcs -l**
  - **Δείχνει τους καθολικά περιορισμούς του συστήματος στους IPC πόρους.**

```
bash-3.1$ ipcs -l

------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 32768
max total shared memory (kbytes) = 8388608
min seg size (bytes) = 1

------ Semaphore Limits --------
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 32
semaphore max value = 32767

------ Messages: Limits --------
max queues system wide = 16
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384
```

Ουρές Μηνυμάτων

- Απαίτηση: #include <sys/msg.h>
- Ουρά Μηνυμάτων ➔Μια συνδεδεμένη λίστα μηνυμάτων αποθηκευμένη στον πυρήνα και η οποία αναγνωρίζεται με ένα προσδιοριστή ουράς μηνυμάτων.
- Χρησιμεύει στην ανταλλαγή μηνυμάτων μεταξύ διεργασιών.
- Συγχρονισμός προσφέρεται *αυτόματα* από τον πυρήνα.
- Νέα μηνύματα προσθέτονται στο τέλος της ουράς.

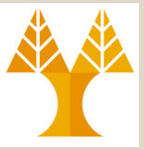
Δομή Μηνύματος

- Ένα μήνυμα αποτελείται από τον τύπο μηνύματος (θετικός *long* ακέραιος) και τα δεδομένα (*data*) του μηνύματος.

```
/* Template for struct to be used as argument for `msgsnd' and
   `msgrcv'.  */
struct msgbuf
{
    long int mtype;         /* type of received/sent message */
    char mtext[1];          /* text of the message */
};
```

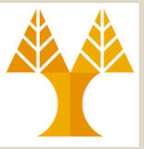
mtext[1] υποδηλώνει το ελάχιστο μέγεθος μηνύματος

Δομή Μηνύματος (συνέχεια)



- Μηνύματα μπορούν να ανακτηθούν από την κεφαλή της ουράς και να προστεθούν στο τέλος της ουράς, είτε ζητώντας συγκεκριμένο *τύπο* μηνύματος.
  - Π.χ.
  - Μια server διεργασία μπορεί να κατευθύνει την κυκλοφορία μηνυμάτων μεταξύ clients στην ουρά μηνυμάτων της, χρησιμοποιώντας το *PID* της client διεργασίας ως τον τύπο μηνυμάτων.
  - Στη συνέχεια, μπορεί να ανακτήσει τα μηνύματα από την ουρά από συγκεκριμένο client.

Δομή Ουράς Μηνυμάτων



- Κάθε ουρά συσχετίζεται με την ακόλουθη *msqid_ds* δομή (ορίζει την υφιστάμενη κατάσταση της ουράς):

```
/* Structure of record for one message inside the kernel. */
struct msqid_ds
{
  struct ipc_perm msg_perm;          /* structure describing operation permission */
  __time_t msg_stime;                /* time of last msgsnd command */
  __time_t msg_rtime;                /* time of last msgrcv command */
  __time_t msg_ctime;                /* time of last change */
  unsigned long int __msg_cbytes;    /* current number of bytes on queue */
  msgqnum_t msg_qnum;                /* number of messages currently on queue */
  msglen_t msg_qbytes;               /* max number of bytes allowed on queue */
  __pid_t msg_lspid;                 /* pid of last msgsnd() */
  __pid_t msg_lrpid;                 /* pid of last msgrcv() */
};
```

Δημιουργία Ουράς Μηνυμάτων

- Κλήση συστήματος *msgget()*

#include <sys/msg.h>

int msgget(key_t key, int flag);

Επιστρέφει: προσδιοριστή ουράς σε επιτυχία ή -1 σε περίπτωση λάθους

  - Ανοίγει μια υφιστάμενη ουρά ή δημιουργεί μια νέα ουρά.
  - Ο προσδιοριστής της ουράς μηνυμάτων που επιστρέφεται αντιστοιχεί στο *κλειδί*
  - Το *key* και *flag* εξηγούνται στις επόμενες διαφάνειες

Δημιουργία Ουράς Μηνυμάτων (συνέχεια)

- Η παράμετρος *key* είναι ένας μοναδικός προσδιοριστής για την ουρά που θέλουμε να δημιουργήσουμε.
  - Οποιαδήποτε άλλη διεργασία θέλει να συνδεθεί με αυτή την ουρά πρέπει να χρησιμοποιήσει το ίδιο κλειδί.
- Το *κλειδί* μπορεί να δοθεί από το χρήστη σαν ένα θετικό ακέραιο που έχει συμφωνηθεί μεταξύ των οντοτήτων που θα χρησιμοποιήσουν την κοινή ουρά μηνυμάτων.
  - Π.χ. ορισμός του κλειδιού σε μια κοινή επικεφαλίδα (*header*)
  - Πρόβλημα όταν το κλειδί αυτό έχει ήδη συσχετιστεί με μια άλλη ουρά μηνυμάτων.
- Αν θέλουμε ο πυρήνας να δημιουργήσει το *κλειδί* υπάρχει και η κλήση συστήματος *ftoc()*

Δημιουργία Ουράς Μηνυμάτων (συνέχεια)

- Το κλειδί μπορεί επίσης να είναι *IPC_PRIVATE (→ sys/ipc.h)*.
  - Με αυτό τον τρόπο, το κλειδί *IPC_PRIVATE* εγγυάται τη δημιουργία μιας νέας IPC δομής ουράς.
  - Το κλειδί *IPC_PRIVATE* επίσης χρησιμοποιείται σε μια σχέση πατέρα-παιδιού.
    - Ο πατέρας δημιουργεί μια νέα ουρά ορίζοντας το κλειδί ως *IPC_PRIVATE,* και ο προσδιοριστής που επιστρέφεται είναι τότε διαθέσιμος στο παιδί μετά το *fork().*

Δημιουργία Ουράς Μηνυμάτων (συνέχεια)

- Η παράμετρος *flag* είναι ένας ακέραιος όπου τίθενται τα επιθυμητά δικαιώματα προστασίας (read/write) της ουράς μηνυμάτων (σε *octal* τιμή), καθώς επίσης και πρόσθετες απαιτήσεις (υπό τη μορφή διάζευξης συμβολικών ονομάτων → *sys/ipc.h*) σχετικές με τη δημιουργία της ουράς μηνυμάτων, όπως:
  - *IPC_CREAT*
    - Αν δεν υπάρχει πόρος (ουρά μηνυμάτων) που αντιστοιχεί στο *κλειδί,* να δημιουργηθεί νέος (αντί να επιστραφεί λάθος)¨ ενώ αν υπάρχει τέτοιος πόρος, να προσπελασθεί αυτός.
  - *IPC_EXCL*
    - Σε συνδυασμό με το προηγούμενο, αν δεν υπάρχει πόρος, να δημιουργηθεί. Αν υπάρχει όμως, να επιστραφεί λάθος.

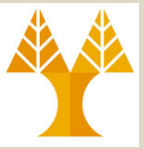
Δημιουργία Ουράς Μηνυμάτων (συνέχεια)

- Παραδείγματα:

```
#include <stdio.h>      /* standard I/O routines. */
#include <sys/types.h> /* standard system data types. */
#include <sys/ipc.h>   /* common system V IPC structures. */
#include <sys/msg.h>   /* message-queue specific functions. */

/* create a private message queue, with access only to the
   owner. */
int queue_id = msgget(IPC_PRIVATE, 0600); /* <-- this is an
                                             octal number. */

if (queue_id == -1) { perror("msgget"); exit(1); }
```

```
key_t key = ftok("/home/cchrys/somefile", 'b');
int queue_id = msgget(key, 0666 | IPC_CREAT);
```

Αποστολή μηνυμάτων



- Κλήση συστήματος msgsnd() → writing to a queue

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *ptr, size_t nbytes, int flag);
```

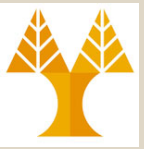
Επιστρέφει -1 σε περίπτωση λάθους ή 0 σε περίπτωση επιτυχίας

  - *msgid*
    - Ο προσδιοριστής της ουράς μηνυμάτων που θέλουμε να γράψουμε (στείλουμε) ένα μήνυμα (επιστρέφεται από τη κλήση συστήματος *msgget()*)
  - *ptr*
    - Ένας δείκτης στη δομή μηνύματος που θέλουμε να γράψουμε στην ουρά
  - *nbytes*
    - Μέγεθος σε bytes των δεδομένων του μηνύματος που θα προσθέσουμε στην ουρά

Αποστολή μηνυμάτων (συνέχεια)

- *flag*
  - Η τιμή 0 μπορεί να τεθεί αν δεν έχουμε κάποιο flag.
  - Το flag *IPC_NOWAIT (→ sys/ipc.h)* μπορεί να ορισθεί.
    - Εάν η ουρά μηνυμάτων είναι γεμάτη (είτε ο συνολικός αριθμός μηνυμάτων στην ουρά έχει φθάσει το επιτρεπτό όριο του συστήματος, είτε ο συνολικός αριθμός bytes στην ουρά έχει φθάσει το επιτρεπτό όριο), με το να ορίσουμε το *flag IPC_NOWAIT* προκαλεί τη *msgsnd()* να επιστρέψει άμεσα με το λάθος *EAGAIN.*
    - Εάν το *flag IPC_NOWAIT* δεν ορισθεί, τότε είμαστε μπλοκαρισμένοι μέχρι
      - » να βρεθεί διαθέσιμος χώρος για το μήνυμα
      - » η ουρά να διαγραφεί από το σύστημα (το λάθος *EIDRM* επιστρέφει)
      - » ή ένα σήμα πιάνεται και το *signal handler* επιστρέφει (το λάθος *EINTR – interrupt* επιστρέφει)

Αποστολή μηνυμάτων (συνέχεια)



- Παράδειγμα:

```
/* first, define the message string */
char* msg_text = "hello world";
/* allocate a message with enough space for length of string and
   one extra byte for the terminating null character. */
struct msgbuf* msg =
   (struct msgbuf*)malloc(sizeof(struct msgbuf) +
   strlen(msg_text));
/* set the message type. for example - set it to '1'. */
msg->mtype = 1;
/* finally, place the "hello world" string inside the message. */
strcpy(msg->mtext, msg_text);

/*we use a message size one larger than the length of the string,
   since we're also sending the null character */
int rc = msgsnd(queue_id, msg, strlen(msg_text)+1, 0);
if (rc == -1) { perror("msgsnd"); exit(1); }
```

Παραλαβή μηνυμάτων

- Κλήση συστήματος msgrcv() → reading from a queue

#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag);

Επιστρέφει -1 σε περίπτωση λάθους ή το μέγεθος των δεδομένων του μηνύματος σε περίπτωση επιτυχίας

  - *msgid*
    - Ο προσδιοριστής της ουράς μηνυμάτων που θέλουμε να διαβάσουμε (παραλάβουμε) ένα μήνυμα (επιστρέφεται από τη κλήση συστήματος *msgget()*)
  - *ptr*
    - Ένας δείκτης στη δομή μηνύματος που θέλουμε να φυλάξουμε τον τύπο και τα δεδομένα του μηνύματος που έχουμε διαβάσει με επιτυχία από την ουρά
  - *nbytes*
    - Μέγεθος σε bytes των δεδομένων στη δομή μηνύματος.

Παραλαβή μηνυμάτων (συνέχεια)

- Εάν το μήνυμα που παραλαμβάνεται/διαβάζεται είναι μεγαλύτερο από την παράμετρο *nbytes* και το *MSG_NOERROR (→ sys/msg.h) flag* ορίζεται, τότε το μήνυμα κόβεται (δε δίνεται καμία ειδοποίηση και το υπόλοιπο του μηνύματος χάνεται).
- Εάν το μήνυμα είναι πολύ μεγάλο και το *MSG_NOERROR flag* δεν ορίζεται, τότε το λάθος *E2BIG* επιστρέφεται.

Παραλαβή μηνυμάτων (συνέχεια)

- Η παράμετρος *type* ορίζει ποιο μήνυμα θέλουμε:
  - *type == 0*
    - Επιστρέφεται το πρώτο μήνυμα στην ουρά
  - *type > 0*
    - Επιστρέφεται το πρώτο μήνυμα στην ουρά του οποίου ο τύπος είναι ίσο με την παράμετρο
  - *type < 0*
    - Επιστρέφεται το πρώτο μήνυμα στην ουρά του οποίου ο τύπος είναι η χαμηλότερη-τιμή μικρότερη ή ίση με την απόλυτη τιμή της παραμέτρου
  - Για παράδειγμα:
    - ο *τύπος* μπορεί να είναι μια τιμή προτεραιότητας, αν η εφαρμογή αναθέτει προτεραιότητες στα μηνύματα.
    - ο *τύπος* μπορεί να περιέχει την ταυτότητα της διεργασίας ενός client, εάν μια ουρά μηνυμάτων χρησιμοποιείται από πολλαπλούς clients και ένα server.

Πλήρες παράδειγμα ουράς μηνυμάτων

- Το *queue_sender.c* δημιουργεί μια ουρά μηνυμάτων και στέλλει μηνύματα με διαφορετικές προτεραιότητες στην ουρά.
- Το *queue_reader.c* διαβάζει τα μηνύματα από την ουρά και τα τυπώνει.
  - Παίρνει μια τιμή στη γραμμή εντολής που αντιπροσωπεύει την προτεραιότητα των μηνυμάτων που θέλουμε να διαβάσουμε.
  - Τρέχουμε πολλαπλά αντίγραφα του προγράμματος αυτού ταυτόχρονα.
- Τέτοιος μηχανισμός μπορεί να χρησιμοποιηθεί από ένα σύστημα στο οποίο πολλοί clients μπορούν να στέλλουν αιτήματα διαφόρων τύπων, τα οποία χρειάζονται διαφορετική αντιμετώπιση.

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)

```
/*
 * queue_sender.c - a program that writes
    messages with one of 3 identifiers to a
    message queue.
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>

#include "queue_defs.h"

int main(int argc, char* argv[])
{
    int queue_id;     /* ID of the created
                                        queue.*/
    struct msgbuf* msg;  /* structure used for
                            sent messages.   */
    int i;                  /* loop counter */
    int rc;                 /* error code
                  returned by system calls. */

    /* create a public message queue, with
    access only to the owning user. */
    queue_id = msgget(QUEUE_ID, IPC_CREAT |
                              IPC_EXCL | 0600);
    if (queue_id == -1) {
          perror("main: msgget");
          exit(1);
    }
```

```
printf("message queue created, queue id
    '%d'.\n", queue_id);
    msg = (struct msgbuf*)malloc(sizeof(struct
    msgbuf)+MAX_MSG_SIZE);

    /* form a loop of creating messages and
    sending them. */
    for (i=1; i <= NUM_MESSAGES; i++) {
        msg->mtype = (i % 3) + 1; /* create
    message type between '1' and '3' */
        sprintf(msg->mtext, "hello world -
                                    %d", i);
        rc = msgsnd(queue_id, msg,
                              strlen(msg-
    >mtext)+1, 0);
        if (rc == -1) {
        perror("main: msgsnd");
        exit(1);
        }
    }
    /* free allocated memory. */
    free(msg);

    printf("generated %d messages,
          exiting.\n", NUM_MESSAGES);

    return 0;
}
```

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)

```
/*
 * queue_reader.c - a program that reads
    messages with a given identifier
 *                  off of a message queue.
 */

#include <stdio.h>        /* standard I/O
                             functions. */
#include <stdlib.h>       /* malloc(), free()
                                    etc. */
#include <unistd.h>       /* sleep(), etc. */
#include <sys/types.h>    /* various type
                             definitions. */
#include <sys/ipc.h>      /* general SysV IPC
                             structures */
#include <sys/msg.h>      /* message queue
                          functions and structs. */

#include "queue_defs.h"   /* definitions shared
                             by both programs  */

int main(int argc, char* argv[])
{
    int queue_id;              /* ID of the
                               created queue.  */
    struct msgbuf* msg;        /* structure used
                         for received messages. */
    int rc;                    /* error code
                  returned by system calls.  */
    int msg_type;              /* type of
                messages we want to receive.  */
```

```
/* read message type from command line */
if (argc != 2) {
    fprintf(stderr, "Usage: %s <message
                type>\n", argv[0]);
    fprintf(stderr, " <message type> must
                be between 1 and 3.\n");
exit(1);
}
msg_type = atoi(argv[1]);
if (msg_type < 1 || msg_type > 3) {
    fprintf(stderr, "Usage: %s <message
                type>\n", argv[0]);
    fprintf(stderr, " <message type> must
                be between 1 and 3.\n");
exit(1);
}
```

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)

```
    /* access the public message queue that the sender program created. */
    queue_id = msgget(QUEUE_ID, 0);
    if (queue_id == -1) {
          perror("main: msgget");
          exit(1);
    }
    printf("message queue opened, queue id '%d'.\n", queue_id);
    msg = (struct msgbuf*)malloc(sizeof(struct msgbuf)+MAX_MSG_SIZE);

    /* form a loop of receiving messages and printing them out. */
    while (1) {
        rc = msgrcv(queue_id, msg, MAX_MSG_SIZE+1, msg_type, IPC_NOWAIT);
        if (rc == -1) {
          perror("main: msgrcv");
          exit(1);
        }
        printf("Reader '%d' read message: '%s'\n", msg_type, msg->mtext);
        /* slow down a little... */
        sleep(1);
    }
    /* NOT REACHED */
    return 0;
}
```

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)

```
#ifndef QUEUE_DEFS_H
# define QUEUE_DEFS_H

/*
 * queue_defs.h - common macros and definitions for the public message
 *                queue example.
 */

#define QUEUE_ID 137       /* ID of queue to generate. */
#define MAX_MSG_SIZE 200   /* size (in bytes) of largest message we'll send.*/
#define NUM_MESSAGES 100   /* number of messages the sender program will send. */
struct msgbuf
{
  long int mtype;              /* type of received/sent message */
  char mtext[1];               /* text of the message */
};

#endif /* QUEUE_DEFS_H */
```

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)

```
bash-3.1$ gcc queue_sender.c -o sender
bash-3.1$ gcc queue_reader.c -o reader
```

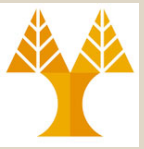
Terminal 1

```
bash-3.1$ ./sender
bash-3.1$ ./reader 1
message queue opened, queue id '98304'.
Reader '1' read message: 'hello world - 3'
Reader '1' read message: 'hello world - 6'
Reader '1' read message: 'hello world - 9'
Reader '1' read message: 'hello world - 12'
Reader '1' read message: 'hello world - 15'
Reader '1' read message: 'hello world - 18'
Reader '1' read message: 'hello world - 21'
Reader '1' read message: 'hello world - 24'
Reader '1' read message: 'hello world - 27'
Reader '1' read message: 'hello world - 30'
Reader '1' read message: 'hello world - 33'
Reader '1' read message: 'hello world - 36'
Reader '1' read message: 'hello world - 39'
Reader '1' read message: 'hello world - 42'
Reader '1' read message: 'hello world - 45'
Reader '1' read message: 'hello world - 51'
Reader '1' read message: 'hello world - 57'
Reader '1' read message: 'hello world - 63'
Reader '1' read message: 'hello world - 69'
Reader '1' read message: 'hello world - 75'
Reader '1' read message: 'hello world - 81'
Reader '1' read message: 'hello world - 87'
Reader '1' read message: 'hello world - 93'
Reader '1' read message: 'hello world - 99'
main: msgrcv: No message of desired type

bash-3.1$
```

Terminal 2

```
bash-3.1$ ./reader 1
message queue opened, queue id '98304'.
Reader '1' read message: 'hello world - 48'
Reader '1' read message: 'hello world - 54'
Reader '1' read message: 'hello world - 60'
Reader '1' read message: 'hello world - 66'
Reader '1' read message: 'hello world - 72'
Reader '1' read message: 'hello world - 78'
Reader '1' read message: 'hello world - 84'
Reader '1' read message: 'hello world - 90'
Reader '1' read message: 'hello world - 96'
main: msgrcv: No message of desired type
bash-3.1$
```

Πλήρες παράδειγμα ουράς μηνυμάτων (συνέχεια)



```
bash-3.1$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000089 98304      cspgcc1    600        1134         67
```

```
bash-3.1$ ipcrm -q 98304
```

Έλεγχος Ουράς Μηνυμάτων

- Κλήση συστήματος msgctl()

```
#include <msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Επιστρέφει -1 σε περίπτωση λάθους ή 0 σε περίπτωση επιτυχίας

  - Η *msgctl()* αλλάζει τα δικαιώματα πρόσβασης και άλλα χαρακτηριστικά μιας ουράς μηνυμάτων.
  - Εκτελεί την ενέργεια (εντολή) που ορίζεται στην παράμετρο *cmd* επάνω στην ουρά μηνυμάτων που αντιστοιχεί στον προσδιοριστή που ορίζεται στην παράμετρο *msqid*.

Έλεγχος Ουράς Μηνυμάτων (συνέχεια)

- Η παράμετρος *cmd* μπορεί να είναι:
  - *IPC_STAT*
    - Φέρε τη *msqid_ds* δομή της συγκεκριμένης ουράς, αποθηκεύοντάς την στη δομή που δείχνεται από τη παράμετρο *buf.*
      - δλδ, φύλαξη των πληροφοριών σχετικά με την κατάσταση της ουράς στη δομή δεδομένων που δείχνεται από τη *buf.*
  - *IPC_SET*
    - Ρύθμιση των δικαιωμάτων πρόσβασης, ιδιοκτήτη, ομάδα, και του επιτρεπτού μέγιστου αριθμού bytes της *msqid_ds* δομής της συγκεκριμένης ουράς από τη *buf*
      - *αντιγραφή των πιο πάνω πεδίων από τη δομή που δείχνεται από τη buf στη msqid_ds* δομή που συσχετίζεται με τη συγκεκριμένη ουρά

Έλεγχος Ουράς Μηνυμάτων (συνέχεια)

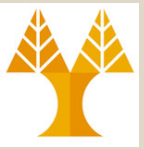
- *IPC_RMID*
  - Διαγράφει (καταστρέφει) την ουρά μηνυμάτων από το σύστημα μαζί με όλα τα δεδομένα που βρίσκονται σ'αυτή.
  - Η αφαίρεση είναι άμεση.
  - Οποιαδήποτε άλλη διεργασία ακόμα χρησιμοποιεί την ουρά μηνυμάτων, θα πάρει ένα μήνυμα (*EIDRM → "identifier removed"*) στην επόμενη προσπάθεια λειτουργίας στην ουρά.
  - Η παράμετρος *buff* τίθεται 0 (null)

Έλεγχος Ουράς Μηνυμάτων (συνέχεια)

- Παραδείγματα:

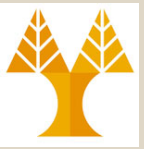
```
#include<sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
...
if (msgctl(msqid, IPC_STAT, &buf) == -1) {
   perror("msgctl: msgctl failed"); exit(1);
}
...
if (msgctl(msqid, IPC_SET, &buf) == -1) {
perror("msgctl: msgctl failed"); exit(1);
}
...

 if (msgctl(msqid, IPC_RMID, NULL) == -1) {
perror("msgctl: msgctl failed"); exit(1);
}
```

Κοινή Μνήμη



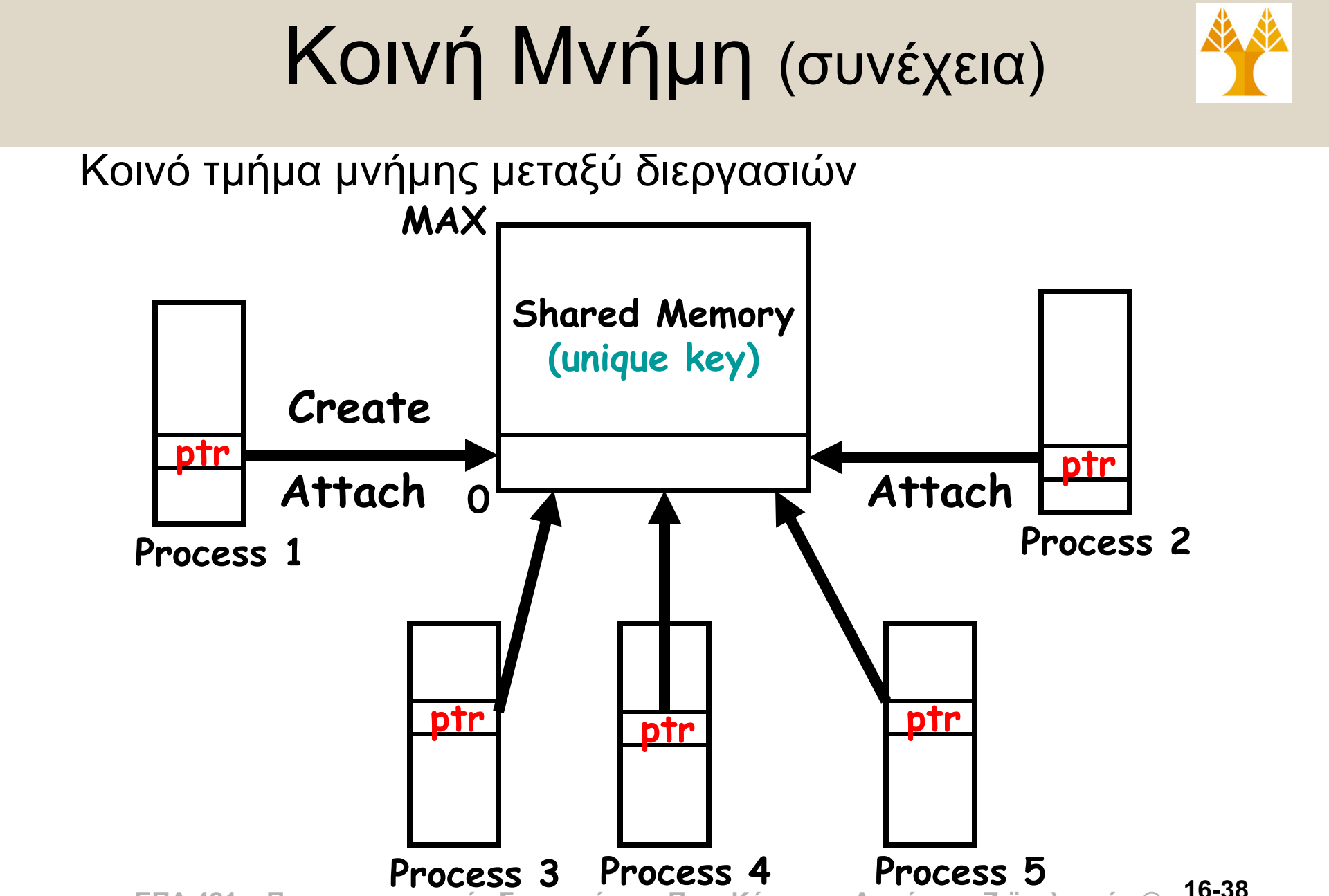
- Απαίτηση: #include <sys/shm.h>
- Δυνατότητα επικοινωνίας μεταξύ διεργασιών μέσω καταχώρησης και ανάγνωσης πληροφοριών σε περιοχή μνήμης που είναι προσπελάσιμη από όλες τις διεργασίες που έχουν δικαίωμα.
- Ανάγκη συγχρονισμού των διεργασιών, συνήθως μέσω σηματοφόρων.

Κοινή Μνήμη (συνέχεια)



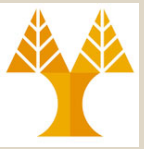
- Σ’ αντίθεση με την ουρά μηνυμάτων, που αντιγράφει δεδομένα από τη διεργασία στη μνήμη μέσα στον πυρήνα, η κοινή μνήμη είναι απευθείας προσβάσιμη.
- Η κοινή μνήμη βρίσκεται στη μνήμη διεργασίας του χρήστη και τότε μοιράζεται μεταξύ άλλων διεργασιών.

Κοινή Μνήμη (συνέχεια)



Κοινό τμήμα μνήμης μεταξύ διεργασιών

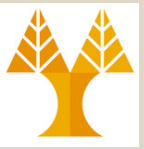
Δομή Κοινής Μνήμης



- Ο πυρήνας διατηρεί μια δομή με τα ακόλουθα μέλη για κάθε τμήμα κοινής μνήμης που δημιουργείται:

```
struct shmid_ds
  {
    struct ipc_perm shm_perm;        /* operation permission struct */
    size_t shm_segsz;                /* size of segment in bytes */
   _time_t shm_atime;                /* time of last shmat() */
   _time_t shm_dtime;                /* time of last shmdt() */
  _ time_t shm_ctime;                /* time of last change by shmctl() */
__pid_t shm_cpid;                    /* pid of creator */
__pid_t shm_lpid;                    /* pid of last shmop */
    shmatt_t shm_nattch;             /* number of current attaches */
    . . .
  };
```

Δημιουργία Κοινής Μνήμης



- Κλήση συστήματος *shmget()*

#include <sys/shm.h>

int shmget(key_t key, size_t size, int flag);

Επιστρέφει: προσδιοριστή κοινής μνήμης σε επιτυχία ή -1 σε περίπτωση λάθους

  - Δημιουργία τμήματος κοινής μνήμης
  - Επιστρέφει ένα προσδιοριστή για την κοινή μνήμη μεγέθους *size* (σε bytes) που αντιστοιχεί στο κλειδί *key*.
  - Στην παράμετρο *flag* 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.

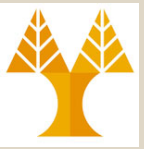
Δημιουργία Κοινής Μνήμης
(συνέχεια)

- Εάν ένα νέο τμήμα κοινής μνήμης δημιουργείται, πρέπει να δηλωθεί ένα *size*.
- Εάν παραπέμπουμε σε υφιστάμενο τμήμα κοινής μνήμης (πελάτης), μπορούμε να ορίσουμε το *size* ως 0.
- Όταν ένα νέο τμήμα κοινής μνήμης δημιουργείται, τα περιεχόμενα του τμήματος αρχικοποιούνται με *0s*.

Παράδειγμα:

```
/* this variable is used to hold the returned segment
   identifier. */
int shm_id;
/* allocate a shared memory segment with size of 2048
   bytes, accessible only to the current user. */
key_t key  = ftok("/home/user/somefile.txt", 'R');
shm_id = shmget(key, 2048, IPC_CREAT | IPC_EXCL | 0600);
if (shm_id == -1) { perror("shmget: "); exit(1); }
```

Προσάρτηση Κοινής Μνήμης



- Κλήση συστήματος *shmat()*

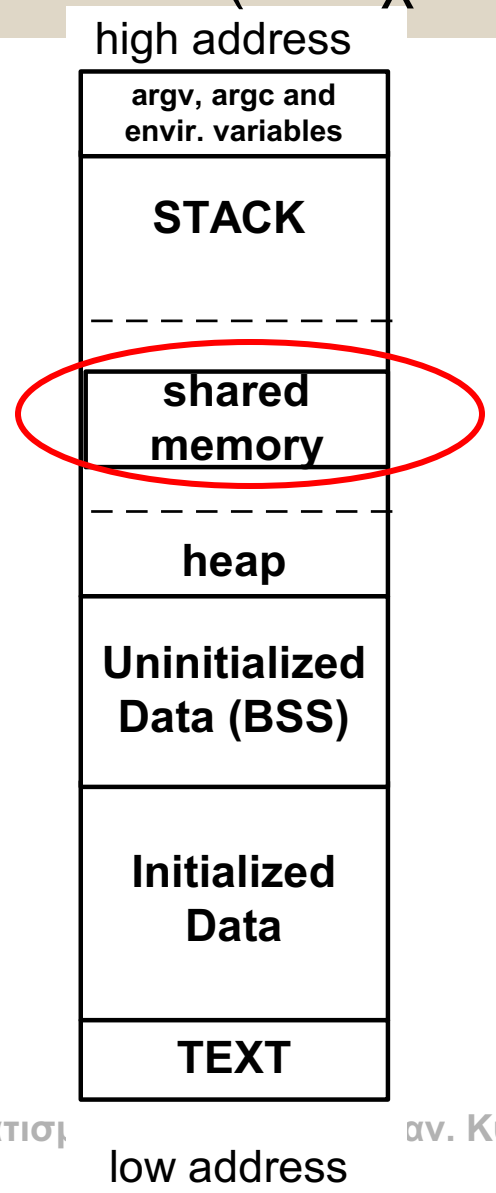
#include <sys/shm.h>

void *shmat(int shmid, const void *addr, int flag);

Επιστρέφει: δείκτη στο τμήμα κοινής μνήμης σε περίπτωση επιτυχίας, αλλιώς -1

- Όταν ένα τμήμα κοινής μνήμης δημιουργείται, μια διεργασία προσαρτά το τμήμα στο χώρο διευθύνσεών της *(address space)*.
- Η *shmat()* 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 παραμέτρων *addr* και *flag* μπορεί να ζητηθεί η προσάρτηση σε συγκεκριμένη περιοχή μνήμης, αλλά είναι πιο σύνηθες και ορθό να αφεθεί η επιλογή αυτή στον πυρήνα θέτοντας 0 και στις δυο παραμέτρους.
- Εάν το *flag* τίθεται ως *SHM_RDONLY*, το τμήμα είναι μόνο για *read*, αλλιώς είναι *read-write*.

Προσάρτηση Κοινής Μνήμης
(συνέχεια)



Memory layout

Απόσπαση Κοινής Μνήμης

- Κλήση συστήματος *shmdt()*

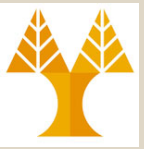
#include <sys/shm.h>

int shmdt(void *addr);

Επιστρέφει: 0 σε περίπτωση επιτυχίας, αλλιώς -1 σε περίπτωση λάθους

- Η παράμετρος *addr* είναι η διεύθυνση που επιστράφηκε από την κλήση συστήματος *shmat()*
- Αποσπά την προσαρτημένη, στη διεύθυνση *addr*, κοινή μνήμη, όταν η συγκεκριμένη διεργασία που την καλεί έχει τελειώσει από το να χρησιμοποιεί την κοινή μνήμη.
- Αυτό δε διαγράφει τον προσδιοριστή και τη σχετική IPC δομή από το σύστημα.
  - Ο προσδιοριστής παραμένει να υφίσταται μέχρι κάποια διεργασία τη διαγράψει συγκεκριμένα.

Έλεγχος Κοινής Μνήμης



- Κλήση συστήματος shmctl()

```
#include <shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

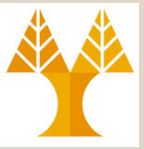
Επιστρέφει -1 σε περίπτωση λάθους ή 0 σε περίπτωση επιτυχίας

  - 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 **buf* με τα χαρακτηριστικά της κοινής μνήμης, ενώ με την ενέργεια ιIPC_SET ρυθμίζονται συγκεκριμένα πεδία από τη *δομή που δείχνεται από τη buf στη shmid_ds* δομή που συσχετίζεται με το συγκεκριμένο τμήμα κοινής μνήμης.
  - Η συνηθέστερη ενέργεια είναι η *IPC_RMID* που καταστρέφει την κοινή μνήμη.

Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης

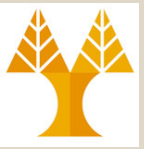
```
/**** shm_server.c ****/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define SHMSZ 27

main() {
   char c; int shmid; key_t key; char *shm, *s;

   /* We'll name our shared memory segment * "5678". */
   key = 5678;

   /* Create the segment. */
   if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
   { perror("shmget"); exit(1); }

/* Now we attach the segment to our data space. */
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
   perror("shmat"); exit(1); }
```

Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης (συνέχεια)



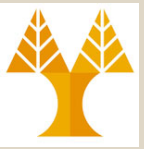
```
/* Now put some things into the memory for the other
   process to read. */
s = shm;
for (c = 'a'; c <= 'z'; c++)
   *s++ = c;
*s = '\0';

/* Finally, we wait until the other process changes the
   first character of our memory to '*', indicating that it
   has read what we put there. */
while (*shm != '*') sleep(1);
/* detach from the segment: */
if (shmdt(shm) == -1) perror("shmdt");
/* delete the segment */
if( shmctl(shmid, IPC_RMID, NULL) == -1) {
   perror("shmctl"); exit(1); }
return 0; }
```

Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης (συνέχεια)



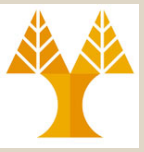
```
/**** shm_client.c ****/
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define SHMSZ 27

main() {
   int shmid; key_t key; char *shm, *s;

/* We need to get the segment named "5678", created by the
   server. */
   key = 5678;

/* Locate the segment. */
   if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
   { perror("shmget"); exit(1); }
```

Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης (συνέχεια)



```
/* Now we attach the segment to our data space. */
if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
   perror("shmat"); exit(1); }

/* Now read what the server put in the memory. */
for (s = shm; *s != '\0'; s++)
   putchar(*s);
putchar('\n');

/* Finally, change the first character of the segment to
   '*', indicating we have read the segment. */
*shm = '*';

/* detach from the segment: */
if (shmdt(shm) == -1) perror("shmdt");

return(0); }
```

Παράδειγμα δυο διεργασιών που επικοινωνούν μέσω κοινής μνήμης (συνέχεια)



```
bash-3.1$ gcc shm_server.c -o shm_server
bash-3.1$ gcc shm_client.c -o shm_client
```

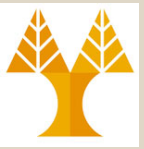
Terminal 1

```
bash-3.1$ ./shm_server

bash-3.1$
```

Terminal 2

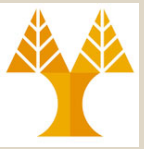
```
bash-3.1$ ./shm_client
abcdefghijklmnopqrstuvwxyz
bash-3.1$
```

Σηματοφόροι



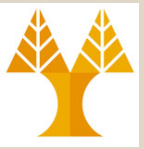
- **Ένας σηματοφόρος είναι ένας μετρητής που προσφέρει πρόσβαση σε ένα κοινό αντικείμενο δεδομένων για πολλαπλές διεργασίες.**
- Μηχανισμός συγχρονισμού διεργασιών για την αποκλειστική διαχείριση κοινών πόρων (π.χ. κοινής μνήμης).
- Πριν την είσοδο σε κρίσιμο τμήμα του προγράμματός της, μια διεργασία ζητά την απαιτούμενη άδεια από ένα 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 της τιμής του σηματοφόρου και η αποδέσμευση με κατάλληλη αύξηση της τιμής του σηματοφόρου.

Σηματοφόροι (συνέχεια)



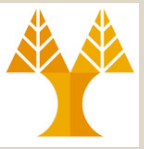
- Δέσμευση κοινού πόρου από μια διεργασία:
  1. Έλεγχος σηματοφόρου που ελέγχει τον πόρο.
  2. Αν η τιμή του σηματοφόρου είναι θετική, η διεργασία μπορεί να χρησιμοποιήσει τον κοινό πόρο. Τότε η διεργασία μειώνει την τιμή του σηματοφόρου κατά 1, δηλώνοντας ότι έχει χρησιμοποιήσει μια μονάδα του πόρου.
  3. Αλλιώς, αν η τιμή του σηματοφόρου είναι 0, η διεργασία αναμένει (*sleep*) μέχρι η τιμή του σηματοφόρου είναι μεγαλύτερη από 0. Όταν η διεργασία *ξυπνήσει* επιστρέφει στο βήμα 1.
- Αποδέσμευση κοινού πόρου από μια διεργασία:
  1. Όταν μια διεργασία τελειώσει με τον κοινό πόρο, τότε η τιμή του σηματοφόρου αυξάνεται κατά 1.
  2. Αν άλλες διεργασίες αναμένουν (*κοιμούνται*), τότε *ξυπνούν*.
- Ο έλεγχος του σηματοφόρου και οι όποιες λειτουργίες πρέπει να είναι ατομική διαδικασία.
- Γι' αυτό, οι σηματοφόροι υλοποιούνται μέσα στον πυρήνα.

Σηματοφόροι (συνέχεια)



- Απαίτηση: #include <sys/sem.h>
- Μπορούμε να ορίσουμε ένα σηματοφόρο ως ένα σύνολο μιας ή περισσοτέρων τιμών σηματοφόρων.
- Όταν δημιουργούμε ένα σηματοφόρο, πρέπει να **ορίσουμε τον αριθμό των τιμών μέσα στο σύνολο.**

Δομή Συνόλου Σηματοφόρων



- Ο πυρήνας διατηρεί μια δομή με τα ακόλουθα μέλη για κάθε σύνολο σηματοφόρων που δημιουργείται:

```
/* Data structure describing a set of semaphores.  */
struct semid_ds
{
  struct ipc_perm sem_perm;      /* operation permission struct */
  __time_t sem_otime;            /* last semop() time */
 __time_t sem_ctime;             /* last time changed by semctl() */
 unsigned long int sem_nsems;    /* number of semaphores in set */
 . . .
};
```

Δημιουργία Συνόλου Σηματοφόρων

- Κλήση συστήματος *semget()*

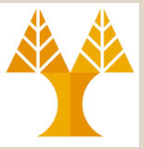
#include <sys/sem.h>

int semget(key_t key, int nsems, int flag);

Επιστρέφει: προσδιοριστή σηματοφόρου σε επιτυχία ή -1 σε περίπτωση λάθους

  - Δημιουργία συνόλου σηματοφόρων
  - Επιστρέφει ένα προσδιοριστή για ένα σύνολο από *nsems* σηματοφόρους που αντιστοιχεί στο κλειδί *key*.
  - Στην παράμετρο *flag* 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.
  - Εάν καλούμε υφιστάμενο σύνολο σημαφόρων, η παράμετρος *nsems* τίθεται 0.

Δημιουργία Συνόλου Σηματοφόρων
(συνέχεια)



Παράδειγμα:

```
/* ID of the semaphore set. */
int sem_set_id_1; int sem_set_id_2;

/* create a private semaphore set with one semaphore in
   it, with access only to the owner. */
sem_set_id_1 = semget(IPC_PRIVATE, 1, IPC_CREAT | 0600);
if (sem_set_id_1 == -1) {perror("main: semget"); exit(1);}

/* create a semaphore set with key 250, three semaphores
   in the set, with access only to the owner. */
sem_set_id_2 = semget(250, 3, IPC_CREAT | 0600);
if (sem_set_id_2 == -1) {perror("main: semget"); exit(1);}
```

Χειρισμός Συνόλου Σηματοφόρων

- Κλήση συστήματος *semop()*

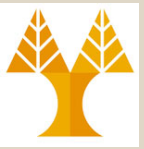
#include <sys/sem.h>

int semop(int semid, struct sembuf semoparray[], size_t nops);

Επιστρέφει 0 σε περίπτωση επιτυχίας ή -1 σε περίπτωση λάθους

  - Εκτελεί στο σύνολο σηματοφόρων, που προσδιορίζονται από την παράμετρο *semid*, τις λειτουργίες που καθορίζονται σε ένα πίνακα μεγέθους *nops* από δομές *struct sembuf*, το πρώτο στοιχείο του οποίου δείχνει η παράμετρος *semoparray.*
  - Η παράμετρος *nops* ορίζει τον αριθμό των λειτουργιών (στοιχεία του πίνακα).

Χειρισμός Συνόλου Σηματοφόρων
(συνέχεια)



– Η παράμετρος *semoparray* είναι ένας πίνακας από λειτουργίες σηματοφόρων, που αντιπροσωπεύεται από δομές *sembuf*:

```
/* Structure used for argument to `semop' to describe
   operations.  */
struct sembuf
{
  unsigned short int sem_num; /* semaphore number */
  short int sem_op;           /* semaphore operation */
  short int sem_flg;          /* operation flag */
};
```

– Περιγράφει τη λειτουργία μεταβολής της τιμής του υπ' αριθμό *sem_num* σηματοφόρου του συνόλου (από *0* έως *nsems -1*) κατά *sem_op* (<0 για δέσμευση και >0 για αποδέσμευση).

Χειρισμός Συνόλου Σηματοφόρων
(συνέχεια)

- *SEM_UNDO flag*
  - Δημιουργείται πρόβλημα όταν μια διεργασία τερματίζεται, ενώ έχει πόρους δεσμευμένους διαμέσου ενός σηματοφόρου.
  - Όταν ορίζουμε τη *SEM_UNDO* flag στην κλήση συστήματος *semop()* για μια λειτουργία σηματοφόρου και δεσμεύουμε πόρους (μια *sem_op* τιμή <0), ο πυρήνας θυμάται πόσους πόρους έχουμε δεσμεύσει από το συγκεκριμένο σηματοφόρο (η απόλυτη τιμή του *sem_op*).
  - Όταν η διεργασία τερματίσει, ο πυρήνας ελέγχει κατά πόσο η διεργασία έχει οποιεσδήποτε μη-ολοκληρωμένες ρυθμίσεις της τιμής του σηματοφόρου και εφαρμόζει αυτές τις ρυθμίσεις.

Έλεγχος Σηματοφόρου

- Κλήση συστήματος semctl()

#include <sem.h>

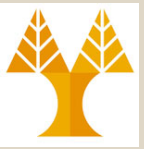
int semctl(int semid, int semnum, int cmd, union semun arg);

Επιστρέφει τιμή ανάλογα με την εντολή

  - Εκτελεί την ενέργεια *cmd* στον *semnum* σηματοφόρο (από *0* έως *nsems -1*) του συνόλου σημαφόρων (ή ανάλογα με την εντολή, σε ολόκληρο το σύνολο) που αντιστοιχεί στον προσδιοριστή *semid*
  - Η ένωση *union semun* είναι ορισμένη ως

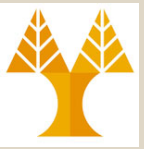
```
/* The user should define a union like the following to use it for
   arguments for `semctl'. */

   union semun
   {
     int val;                    /* value for SETVAL */
     struct semid_ds *buf;       /* buffer for IPC_STAT & IPC_SET */
     unsigned short int *array;        /* array for GETALL & SETALL */
     struct seminfo *__buf;            /* buffer for IPC_INFO */
   };
```

Έλεγχος Σηματοφόρου
(συνέχεια)



- Με την εντολή *IPC_STAT* συμπληρώνονται τα πεδία της δομής **(arg.buf)* με τα χαρακτηριστικά του συνόλου σηματοφόρων.
- Με τις εντολές *SETVAL* και *GETVAL* τίθεται σαν τιμή ενός σηματοφόρου το *arg.val* ή επιστρέφει η κλήση συνάρτησης *semctl()* την τιμή του, αντίστοιχα.
- Με τις εντολές *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*), αντίστοιχα.
- Με την εντολή *IPC_RMID* καταστρέφεται το σύνολο των σηματοφόρων.

Παράδειγμα

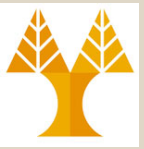


Παράδειγμα επικοινωνίας δυο διεργασιών μεταξύ τους μέσω κοινής μνήμης και το συγχρονισμό τους μέσω σηματοφόρων

/* File: **shm_sem_server.c** */

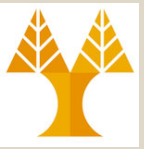
```
#include <sys/types.h>                  /* For System V IPC */
#include <sys/ipc.h>                    /* For System V IPC */
#include <sys/shm.h>                     /* For shared memory */
#include <sys/sem.h>                    /* For semaphores */
#include <stdio.h>                      /* For I/O */
#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; };
```

Παράδειγμα (συνέχεια)



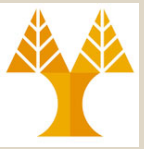
```
main()
{ int shmid, semid; char line[128], *shmem;
struct sembuf oper[1] = {0, 1, 0};    /*want to release shared memory region*/
union semun arg;

/* Create shared memory */
if ((shmid = shmget(SHMKEY, SHMSIZ, PERMS | IPC_CREAT)) < 0) {
perror("shmget"); exit(1); }
printf("Created shared memory region with identifier %d\n", shmid);

/* Create semaphore set with 1 item */
if ((semid = semget(SEMKEY, 1, PERMS | IPC_CREAT)) < 0) {
perror("semget"); exit(1); }
printf("Created semaphore with identifier %d\n", semid);

/* Initialize semaphore for locking */
arg.val=0;
if (semctl(semid, 0, SETVAL, arg) < 0) {
perror("semctl"); exit(1); }
printf("Initialized semaphore to lock shared memory region\n");
```

Παράδειγμα (συνέχεια)



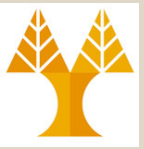
```
/* Attach shared memory region locally */
if ((shmem = shmat(shmid, (char *) 0, 0)) == (char *) -1) {
perror("shmat"); exit(1); }
printf("Attached shared memory region\n");

/* Write message to shared memory */
printf("Give input line: ");
fgets(line, sizeof line, stdin);
line[strlen(line)-1] = '\0';
strcpy(shmem, line);
printf("Wrote to shared memory region: %s\n", line);

/* Make shared memory available */
if (semop(semid, &oper[0], 1) < 0) {
perror("semop"); exit(1); }

printf("Released shared memory region\n");
}
```

Παράδειγμα (συνέχεια)



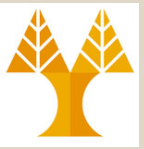
```
/* File: shm_sem_client.c */

#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 */

#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};     /*want to reserve shared memory region*/
```

Παράδειγμα (συνέχεια)



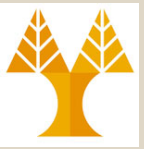
```
/* Access shared memory */
if ((shmid = shmget(SHMKEY, SHMSIZ, PERMS)) < 0) {
perror("shmget"); exit(1); }
printf("Accessing shared memory region with identifier %d\n", shmid);

/* Access semaphore set */
if ((semid = semget(SEMKEY, 1, PERMS)) < 0) {
perror("semget"); exit(1); }
printf("Accessing semaphore with identifier %d\n", semid);

/* Attach shared memory region locally */
if ((shmem = shmat(shmid, (char *) 0, 0)) == (char *) -1) {
perror("shmat"); exit(1); }
printf("Attached shared memory region\n");
```

Παράδειγμα (συνέχεια)



```
printf("Asking for access to shared memory region\n");

/* Ask if you may access shared memory */
if (semop(semid, &oper[0], 1) < 0) {
perror("semop"); exit(1); }
printf("Read from shared memory region: %s\n", shmem); /* Accessing */

/* Destroy shared memory */
if (shmctl(shmid, IPC_RMID, (struct shmid_ds *) 0) < 0) {
perror("shmctl"); exit(1); }
printf("Removed shared memory region with identifier %d\n", shmid);

/* Destroy semaphore set */
if (semctl(semid, 0, IPC_RMID, 0) < 0) {
perror("semctl"); exit(1); }
printf("Removed semaphore with identifier %d\n", semid);
 }
```

Παράδειγμα (συνέχεια)



```
bash-3.1$ gcc shm_sem_server.c -o shm_sem_server
bash-3.1$ gcc shm_sem_client.c -o shm_sem_client
```

```
bash-3.1$ ./shm_sem_server
Created shared memory region with identifier 6946821
Created semaphore with identifier 327680
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
bash-3.1$
bash-3.1$ ./shm_sem_client
Accessing shared memory region with identifier 6946821
Accessing semaphore with identifier 327680
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 6946821
Removed semaphore with identifier 327680
bash-3.1$
```