Shared Memory
Semaphores

Goals of this Lecture

• Understand the design and programming of the System V Unix shared memory interprocess communication mechanism and its relation with the Unix memory management functions.

• Understand the general concepts of semaphores.

• Understand the design and programming of the System V Unix semaphore interprocess communication mechanism.
System V IPC Shared Memory

- Shared memory allows sharing the same physical memory content between different processes
- Uses a dedicated segment type, but is based on the regular Unix memory management
- Allows communication between non-related processes
- But: limited to local Unix system

Creating a Shared Memory Segment

```c
#include <sys/types.h> /* supplies key_t */
#include <sys/ipc.h>   /* supplies ipc_perm */
#include <sys/shm.h>   /* supplies structures and macros for
                      * shared mem. data structures etc. */

int size, permflags, shm_id;
key_t key;
...
shm_id = shmget (key, size, permflags);
```
The allocreg() Pseudocode

```c
algorithm allocreg /* M. J. Bach, The Design of the */
input: (1) inode pointer /* UNIX Operating System, Figure 6.18 */
       (2) region type
output: locked region
{
    remove region from linked list of free regions;
    assign region type;
    assign region inode pointer;
    if (inode pointer not null)
        increment inode reference count;
    place region on linked list of active regions;
    return (locked region);
}
```

Including a Shared Memory Segment into a Process

```c
#include <sys/types.h> /* supplies key_t */
#include <sys/ipc.h>   /* supplies ipc_perm */
#include <sys/shm.h>   /* supplies structures and macros for *
                      * shared mem. data structures etc. */

int shm_id, shmflags;
char *memptr, *daddr, *shmat();
...

memptr = shmat (shm_id, daddr, shmflags);
```

Haviland/Salama, UNIX System Programming, page 200
The shmat() Pseudocode

algorithm shmat /* M. J. Bach, The Design of the */
/* UNIX Operating System, Figure 11.10 */

input: (1) shared memory descriptor
(2) virtual address to attach memory
(3) flags
output: virtual address where memory was attached
{
    check validity of descriptor, permissions
    if (user specified virtual address) {
        round off virtual address, as specified by flags;
        check legality of virtual address, size of region;
    }
    else /* user wants kernel to find good address */
        kernel picks virtual address: error if none available;
        attach region to process address space (algorithm attachreg);
        if (region being attached for the first time)
            allocate page tables, memory for region (algorithm growreg);
        return (virtual address where attached);
}

Example Process Address Layout

[Diagram showing the example process address layout]

User Context
- Stack
- Heap
- Uninitialised Data
- initialised Data
- Text

Kernel Context
- Kernel Data

Automatic Allocation of Shared Memory Segments

Read from executable file
The attachreg() Pseudocode

algorithm attachreg /* M. J. Bach, The Design of the */ /* UNIX Operating System, Figure 6.19 */
input: (1) pointer to (locked) region being attached
(2) process to which region is being attached
(3) virtual address in process where region will be attached
(4) region type
output: per process region table entry
{
  allocate per process region table entry for process;
  initialize per process region table entry;
  set pointer to region being attached;
  set type field;
  set virtual address field;
  check legality of virtual address, region size;
  increment region reference count;
  increment process size according to attached region;
  initialize new hardware register triple for process;
  return (per process region table entry);
}

Data Structures After shmat()

M.J. Bach, The Design of the UNIX Operating System, Figure 11.9
Removing a Shared Memory Segment from a Process

```c
int      retval;
char    *memptr;

retval = shmdt (memptr);
```

Why does `shmdt()` not use an IPC Object Identifier as its argument?

The detachreg() Pseudocode

```c
algorithm detachreg /* M. J. Bach, The Design of the */
        /* UNIX Operating System, Figure 6.26 */
input: pointer to per process region table entry
output: none
{
    get auxiliary memory management tables for process,
    release as appropriate;
    decrement process size;
    decrement region reference count;
    if (region count is 0 and region not sticky bit)
        free region (algorithm freereg);
    else { /* either reference count non-0 or 
            region sticky bit on */
        free inode lock, if applicable
        (inode associated with region);
        free region lock;
    }
}```
Controlling a Shared Memory Segment

```c
#include <sys/types.h> /* supplies key_t */
#include <sys/ipc.h>   /* supplies ipc_perm */
#include <sys/shm.h>   /* supplies structures and macros for
* shared mem. data structures etc. */

int shm_id, command, retval;
struct shmid_ds shm_stat; /* shm id data structure */
...
retval = shmctl (shm_id, command, &shm_stat);
```

Shared Memory Control Structure

```c
struct shmid_ds {
    struct ipc_perm shm_perm; /* operation permission struct */
    short shm_segsz; /* size of segment in bytes */
    /* struct region */
    char pad[4]; /* for swap compatibility */
    short shm_lpid; /* pid of last shmop */
    short shm_cpid; /* pid of creator */
    short shm_nattch; /* region attach counter */
    /* shm_kaddr added / shm_reg removed */
    time_t shm_atime; /* last shmat time */
    time_t shm_dtime; /* last shmdt time */
    time_t shm_ctime; /* last change time */
    /* PRE-VM-REWRITE */
    /* NOTE: shm_kaddr added / shm_reg removed */
    uint shm_kaddr; /* kernel v-space */
};
```
System Parameters for Shared Memory

/*
 * Configuration Parameters
 * These parameters are tuned by editing the system config. file.
 * The following lines establish the default values.
 */
#ifndef SHMPOOL
#define SHMPOOL 512 /* max total shared mem system wide (in Kb) */
#endif
#ifndef SHMSEG
#define SHMSEG 6 /* max attached shared mem segments per proc. */
#endif
#ifndef SHMMNI
#define SHMMNI 100 /* # of shared mem identifiers */
#endif
/* The following parameters are assumed not to require tuning */
#define SHMMIN 1 /* min shared mem segment size ... */

Shared Memory Program Example I

/* M. J. Bach, The Design of the
 * UNIX Operating System, Figure 11.11 */
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define SHMKEY 75
#define K 1024
#define MAXCNT 10
int shmid;
main () {
    int i, *pint;
    int pid;
    char *addr1, *addr2;
    struct shmid_ds shm_status;
    extern char *shmat();
    extern cleanup();
    extern void shmstat_print();
    for (i=0; i<32; i++)
        signal (i, cleanup);
Shared Memory Program Example II

```c
shmid = shmget (SHMKEY, 128 * K, 0777|IPC_CREAT);
shmctl (shmid, IPC_STAT, &shm_status);
shmstat_print(SHMKEY, shmid, &shm_status);
addr1 = shmat (shmid, 0, 0);
addr2 = shmat (shmid, 0, 0);
printf ("\n   addr1 0x%x addr2 0x%x\n", addr1, addr2);
shmctl (shmid, IPC_STAT, &shm_status);
shmstat_print(SHMKEY, shmid, &shm_status);

pint = (int *) addr1;
for (i=0; i<MAXCNT; i++)
   *pint++ = i;
pint = (int *) addr1;
*pint = MAXCNT;

pint = (int *) addr2;
printf ("\n   index %d\tvalue %d\n", i, *pint);
```

Shared Memory Program Example III

```c
shmctl (shmid, IPC_STAT, &shm_status);
shmstat_print(SHMKEY, shmid, &shm_status);
shmctl (shmid, IPC_RMID, 0);
exit ();
```
Shared Memory Program Example IV

```c
void shmstat_print (mkey, shmid, sstat)
key_t mkey; int shmid;
struct shmid_ds *sstat; {
    char *ctime();
    char *atime;
    printf ("\nKey %d, shmid %d, ", mkey, shmid);
    printf ("Last change %s", ctime (&(sstat->shm_ctime)));
    printf ("Access rights %o\n", sstat->shm_perm.mode);
    printf ("uid of creator %d, gid of creator %d, ",
            sstat->shm_perm.cuid, sstat->shm_perm.cgid);
    printf ("uid of owner %d, gid of owner %d\n",
            sstat->shm_perm.uid, sstat->shm_perm.cgid);
    printf ("size of segment %d, pid of last shmem %d\n",
            sstat->shm_segsz, sstat->shm_lpid);
    printf ("pid of creator %d, ", sstat->shm_cpid);
    printf ("region attach counter %d\n", sstat->shm_nattch);
    atime= ctime (&(sstat->shm_atime));
    atime[strlen(atime)-1] = '\0';
    printf ("Last shmat %s, ", atime);
    printf ("Last shmdt %s", ctime (&(sstat->shm_dtime)));
}
```

Program Execution Result I

Key 75, shmid 2100, Last change Fri Dec 12 07:47:37 2003
Access rights 101777
uid of creator 276, gid of creator 100, uid of owner 276, gid of owner 100
size of segment 131072, pid of last shmem 0
pid of creator 18951, region attach counter 0
Last shmat Thu Jan  1 00:00:00 1970, Last shmdt Thu Jan  1 00:00:00 1970
```
addr1 0x26000 addr2 0x48000
```

Key 75, shmid 2100, Last change Fri Dec 12 07:47:37 2003
Access rights 100777
uid of creator 276, gid of creator 100, uid of owner 276, gid of owner 100
size of segment 131072, pid of last shmem 18951
pid of creator 18951, region attach counter 2
Last shmat Fri Dec 12 07:47:37 2003, Last shmdt Thu Jan  1 00:00:00 1970
```
Program Execution Result II

<table>
<thead>
<tr>
<th>index 0</th>
<th>value 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>index 1</td>
<td>value 1</td>
</tr>
<tr>
<td>index 2</td>
<td>value 2</td>
</tr>
<tr>
<td>index 8</td>
<td>value 8</td>
</tr>
<tr>
<td>index 9</td>
<td>value 9</td>
</tr>
</tbody>
</table>

Key 75, shmid 2100, Last change Fri Dec 12 07:47:37 2003
Access rights 100777
uid of creator 276, gid of creator 100, uid of owner 276, gid of owner 100
size of segment 131072, pid of last shmop 18951
pid of creator 18951, region attach counter 2
Last shmat Fri Dec 12 07:47:37 2003, Last shmdt Thu Jan  1 00:00:00 1970

Key 75, shmid 2100, Last change Fri Dec 12 07:47:37 2003
Access rights 100777
uid of creator 276, gid of creator 100, uid of owner 276, gid of owner 100
size of segment 131072, pid of last shmop 18951
pid of creator 18951, region attach counter 0

Shared Memory Size Indicators

<table>
<thead>
<tr>
<th>AT&amp;T</th>
<th>DEC Vax</th>
<th>AT&amp;T</th>
<th>Xenix</th>
<th>Linux</th>
<th>SunOS-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sys V R2</td>
<td>Ultrix</td>
<td>UNIX-PC</td>
<td>386</td>
<td>2.4</td>
<td>Solaris</td>
</tr>
<tr>
<td>131072</td>
<td>131072</td>
<td>65536</td>
<td>4194304</td>
<td>33554432</td>
<td>1048576</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>100</td>
<td>25</td>
<td>4096</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>4096</td>
<td>6</td>
</tr>
<tr>
<td>8192</td>
<td>21768</td>
<td>8192</td>
<td>0</td>
<td>xxxxxxxx</td>
<td>7</td>
</tr>
</tbody>
</table>

max. # bytes per segment
min. # bytes per segment
max. # segments/system
max. # segments/process
shmbrk size
System V IPC Semaphores

- Semaphores allow the synchronisation of independent processes, as well as the protection of critical sections though mutual exclusion.
- Semaphores in Unix are support sets, and are additive.
- But: limited to local Unix system

Use of Semaphores for Mutual Exclusion

s: semaphore (1);

P1 : process
....
P(s);
... -- critical section
V(s);
...
end process

P2 : process
....
P(s);
... -- critical section
V(s);
...
end process

Herrtwich/Hommel, Kooperation und Konkurrenz, 1989
Use of Semaphores for Process Synchronisation

\[ s: \text{semaphore} \ (0); \]
\[ \text{P1 : process} \]
\[ \ldots \]
\[ \text{V}(s); \quad \text{-- signal event} \]
\[ \ldots \]
\[ \text{end process} \]

\[ \text{P2 : process} \]
\[ \ldots \]
\[ \text{P}(s); \quad \text{-- wait for event} \]
\[ \ldots \]
\[ \text{end process} \]

Herrtwich/Hommel, Kooperation und Konkurrenz, 1989

Additive Semaphores

\[ \text{exclusion : add\_semaphore} \ (N); \]
\[ \text{type reader = process} \]
\[ \ldots \]
\[ \text{P} \ (\text{exclusion, 1}); \]
\[ \ldots \quad \text{-- read within critical section} \]
\[ \text{V} \ (\text{exclusion, 1}); \]
\[ \text{end process}; \]

\[ \text{type writer = process} \]
\[ \ldots \]
\[ \text{P} \ (\text{exclusion, N}); \]
\[ \ldots \quad \text{-- write within critical section} \]
\[ \text{V} \ (\text{exclusion, N}); \]
\[ \text{end process}; \]

Herrtwich/Hommel, Kooperation und Konkurrenz, 1989
Semaphores in System V Unix

P(sem)
if (sem != 0)
decrement sem value by one
else
wait until sem becomes non-zero

V (sem)
if (queue of waiting processes not empty)
restart first process in wait queue
else
increment sem value by one

Haviland/Salama, UNIX System Programming, Seite 191

Allocating A Semaphore Set

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

key_t key;
int sem_id, nsems, permflags, command;
int retval, sem_num;

union semun {
    int val;
    struct semid_ds *stat;
    ushort *array;
} ctl_arg;

...
sem_id = semget (key, nsems, permflags);
...
retval = semctl (sem_id, sem_num, command, ctl_arg);

Haviland/Salama, UNIX System Programming, page 192
The Semaphore System

Data Structures

Semaphore Table
Semaphore Arrays

Operations on Semaphores

Standard Operations:
- IPC_STAT: Copy status information of object into structure ctl_arg.stat
- IPC_SET: Set ownership / permission for object from ctl_arg.stat
- IPC_RMID: Remove semaphore set from the system

Operations on Individual Semaphores:
- GETVAL: Returns the semaphore value (semval)
- SETVAL: Sets the semaphore value (semval) to the value in ctl_arg.val
- getpid: Returns the value of sempid
- getncnt: Returns the value of semncnt
- getzcnt: Returns the value of semzcnt

Operations on all Semaphores of Set:
- GETALL: The semval values of all semaphores in set are stored in ctl_arg.array
- SETALL: The semval values of all semaphores in set are set according to values in ctl_arg.array

M.J. Bach, The Design of the UNIX Operating System, Figure 11.13

HAVILAND/SALAMA, UNIX SYSTEM PROGRAMMING, PAGE 193

Hovland/Salama, UNIX System Programming, page 193
C Data Structures for Semaphores I

/* Semctl Command Definitions. */
#define GETNCNT 3 /* get semncnt */
#define GETPID 4 /* get sempid */
#define GETVAL 5 /* get semval */
#define GETALL 6 /* get all semval's */
#define GETZCNT 7 /* get semzcnt */
#define SETVAL 8 /* set semval */
#define SETALL 9 /* set all semval's */

/* Structure Definitions. */
/* There is one semaphore id data structure for each set of semaphores */
/* in the system. */
struct semid_ds {
    struct ipc_perm sem_perm; /* operation permission struct */
    struct sem *sem_base; /* ptr to first semaphore in set */
    ushort sem_nsems; /* # of semaphores in set */
    time_t sem_otime; /* last semop time */
    time_t sem_ctime; /* last change time */
};

C Data Structures for Semaphores II

/* There is one semaphore structure for each semaphore in the system. */
struct sem {
    ushort semval; /* semaphore text map address */
    short sempid; /* pid of last operation */
    ushort semncnt; /* # awaiting semval > cval */
    ushort semzcnt; /* # awaiting semval = 0 */
};

/* 'arg' argument template for semctl system calls. */
union semun {
    int val; /* value for SETVAL */
    struct semid_ds *buf; /* buffer for IPC_STAT & IPC_SET */
    ushort *array; /* array for GETALL & SETALL */
};
Programming Example
Initialising a Semaphore Set

```c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <errno.h>

#define SEMPERM 0600

int initsem (semkey) key_t semkey;
{
    int status = 0, semid;
    if ((semid = semget (semkey, 1, SEMPERM|IPC_CREAT|IPC_EXCL)) == -1) {
        if (errno == EEXIST)
            semid = semget (semkey, 1, 0);
        else /* if semaphore successfully created */
            status = semctl (semid, 0, SETVAL, 1);
        if (semid == -1 || status == -1) {
            perror ("initsem failed");
            return (-1);
        } else
            return semid; /* everything is ok */
    }
}
```

The Semaphore Operation and the semop() Data Structure

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

int retval, sem_id;
struct sembuf op_array[SOMEVALUE];
...
retval = semop (sem_id, op_array, SOMEVALUE);

/* User semaphore template for semop system calls. */
struct sembuf {
    short sem_num; /* semaphore # */
    short sem_op; /* semaphore operation */
    short sem_flg; /* operation flags */
};
```
The Semaphore Undo Operation

/*
 * There is one undo structure per process in the system.
 */

struct sem_undo {
    struct sem_undo *un_np; /* ptr to next active undo structure */
    short un_cnt; /* # of active entries */
    struct undo {
        short un_aoe;        /* "adjust on exit"-values */
        short un_num;        /* semaphore # */
        int un_id;           /* semid */
        } un_ent[1];        /* (semume) undo entries (one min.) */
};
System Parameters for Shared Memory I

/* semaphore information structure */
struct seminfo {
    int semmap, /* # of entries in semaphore map */
    semni,    /* # of semaphore identifiers */
    semns,    /* # of semaphores in system */
    semnu,    /* # of undo structures in system */
    semmsl,   /* max # of semaphores per id */
    semopm,   /* max # of operations per semop call */
    semume,   /* max # of undo entries per process */
    semusz,   /* size in bytes of undo structure */
    semvmx,   /* semaphore maximum value */
    semaem;   /* adjust on exit max value */
};
struct seminfo seminfo; /* configuration parameters */

System Parameters for Shared Memory II

/* Implementation Constants. */
#define PSEMN (PZERO + 3) /* sleep priority waiting for greater value */
#define PSEMZ (PZERO + 2) /* sleep priority waiting for zero */
#define SEMVMX 32767    /* semaphore maximum value */
#define SEMAEM 16384    /* adjust on exit max value */
/* Permission Definitions. */
#define SEM_A 0200    /* alter permission */
#define SEM_R 0400    /* read permission */
/* Configuration Parameters */
/* These parameters are tuned by editing the system configuration * file. The following lines establish the default values. */
#ifndef SEMMNI
#define SEMMNI 10    /* # of semaphore identifiers */
#endif
System Parameters for Shared Memory III

```c
#ifndef SEMMNS
#define SEMMNS 60 /* # of semaphores in system */
#endif
#ifndef SEMUME
#define SEMUME 10 /* max # of undo entries per process */
#endif
#ifndef SEMMNU
#define SEMMNU 30 /* # of undo structures in system */
#endif
/* The following parameters are assumed not to require tuning */
#ifndef SEMMAP
#define SEMMAP 30 /* # of entries in semaphore map */
#endif
#ifndef SEMMSL
#define SEMMSL SEMMNS /* max # of semaphores per id */
#endif
#ifndef SEMOPM
#define SEMOPM 100 /* max # of operations per semop call */
#endif
/* size in bytes of undo structure */
#define SEMUSZ (sizeof(struct sem_undo)+sizeof(struct undo)*SEMUME)
```

Programming Example Implementing “P” and “V”

```c
P (semid) int semid; {
    struct sembuf p_buf;
    p_buf.sem_num = 0;
    p_buf.sem_op = -1; /* negativer Wert, also Fall 1 = P() */
    p_buf.sem_flg = SEM_UNDO;
    if (semop (semid, &p_buf, 1) == -1) {
        perror ("p(semid) failed");
        exit (1);
    } else return (0);
}

V (semid) int semid; {
    struct sembuf v_buf;
    v_buf.sem_num = 0;
    v_buf.sem_op = 1; /* positiver Wert, also Fall 2 = V() */
    v_buf.sem_flg = SEM_UNDO;
    if (semop (semid, &v_buf, 1) == -1) {
        perror ("v(semid) failed");
        exit (1);
    } else return (0);
}
```

Haviland/Salama, UNIX System Programming, page 197
Programming Example
Using “P” and “V”

```c
main () {
    key_t semkey = 0x200;
    if (fork () == 0) handlesem (semkey);
    if (fork () == 0) handlesem (semkey);
    if (fork () == 0) handlesem (semkey);
}
handlesem (skey) key_t skey; {
    int semid, pid = getpid();
    if ((semid = initsem (skey)) > 0) exit (1);
    printf ("%d before critical section\n", pid);
    P (semid);
    printf ("%d in critical section\n", pid);
    /* in real life do something interesting */
    sleep (10);
    printf ("%d leaving critical section\n", pid);
    V (semid);
    printf ("%d exiting\n", pid);
    exit (0);
}
```

Program Execution Result

```
process 799 before critical section
process 800 before critical section
process 801 before critical section
process 799 in critical section
process 799 leaving critical section
process 799 exiting
process 801 in critical section
process 801 leaving critical section
process 801 exiting
process 800 in critical section
process 800 leaving critical section
process 800 exiting
```
### Semaphores Size Indicators

<table>
<thead>
<tr>
<th>AT&amp;T Sys V R2</th>
<th>DEC Vax Ultras</th>
<th>AT&amp;T UNIX-PC</th>
<th>Xenix 286/386</th>
<th>Linux 2.4</th>
<th>SunOS-4 Solaris</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>128</td>
<td>10</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
<td>60</td>
<td>40</td>
<td>32000</td>
<td>60</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>25</td>
<td>10</td>
<td>250</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>30</td>
<td>20</td>
<td>32000</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>32767</td>
<td>32767</td>
<td>32767</td>
<td>32766</td>
<td>32767</td>
<td>32767</td>
</tr>
<tr>
<td>16384</td>
<td>16384</td>
<td>16383</td>
<td>16384</td>
<td>16383</td>
<td>16384</td>
</tr>
</tbody>
</table>

- **max. # unique semaphore sets**
- **max. # semaphores**
- **max. # semaphores/set**
- **max. # operations/semop()**
- **max. # undo-structures**
- **max. # undo-entries/struc.**
- **max. value of any semaphore**
- **max. value for any ace**

### Summary: Shared Mem./Semaphores

- Shared memory segments allow the “native” sharing of data between independent processes, based on the operating system’s memory management functions.
- There is no built-in process synchronisation for shared memory.
- Semaphores allow mutual exclusion as well as process synchronisation of independent processes.
- Unix semaphores can bet sets and are additive, to provide support for multiple reader/writer scenarios.
- Unix semaphores can use an UNDO structure per operation to deal with processes exiting within a critical section.
- Most Unix variants (including Linux) use semaphores internally to protect critical sections of the operating system.
- POSIX defines another, incompatible shared memory concept, as well as semaphores concept (named / unnamed via files).