Topic 7: IPC*

K24: Systems Programming

Instructor: Mema Roussopoulou

Interprocess Communication (System V)

- Until now we've learned
  - Sockets
  - Pipes
  - Named pipes (aka FIFO)
  - Signals, etc…
- Pipes are limiting
  - If unnamed
    - Processes must be related
    - Pipe must be created before one of the processes is born
    - File descriptors must be inherited
  - Data typically flows in only one direction
  - Data must be read in order written
    - Cannot prioritize data message types
  - Potential performance hit for large data transfers
- Named pipes
  - Processes don't need to be related
  - But exhibit remaining limitations of regular pipes

Interprocess Communication (System V)

- Three more types of IPC
  - Message Queues
  - Shared Memory
  - Semaphores
  - For IPC on **same machine**
- System V (older) vs POSIX (standard)
  - POSIX is an attempt to make a standard, starting 1993
  - Υποτίθεται το POSIX IPC είναι πιο φορητό από το IPC του System V
  - Στη πράξη συμβαίνει το αντίθετο – System V έχει υλοποιηθεί σχεδόν σε κάθε σύστημα
  - POSIX IPC ακόμα δεν είναι διαθέσιμο σε αρκετά συστήματα UNIX
  - System V IPC είναι υποχρεωτικό τμήμα για τα πιστοποιημένα συστήματα Unix της Open Group

(System V) IPC Characteristics

- Lifetime of IPC objects lasts until kernel reboots.
- Each IPC structure is referred to by a non-negative integer *identifier)*
  - When an IPC is created the program doing the creation provides a *key* of type **key_t**
  - The OS converts this key into an IPC identifier
- There are no inodes or pathnames so traditional file management syscalls (read, write, stat, unlink) unusable

ipcrm and ipcs – εντολές για να διαγράψουμε ένα αντικείμενο ή να δούμε την κατάσταση ενός αντικειμένου

Keys

- Προσπέλαση ενός αντικειμένου IPC του System V γίνεται με αναγνωριστικό
- Μπορείτε επίσης να αναφέρεστε σε ένα αντικείμενο IPC με ένα κλειδί που παραμένει σταθερό (across machine reboots)
- For ease of use, generate a key using a pathname and ftok() – (next page)

Keys (cont'd)

- Keys should be unique
- Keys can be specified in three ways:
  - Server and client can agree on a pathname to an existing file in the file system AND a project ID (1..255) and then call *ftok*() to convert these two values into a key
  - The server creates a new structure by specifying a key of IPC_PRIVATE.
    - Key guaranteed to be unique
    - Client has to become aware of this private key (often through a file generated by server and then looked up by the client).
  - Server and client agree on a key value (often defined in the header)

Access Rights for IPC objects

- Keys provide access to IPC objects which are data structures of type:
  - struct msqid_ds
  - struct shmid_ds
  - struct semid_ds
- If IPC objects were files could use file permissions to manage access
  - They are NOT, so need separate access right management
  - Looks a lot like file access management
  - 9 bits for read, write, execute (execute bits unused)
- Wrongly accessing resources returns -1

Access Rights for IPC objects

- Access rights for IPC mechanisms: read/write stored in struct ipc_perm

```
Struct ipc_perm {
        uid_t uid;
        gid_t gid;
        uid_t  cuid;
        gid_t cgid;
        mode_t mode;
}
```

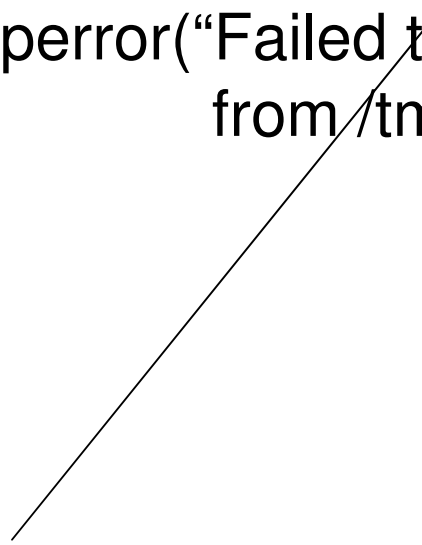
- Include:
  - #include <sys/ipc.h>
  - #include <sys/types.h>

ftok()

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

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

```
if ( (thekey=ftok("/tmp/ad.tempfile",23))  == -1)
        perror("Failed to create key
                from /tmp/ad.tempfile");
```



- File must exist and be accessible to the calling process
- proj_id must be integer between 1 and 255

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>`

Syscalls for sending/receiving messages take as argument a pointer to a structure like the above

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 ακέραιος 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

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

Θυμίζουν δημιουργία αρχείων

- 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 τον τύπο μηνύματος που 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 */
```

Server

```
#include <header file lines omitted here>
…
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L
struct message{
        long mtype;
        char mtext[MSGSIZE];
        };

main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;

        the_key = ftok("/home/ad/K24/MoIC/MesgQueues",
                                226);
        if ( (qid = msgget(the_key,
                                PERMS | IPC_CREAT)) < 0 ){
                perror("megget"); exit(1);
                }
        printf("Creating message queue with
                  identifier %d \n",qid);
```

Server

```
sbuf.mtype = SERVER_MTYPE;
strcpy(sbuf.mtext,"A message from server");
if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0)
        perror("msgsnd"); exit(1);
        }
printf("Sent message: %s\n",sbuf.mtext);

if ( msgrcv(qid, &rbuf, MSGSIZE,
                CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
printf("Received message: %s\n",rbuf.mtext);

if (msgrcv(qid, &rbuf, MSGSIZE, CLIENT_MTYPE, 0) < 0){
        perror("msgrcv"); exit(1);}
printf("Received message: %s\n",rbuf.mtext);

if (msgctl(qid, IPC_RMID, (struct msqid_ds *)0) < 0){
                perror("msgctl"); exit(1);}
printf("Removed message queue with
          identifier %d\n",qid);
}
```

client1

```
.........
#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

struct message{
        long mtype;
        char mtext[MSGSIZE];
        };
main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;

        the_key = ftok("/home/ad/K24/MoIC/MesgQueues",
                                226);
        if ( (qid = msgget(the_key, PERMS)) < 0 ){
                perror("megget"); exit(1);
                }
```

client1

```
printf("Accessing message queue with
        identifier %d \n",qid);

      if ( msgrcv(qid, &rbuf, MSGSIZE,
                  SERVER_MTYPE, 0) < 0){
          perror("msgrcv"); exit(1);}
      printf("Received message: %s\n",rbuf.mtext);

      sbuf.mtype = CLIENT_MTYPE;
      strcpy(sbuf.mtext,"A message from client 1");

      if (msgsnd(qid, &sbuf,
                  strlen(sbuf.mtext)+1, 0) < 0){
          perror("msgsnd"); exit(1);
          }
      printf("Sent message: %s\n",sbuf.mtext);
}
```

client2

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

#define MSGSIZE 128
#define PERMS 0666
#define SERVER_MTYPE 27L
#define CLIENT_MTYPE 42L

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

main(){
        int qid;
        struct message sbuf, rbuf;
        key_t the_key;
```

client2

```
the_key = ftok("/home/ad/K24/MoIC/MesgQueues", 226)

if ( (qid = msgget(the_key, PERMS)) < 0 ){
        perror("megget"); exit(1);
        }
printf("Accessing message queue with
           identifier %d \n",qid);

sbuf.mtype = CLIENT_MTYPE;
strcpy(sbuf.mtext,"A message from client 2");

if (msgsnd(qid, &sbuf, strlen(sbuf.mtext)+1, 0) < 0)
        perror("msgsnd"); exit(1);
        }
printf("Sent message: %s\n",sbuf.mtext);
}
```

Output

```
mema@bowser> ./msg-server
Creating message queue with identifier 0
Sent message: A message from server
Received message: A message from client 1
Received message: A message from client 2
Removed message queue with identifier 0
mema@bowser>
```

Output

```
mema@bowser> ./msg-client1
Accessing message queue with identifier 0
Received message: A message from server
Sent message: A message from client 1
mema@bowser>
```

```
mema@bowser> ./msg-client2
Accessing message queue with identifier 0
Sent message: A message from client 2
mema@bowser>
```

Developing a Priority Queue

- Implement a Queue in which jobs have Priorities
- A server gets the items from the queue and in some way “processes” these items

“q.h”

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <string.h>
#include <errno.h>

#define QKEY          (key_t) 111
#define QPERM          0660
#define MAXOBN          50
#define MAXPRIOR        10

struct q_entry{
        long mtype;
        char mtext[MAXOBN+1];
        };
```

“init_queue.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int warn(char *s){
        fprintf(stderr,"Warning: %s\n",s);
}

int init_queue(void){
        int queue_id;

        if ( (queue_id = msgget(QKEY,
                                IPC_CREAT | QPERM)) == -1 )
                perror("msgget failed");
        return(queue_id);
}
```

“enter.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

int enter(char *objname, int priority){
        int len, s_qid;
        struct q_entry s_entry;

        if ( (len=strlen(objname)) > MAXOBN){
                warn("name too long\n"); exit(1);
                }
        if ( priority > MAXPRIOR || priority < 0 ){
                warn("invalid priority level"); return(-1);
                }
        if ( (s_qid = init_queue()) == -1 ) return(-1);

        s_entry.mtype= (long)priority;
        strncpy(s_entry.mtext, objname, MAXOBN);

        if (msgsnd(s_qid, &s_entry, len, 0) == -1 ){
                perror("msgsnd failed"); return(-1);}
        else    return(0);
}
```

"etest.c"

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"

main(int argc, char *argv[]){
        int priority;

        if ( argc!= 3){
                fprintf(stderr,"usage: %s objname
                                        priority\n",argv[0]);
                }
        if ((priority = atoi(argv[2])) <=0 ||
              priority > MAXPRIOR){
                warn("invalid priority"); exit(2);
                }

        if ( enter(argv[1], priority) < 0 ){
                warn("enter failure"); exit(3);
                }
        exit(0);
}
```

```
gcc enter.c init_queue.c etest.c -o etest
```

“serve.c”

```
#include "q.h"

int serve(void){
        int mlen, r_qid;
        struct q_entry r_entry;

        if ( (r_qid=init_queue()) == -1) return(-1);

      for(;;){
         if ( (mlen=msgrcv(r_qid, &r_entry, MAXOBN,
                 (-1 * MAXPRIOR),
                 MSG_NOERROR) ) == -1 ){
             perror("mesgrcv failed"); return(-1);
         }
         else {
             r_entry.mtext[mlen]='\0';
             proc_obj(&r_entry);
         }
      }
}
```

“stest.c”

```
#include <stdio.h>
#include <stdlib.h>
#include "q.h"
extern void server();
main(){
        pid_t pid;

        switch (pid=fork()){
        case 0: // child
                serve();
                break;
        case -1:
                warn("fork to start the server failed");
                break;
        default:
                printf("server process pid is %d \n", pid);
        }
exit(pid != 1 ? 0 : 1);
}

int proc_obj(struct q_entry *msg){
        printf("\npriority: %ld name: %s\n",
                msg->mtype, msg->mtext);
        }
```

```
gcc stest.c serve.c init_queue.c -o stest
```

```
mema@bowser> ./etest object1 3
mema@bowser> ./etest object2 1
mema@bowser> ./etest object3 6
mema@bowser> ./etest object4 8

mema@bowser> ./stest
server process pid is 2213
priority : 1 name: object2
priority : 3 name: object1
priority : 6 name: object3
priority : 8 name: object4
mema@bowser>
```

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");
```

Private key

Όλοι την προσαρτούν

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

Όλοι την αποδεσμεύουν

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

Ένας μόνο την διαγράφει

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

out1.c

```
#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 memory 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)
```

out1.c

```
/* 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;
}
```

out2.c

```
#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)
```

out2.c

```
/* 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;
}
```

Output

```
mema@bowser> ./out1
Allocated. 262145
Attached. Mem contents 0
Start other process. >
```

```
mema@bowser> ./out2 262145
Id is 262145
Attached. Mem contents 1
Changed mem is now 2
Detachment 0
mema@bowser>
```

```
Start other process. >s
mem is now 2
Removed. 0
mema@bowser>
```

Shared Memory vs Message Queues

- May prefer shared memory when
  - Messages (data) exchanged VERY large (e.g., 100,000 bytes)
  - For performance, to avoid copying messages from user to kernel to user space
- Downside of shared memory
  - Synchronization necessary!

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

Έχουν συμβουλευτικό χαρακτήρα: αν μια διεργασία δεν ελέγξει το σηματοφόρο πριν προσπελάσει κοινόχρηστο πόρο, μπορεί να προκύψει χάος

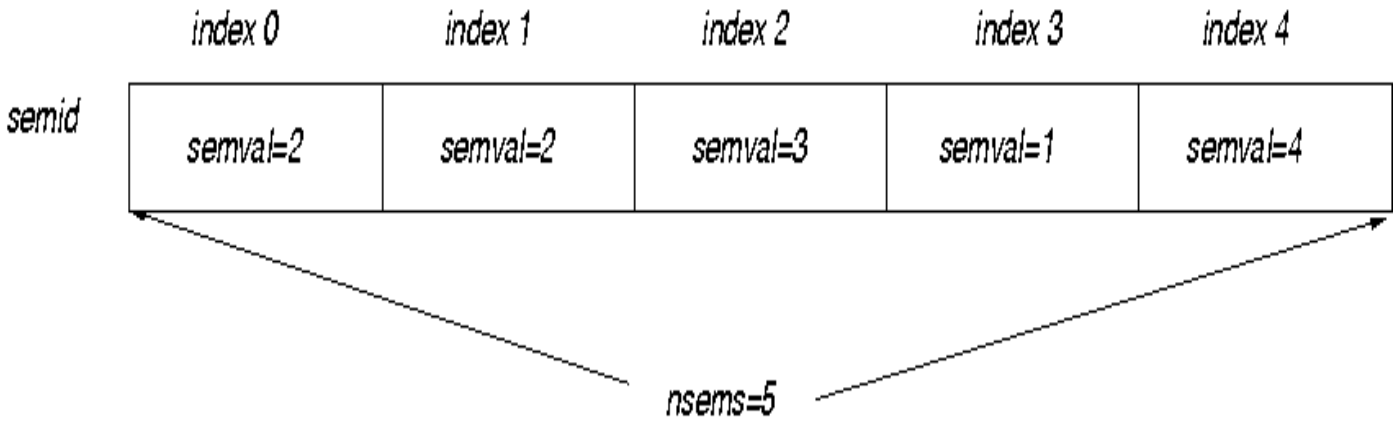
Σηματοφόροι (συν)

- 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 σημαφόρων 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

Structure of a Semaphore set



Associated with *EACH* semaphore in the set are the following (among others) values:

semval: the sem value, always a positive number

sempid: the pid of the process that last "acted" on the sem

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

Μερικές φορές η δέσμευση και αποδέσμευση λέγονται P και V (down and up –Dijkstra's terms).

Παράδειγμα Πρόσβασης σε 2 Ταινίες από 3 διεργασίες

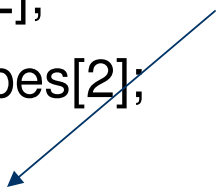
```
#include <sys/sem.h>
void setsembuf(strcut sembuf *s, int num, int op,
     int flg) {
   s->sem_num = (short) num;
   s-> sem_op = (short) op;
   s->sem_flg = (short) flg;
   return;
}

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);
```

- Πρώτο Σηματοφόρο (0)

  Ενέργεια = -1
- Σημαία (flag) = 0



Παράδειγμα (συνέχεια)

Προσπάθησε να αφαιρέσεις 1 (P) από τον πρώτο σημαφόρο

Process 1: semop(S, get_tapes, 1);

<use tape A>

semop(S, release_tapes, 1);

Προσπάθησε να αφαιρέσεις (P) από τους 2 πρώτους σημαφόρους

Process 2: semop(S, get_tapes, 2);

<use tapes A and B>

semop(S, release_tapes, 2);

Προσπάθησε να αφαιρέσεις 1 (P) από το δεύτερο σημαφόρο

Process 3: semop(S, get_tapes + 1, 1);

<use tape B>

semop(S, release_tapes + 1, 1);

Προσπάθησε να προσθέσεις 1 (V) στο δεύτερο σημαφόρο

semget() not enough for initialization

- Μετά από semget κλήση, οι σηματοφόροι δεν μπορούν να χρησιμοποιηθούν αμέσως
- Πρέπει να ετοιμαστούν πρώτα με μια κλήση της semctl

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 */
```

With IPC_STAT command, the fields of the struct semid_ds pointed to by arg.buf are Filled in.

Server (for shared memory + semaphore)

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

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

main(){
        int shmid, semid;
        char line[128], *shmem;
        struct sembuf oper[1]={0,   1,   0};
```

Increase by 1 the value of semaphore #0

Server

```
union semnum arg;

 if ((shmid = shmget (SHMKEY, SHMSIZE,
        PERMS | IPC_CREAT)) < 0) {
        perror("shmget");
        exit(1);
        }

 printf("Creating shared memory with
           ID: %d\n",shmid);

 /* access a SINGLE (1) semaphore from the system */
 if ((semid = semget(SEMKEY, 1,
                  PERMS| IPC_CREAT)) <0) {
        perror("semget");
        exit(1);
        }
 printf("Creating a semaphore with
           ID: %d \n",semid);
```

Server

```
arg.val=0;
/* initialize the semaphore's value  for locking
   set semaphore 0 to have value arg*/
  if (semctl(semid, 0, SETVAL, arg) <0) {
        perror("semctl");
        exit(1);
        }
  printf("Initializing semaphore to lock\n");

   if ( (shmem = shmat(shmid, (char *)0, 0))
          == (char *) -1) {
        perror("shmem");
        exit(1);
        }
  printf("Attaching shared memory
         segment \nEnter a string: ");
  fgets(line, sizeof(line), stdin);
  line[strlen(line)-1]='\0';

  /* Write message in shared memory */
  strcpy(shmem, line);
```

Server

```
 printf("Writing to shared memory region:
         %s\n", line);

  /* Make shared memory available for reading */
  if ( semop(semid, &oper[0], 1) < 0 ) {
       perror("semop");
       exit(1);
  }
 shmdt(shmem);
 printf("Releasing shared memory region\n");
}
```

Take a single action (3rd argument)

Client

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SHMKEY (key_t)4321
#define SEMKEY (key_t)9876
#define SHMSIZE 256
#define PERMS 0600

main(){
        int shmid, semid;
        char *shmem;
        struct sembuf oper[1]={ 0,  -1,   0};     // down
```

Down semaphore

Client

```
//  get access to shared memory segment
if ((shmid = shmget (SHMKEY, SHMSIZE,
                              PERMS )) < 0) {
            perror("shmget");
            exit(1);
            }
printf("Accessing shared memory
              with ID: %d\n",shmid);

  // accessing the SINGLE  (one) semaphore
     (from the system)
 if ((semid = semget(SEMKEY, 1,
                              PERMS )) <0) {
            perror("semget");
            exit(1);
            }
printf("Accessing semaphore with
          ID: %d \n",semid);
```

Client (continued)

```
if ( (shmem = shmat(shmid, (char *) 0, 0))
                    == (char *) -1 ) {
        perror("shmat");
        exit(1);
        }
   printf("Attaching shared memory segment\n");

   printf("Asking for access to shared memory
           region \n");
   if (semop(semid, &oper[0], 1) <0)  {
        perror("semop");
        exit(1);
        }
   printf("Reading from shared memory
           region: %s\n", shmem);

   /* detach shared memory */
   shmdt(shmem);
```

Take one (1) action

Client (continued)

```
/* destroy shared memory */
if (shmctl(shmid, IPC_RMID,
                (struct shmid_ds *)0 ) <0) {
        perror("semctl");
        exit(1);
        }
printf("Releasing shared segment with
          identifier %d\n", shmid);

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

Output

```
mema@bowser> ./sem-server
Creating shared memory with ID: 360449
Creating a semaphore with ID: 819203
Initializing semaphore to lock
Attaching shared memory segment
Enter a string: have a good day!
Writing to shared memory region: have a good day
Releasing shared memory region
```

```
mema@bowser> ./sem-client.c
Accessing shared memory with ID: 360449
Accessing semaphore with ID: 819203
Attaching shared memory segment
Asking for access to shared memory region
Reading from shared memory region: have a good
day!
Releasing shared segment with identifier 360449
Releasing semaphore with identifier 819203
mema@bowser>
```

Example: Access to Critical Section

```
/* Example code using "POSIX-like" calls for
semaphores and shared memory */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/sem.h>
#include <sys/ipc.h>

/* Union semun */
union semun {
   int val;                    /* value for SETVAL */
   struct semid_ds *buf; /*buffer for IPC_STAT, IPC_SET */
   unsigned short *array;  /* array for GETALL, SETALL */
};
void free_resources(int shm_id, int sem_id) {
   /* Destroy the shared memory segment */
   shmctl(shm_id,IPC_RMID,NULL);
   /* Destroy the semaphore */
   semctl(sem_id,0,IPC_RMID,0);
}
```

Example: Access to Critical Section

```
/* Semaphore P - down operation, using semop
    [simulates sem_wait call in POSIX for reserving
      a resource.] */
int sem_P(int sem_id) {
   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = -1;
   sem_d.sem_flg = 0;
   if (semop(sem_id, &sem_d, 1) == -1) {
       perror("# Semaphore down (P) operation ");
       return -1;
   }
   return 0;
}
```

Example: Access to Critical Section

```
/* Semaphore V - up operation, using semop
   [simulates sem_post call in POSIX for releasing
    a resource.] */
int sem_V(int sem_id) {

   struct sembuf sem_d;

   sem_d.sem_num = 0;
   sem_d.sem_op = 1;
   sem_d.sem_flg = 0;
   if (semop(sem_id, &sem_d, 1) == -1) {
       perror("# Semaphore up (V) operation ");
       return -1;
   }
   return 0;

}
```

Example: Access to Critical Section

```
/* Semaphore Init - set a semaphore's value
    to val */
int sem_Init(int sem_id, int val) {

   union semun arg;

   arg.val = val;
   if (semctl(sem_id, 0, SETVAL, arg) == -1) {
      perror("# Semaphore setting value ");
      return -1;
   }
   return 0;

}
```

Example: Access to Critical Section

```
int main () {
   int shm_id;
   int sem_id;
   int t = 0;
   int *sh;
   int pid;

   /* Create a new shared memory segment */
   shm_id = shmget(IPC_PRIVATE,sizeof(int),
                              IPC_CREAT | 0660);
   if (shm_id == -1) {
       perror("Shared memory creation");
       exit(EXIT_FAILURE);
   }
   /* Create a new semaphore id */
   sem_id = semget(IPC_PRIVATE, 1,
                              IPC_CREAT | 0660);
   if (sem_id == -1) {
       perror("Semaphore creation ");
```

Example: Access to Critical Section

```
    shmctl(shm_id,IPC_RMID,
        (struct shmid_ds *)NULL);
    exit(EXIT_FAILURE);
}

/* Set the value of the semaphore to 1 */
if (sem_Init(sem_id, 1) == -1) {
    free_resources(shm_id,sem_id);
    exit(EXIT_FAILURE);
}

/* Attach the shared memory segment */
sh = (int *)shmat(shm_id,NULL,0);
if (sh == NULL) {
    perror("Shared memory attach ");
    free_resources(shm_id,sem_id);
    exit(EXIT_FAILURE);
}
/* Setting shared memory to 0 */
*sh = 0;
```

```
/* New process */
 if ((pid = fork()) == -1) {
     perror("fork");
     free_resources(shm_id,sem_id);
     exit(EXIT_FAILURE);
 }

 if (pid == 0) {
    /* Child process */
    printf("# I am the child process with
              process id: %d\n", getpid());
 } else {
    /* Parent process */
    printf("# I am the parent process with
              process id: %d\n", getpid());
    sleep(2);
 }

 printf("(%d): trying to access the critical
            section\n", getpid());
 sem_P(sem_id);
 printf("(%d): accessed the critical
           section\n", getpid());
```

```
    (*sh)++;
    printf("(%d): value of shared memory is
            now: %d\n", getpid(), *sh);

    printf("(%d): getting out of the critical
            section\n", getpid());
    sem_V(sem_id);

    printf("(%d): got out of the critical
             section\n", getpid());

    /* Child process */
    if (!pid)
      exit(EXIT_SUCCESS);

    /* Wait for child process */
    wait(NULL);

    /* Clear recourses */
    free_resources(shm_id,sem_id);
    return 0;
}
```

Output

```
mema@bowser> ./posix-sample
# I am the child process with process id: 9109
(9109): trying to access the critical section
(9109): accessed the critical section
(9109): value of shared memory is now: 1
(9109): getting out of the critical section
(9109): got out of the critical section
# I am the parent process with process id: 9108
(9108): trying to access the critical section
(9108): accessed the critical section
(9108): value of shared memory is now: 2
(9108): getting out of the critical section
(9108): got out of the critical section
mema@bowser>
```

Semaphore challenges

Like threads, synchronizing processes via semaphores entails three challenges:

1. Ensuring mutual exclusion when using a resource
2. Avoiding deadlocks
3. Avoiding starvation (where a process is never able to access the resource)

Locking a File

- Another synchronization mechanism between processes
- File = resource
  - Could use semaphores to synchronize access by processes to files
  - But Unix supplies special syscall in case of files called *fcntl*

Locking a file

read – write locks (on sections of) files.

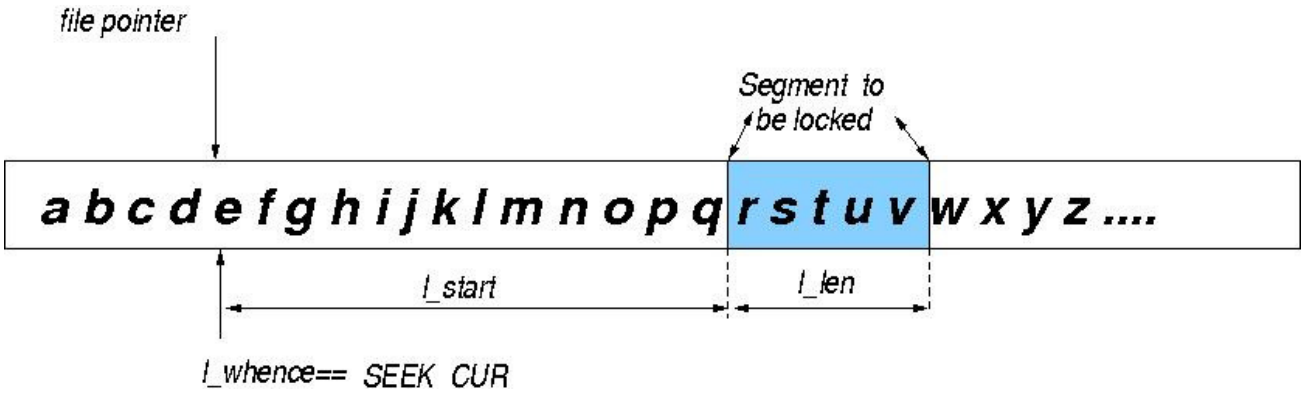
#include <fnctl.h>

int fcntl(int filedes, int cmd, struct flock *ldata)

Notes:

- The file filedes must be opened with O_RDONLY or O_WRONLY.
- The cmd can be one of the three:
  - F_GETLK: get lock from data returned from ldata (i.e., check if there is a lock already on it)
  - F_SETLK: obtain lock on a file; return immediately if this is not feasible.
  - F_SETLKW: obtain lock on a file. Block, if lock held by another process.
- The flock structure defined in <fnctl.h> includes:
  - short l_type; /* describes lock type: F_RDLCK, F_WRLCK, F_UNLCK */
  - short l_whence; /*SEEK_SET, SEEK_CUR, SEEK_END (determines where l_start field starts) */
  - off_t l_start; // starting offset, relative to l_whence
  - off_t l_len; // segment size in bytes
  - pid_t l_pid; // ID of process dealing with the lock

Locking a file



l_whence: can be SEEK_SET, SEEK_CUR or SEEK_END
l_start: start position of the segment
l_len: segment in bytes

The l_type (lock type) can be:
- F_RDLCL: lock to be applied is *read*
- F_WRLCL: lock to be applied is *write*
- F_UNLCK: lock on specified segment to be removed

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

main( ){
        int fd;
        struct flock my_lock;

        my_lock.l_type = F_WRLCK;
        my_lock.l_whence =  SEEK_SET;
        my_lock.l_start = 0 ;
        my_lock.l_len= 10;

        fd=open("locktest", O_RDWR);

        // lock first 10 bytes
        if ( fcntl(fd, F_SETLKW, &my_lock) == -1 ){
                perror("parent: locking");
                exit(1);
                }

        printf("parent: locked record \n");
```

```
switch(fork()){
    case -1:
        perror("fork"); exit(1);
    case 0:
        printf("child: trying to lock file \n");
        my_lock.l_len = 5 ;
        if ( (fcntl(fd, F_SETLKW, &my_lock)) == -1 ){
                perror("child: problem in locking");
                exit(1);
        }
        printf("child: locked \n"); sleep(1);
        printf("child: exiting \n");
        fflush(stdout); fflush(stderr); exit(1);
    default:
       printf("parent: just about unlocking now \n");
       sleep(5);
       my_lock.l_type = F_UNLCK;
       printf("parent: unlocking -now- \n");
       if ( fcntl(fd, F_SETLK, &my_lock) == -1 ){
              perror("parent: problem in unlocking! \n");
              exit(1); }
       printf("parent: has unlocked and is now exiting \n");
       fflush(stdout); fflush(stderr); wait(NULL);
       }
       sleep(2); }
```

Execution outcome

```
mema@bowser> ./lockit
parent: locked record
child: trying to lock file
parent: just about unlocking now
parent: unlocking -now-
child: locked
parent: has unlocked and is now exiting
child: exiting
mema@bowser>
```

Deadlock possible

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

main( ){
        int fd;
        struct flock first_lock;
        struct flock second_lock;

        first_lock.l_type = F_WRLCK;
        first_lock.l_whence =  SEEK_SET;
        first_lock.l_start = 0 ;
        first_lock.l_len= 10;

        second_lock.l_type = F_WRLCK;
        second_lock.l_whence =  SEEK_SET;
        second_lock.l_start = 10;
        second_lock.l_len= 5;

        fd=open("locktest", O_RDWR);
        if ( fcntl(fd, F_SETLKW, &first_lock) == -1 )
                perror("-A:");
       printf("A: lock obtained by process %d \n",getpid());
```

Deadlock possible

```
switch(fork()) {
   case -1:
        perror("error on fork");
        exit(1);
   case 0: /* child */
        if ( fcntl(fd, F_SETLKW, &second_lock) == -1)
            perror("-B:");
        printf("B: lock obtained by process
                    %d\n",getpid());

        if ( fcntl(fd, F_SETLKW, &first_lock) == -1 ){
            perror("-C:");
            printf("Process %d terminating\n",
                     getpid());
            exit(1);
        }
        else printf("C: lock obtained by process
                      %d\n",getpid());
        printf("Process %d successfully acquired
                 BOTH locks \n", getpid());
        exit(0);
```

Deadlock possible

```
default:   /* parent */
   printf("Parent process %d sleeping \n",getpid());
   sleep(10);
   // returns immediately – F_SETLK
   if ( fcntl(fd, F_SETLK, &second_lock) == -1 ){
        perror("--D:");
        printf("Process %d about to terminate\n",getpid());
   }
   else printf("D: lock obtained by process
                    %d\n",getpid());
   sleep(1);
   printf("Process %d on its way out of here
            \n",getpid());
}
   }
```

Execution outcome

```
mema@bowser> ./deadlock
A: lock obtained by process 18979
B: lock obtained by process 18980
Parent process 18979 sleeping
--D:: Resource temporarily unavailable
Process 18979 about to terminate
Process 18979 on its way out of here
C: lock obtained by process 18980
Process 18980 successfully acquired BOTH locks
mema@bowser>
```