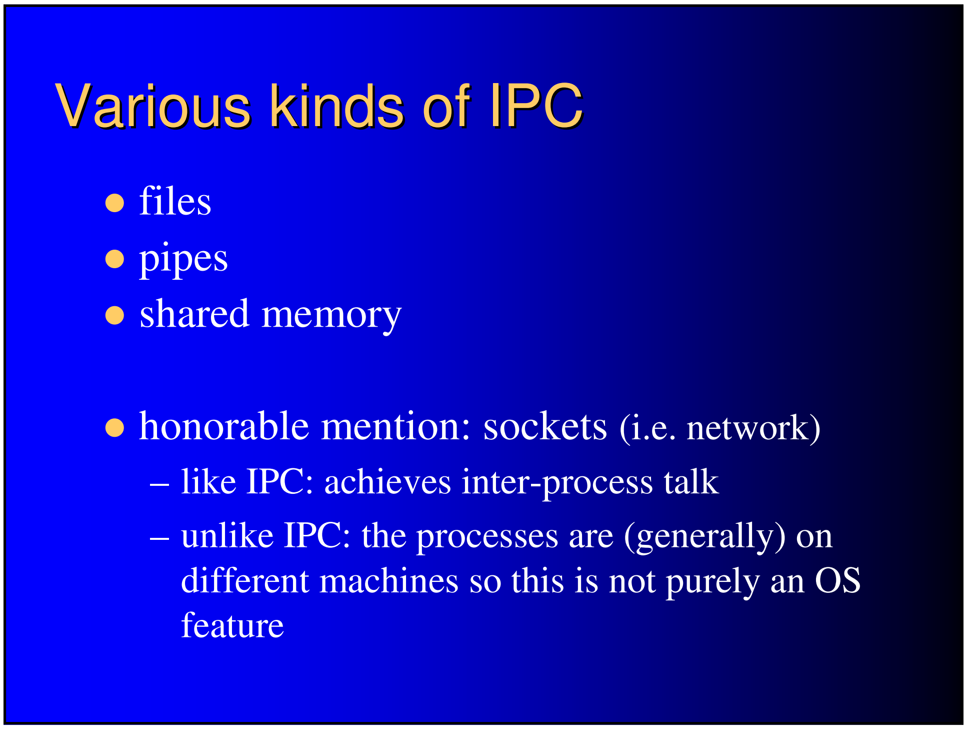
Interprocess communication (IPC)

operating system mechanisms to provide controlled exception to the “solitary confinement” policy normally enforced between processes

David Morgan

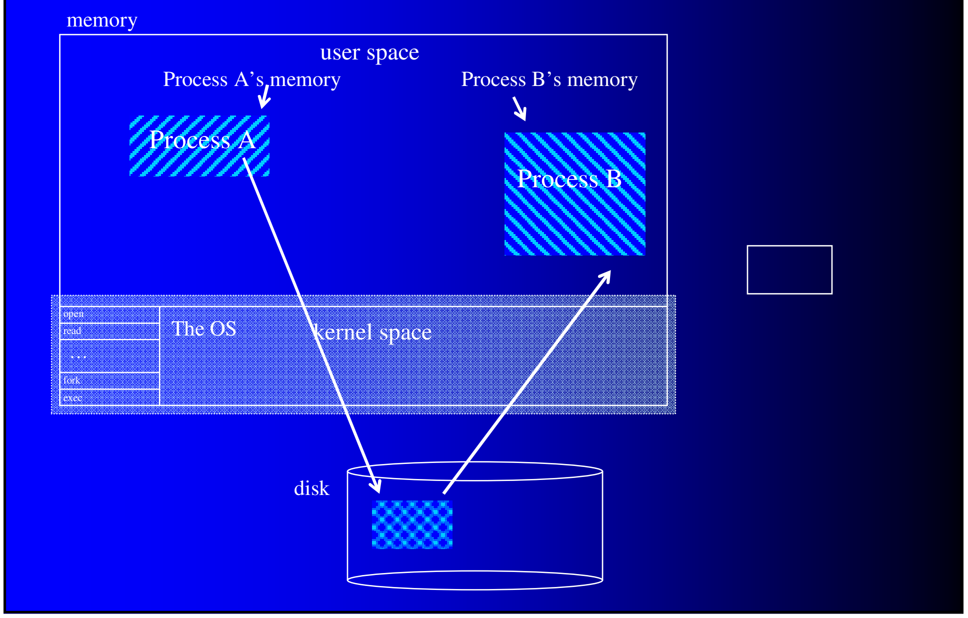


Various kinds of IPC

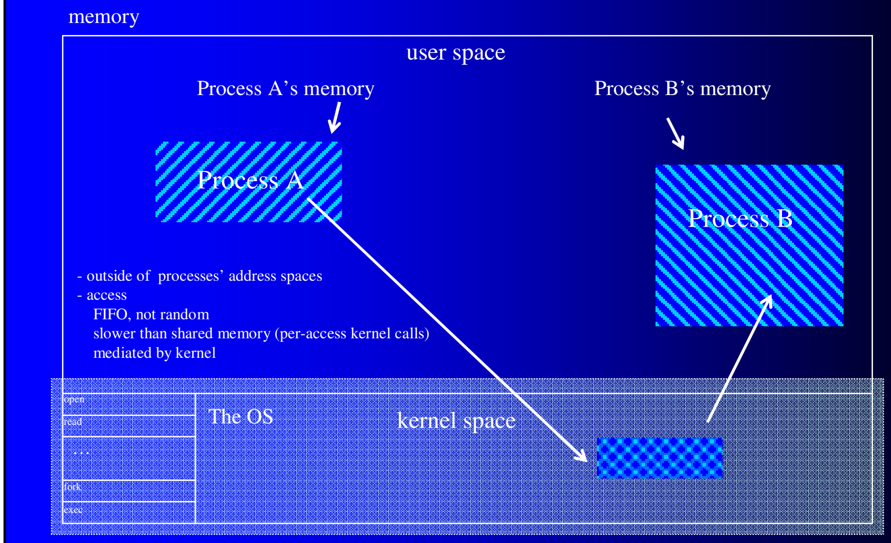
- files
- pipes
- shared memory

- honorable mention: sockets (i.e. network)
  - like IPC: achieves inter-process talk
  - unlike IPC: the processes are (generally) on different machines so this is not purely an OS feature

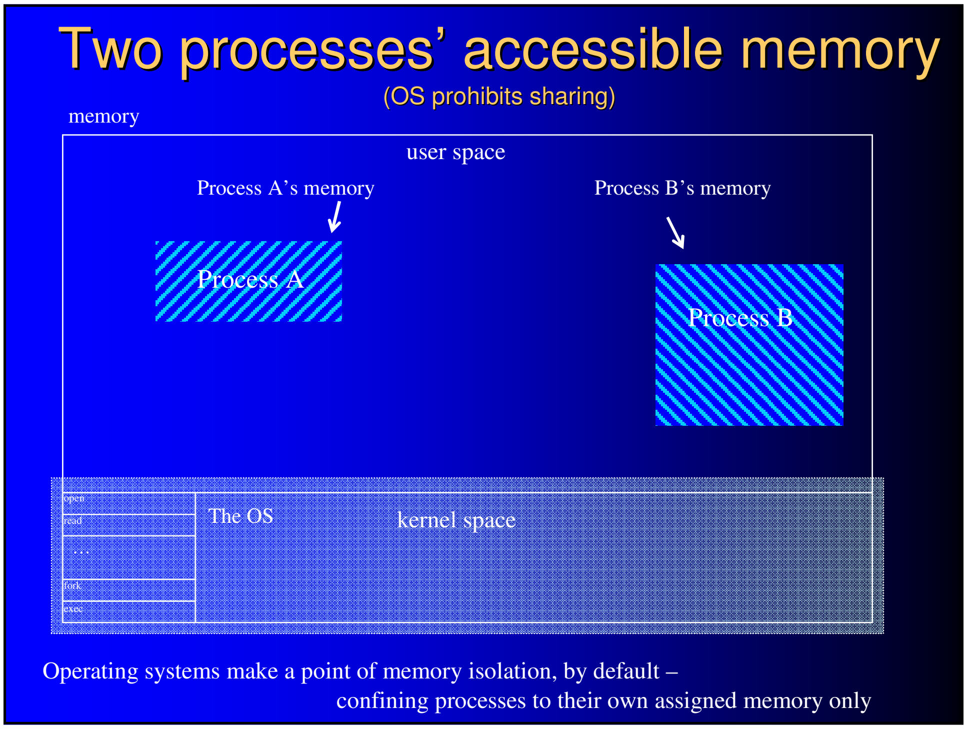
Provide a file



Provide a "pipe" (a.k.a. fifo)



- outside of processes' address spaces
- access
  FIFO, not random
  slower than shared memory (per-access kernel calls)
  mediated by kernel



Two processes’ accessible memory

(OS prohibits sharing)

memory

user space

Process A’s memory

Process B’s memory

Process A

Process B

open

read

. . .

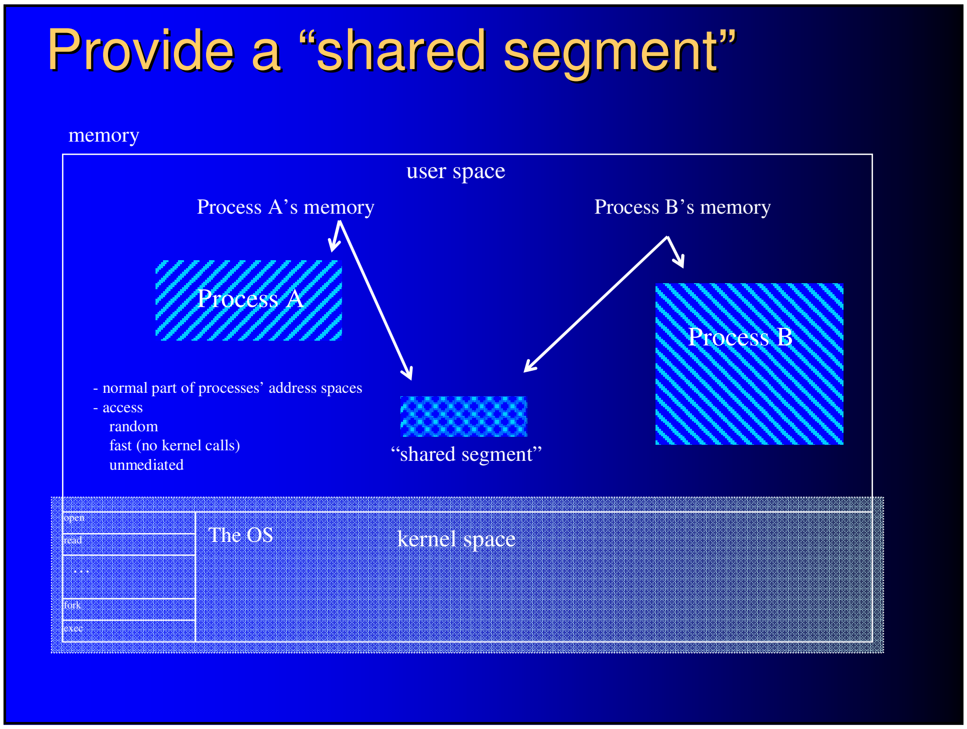
fork

exec

The OS

kernel space

Operating systems make a point of memory isolation, by default –
confining processes to their own assigned memory only



Provide a “shared segment”

memory

user space

Process A’s memory

Process B’s memory

Process A

Process B

- normal part of processes’ address spaces
- access
  random
  fast (no kernel calls)
  unmediated

“shared segment”

open

read

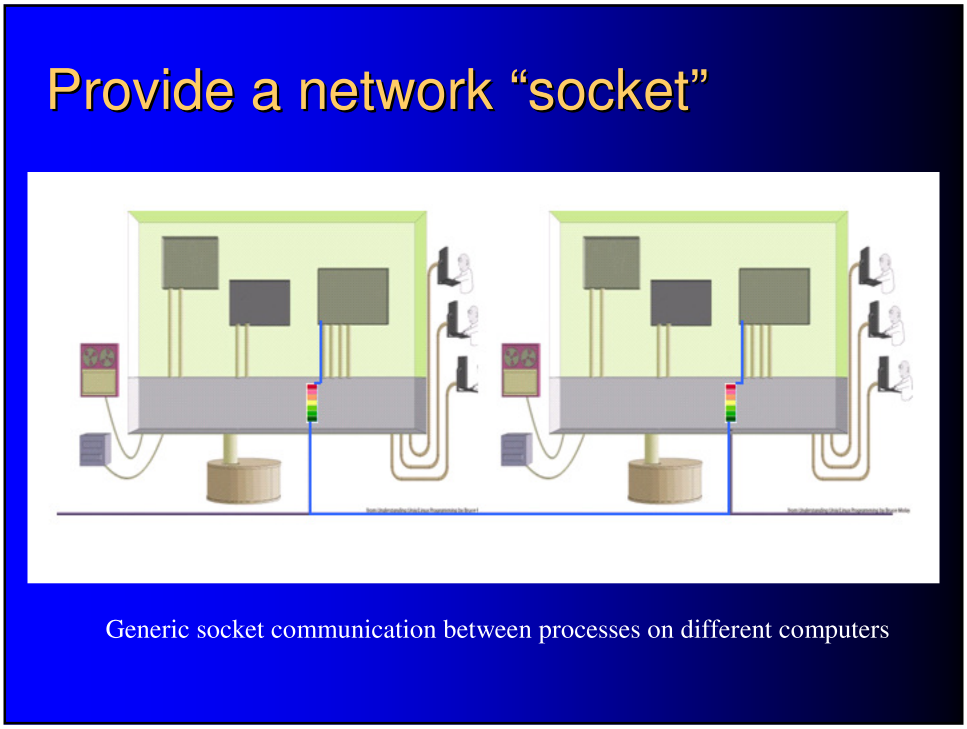
. . .

fork

exec

The OS

kernel space

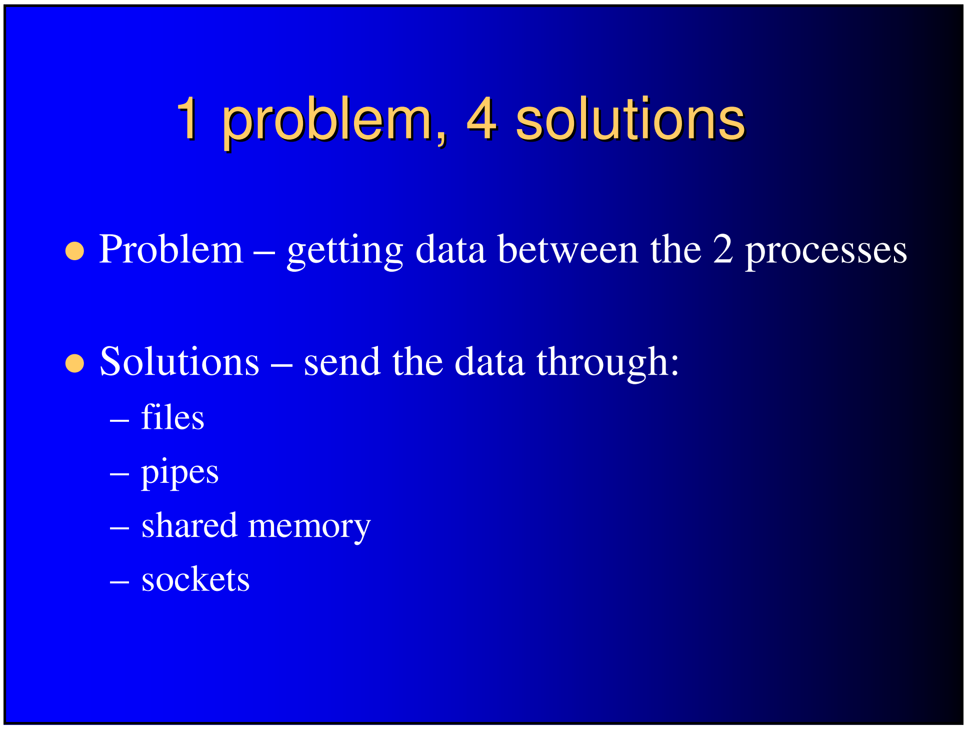


Provide a network "socket"

from Understanding Unix/Linux Programming by Bruce I

from Understanding Unix/Linux Programming by Bruce Molay

Generic socket communication between processes on different computers



1 problem, 4 solutions

- Problem – getting data between the 2 processes
- Solutions – send the data through:
  - files
  - pipes
  - shared memory
  - sockets

Danger – race condition

- multiple processes read/write data
- race condition means
  - result is not deterministic
  - depends on execution order of processes' instructions
- examples
  - server writes, client reads before server finishes
  - client reads, server writes before client finishes
  - client gets neither intended "before" nor "after"

Some comparisons

	range	race condition avoidance responsibility/mediation
file	intra-machine (unless filesystem is net-shared)	application
shared memory	intra-machine	application
pipe	intra -machine	kernel
socket	inter-machine	kernel

Avoiding race conditions
concurrency control possibilities for apps

- locks
  - read lock, “I am reading, *writers should wait* till I’m done but *anyone can read*”
  - write lock, “I am writing, *everyone should wait* till I’m done”
- implementations
  - file locks
  - semaphores

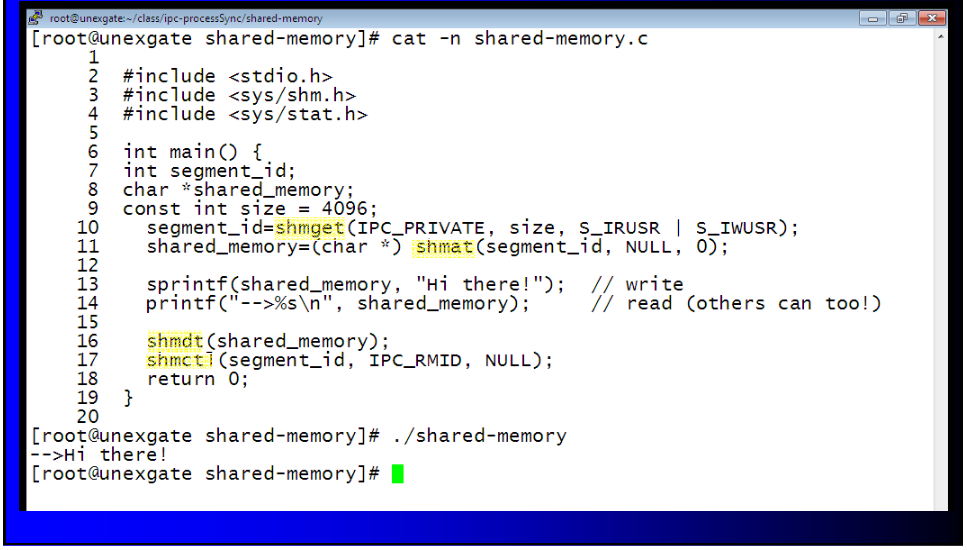
background tutorial
shared memory

- a chunk (“segment”) of memory in user space
  - cf. pipe, a chunk (“buffer”) in *kernel* space
- independent/outside of any process’s memory
- but sharing processes get normal memory pointers to it
  - used equivalently to their own memory
- read/write it with usual pointer-based memory access functions
  - strcpy( )
  - sprintf( )
  - memcpy( )

Managing shared memory segments

action	function call	description
create	shmget( )	allocates a shared memory segment
attach	shmat( )	attach one to calling process's address space (i.e., get pointer)
use	regular pointer-oriented functions (eg strcpy( ))	write/read the shared memory same way as any memory
detach	shmdt( )	detach an attached one from address space
destroy	shmctl( )	mark one for destruction (when detached from all)

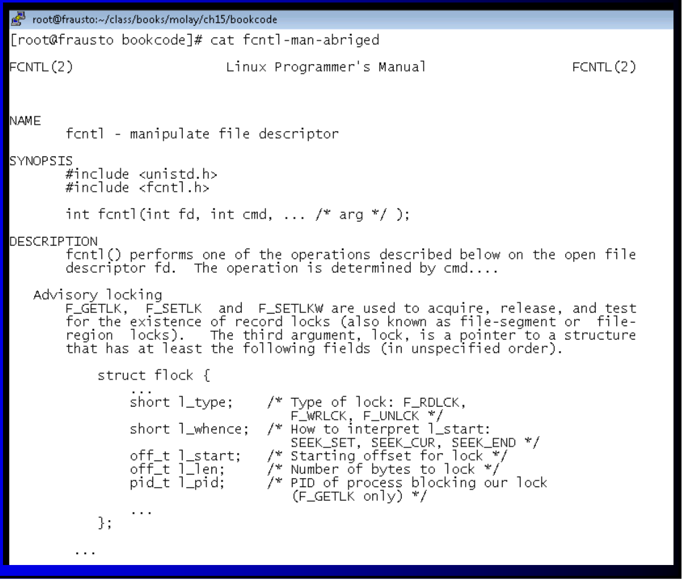
Essentials of exercising shared memory



```
[root@unexgate shared-memory]# cat -n shared-memory.c
     1
     2  #include <stdio.h>
     3  #include <sys/shm.h>
     4  #include <sys/stat.h>
     5
     6  int main() {
     7  int segment_id;
     8  char *shared_memory;
     9  const int size = 4096;
    10    segment_id=shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);
    11    shared_memory=(char *) shmat(segment_id, NULL, 0);
    12
    13    sprintf(shared_memory, "Hi there!");  // write
    14    printf("-->%s\n", shared_memory);     // read (others can too!)
    15
    16    shmdt(shared_memory);
    17    shmctl(segment_id, IPC_RMID, NULL);
    18    return 0;
    19  }
    20
[root@unexgate shared-memory]# ./shared-memory
-->Hi there!
[root@unexgate shared-memory]# 
```

background tutorial

fcntl
file
locking



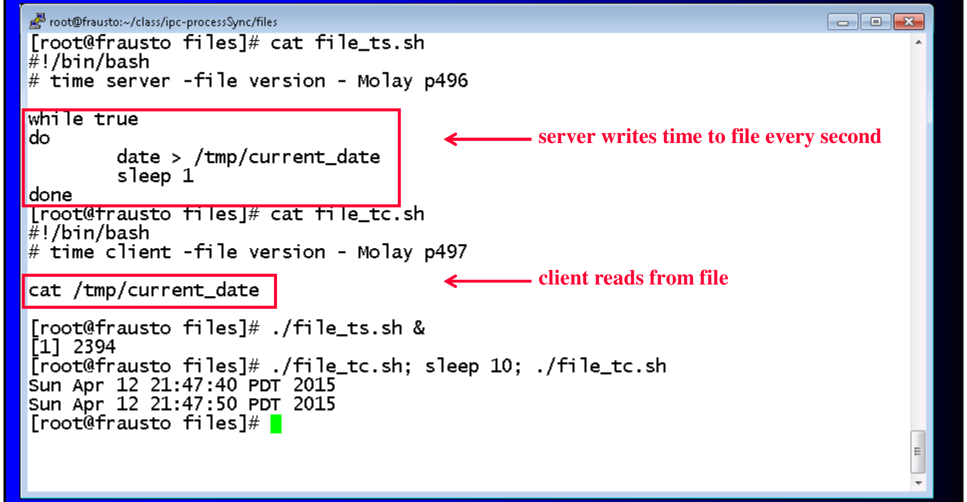
root@frausto:~/class/books/molay/ch15/bookcode

```
[root@frausto bookcode]# cat fcntl-man-abriged
FCNTL(2)                   Linux Programmer's Manual                  FCNTL(2)



NAME
       fcntl - manipulate file descriptor

SYNOPSIS
       #include <unistd.h>
       #include <fcntl.h>

       int fcntl(int fd, int cmd, ... /* arg */ );

DESCRIPTION
       fcntl() performs one of the operations described below on the open file
       descriptor fd.  The operation is determined by cmd....

   Advisory locking
       F_GETLK,  F_SETLK  and  F_SETLKW are used to acquire, release, and test
       for the existence of record locks (also known as file-segment or  file-
       region  locks).   The third argument, lock, is a pointer to a structure
       that has at least the following fields (in unspecified order).

           struct flock {
               ...
               short l_type;    /* Type of lock: F_RDLCK,
                                   F_WRLCK, F_UNLCK */
               short l_whence;  /* How to interpret l_start:
                                   SEEK_SET, SEEK_CUR, SEEK_END */
               off_t l_start;   /* Starting offset for lock */
               off_t l_len;     /* Number of bytes to lock */
               pid_t l_pid;     /* PID of process blocking our lock
                                   (F_GETLK only) */
               ...
           };

         ...
```

A client and server example
- time service -

- a client asks a server for the time
- the server tells the client the time

File-based time service – a script version



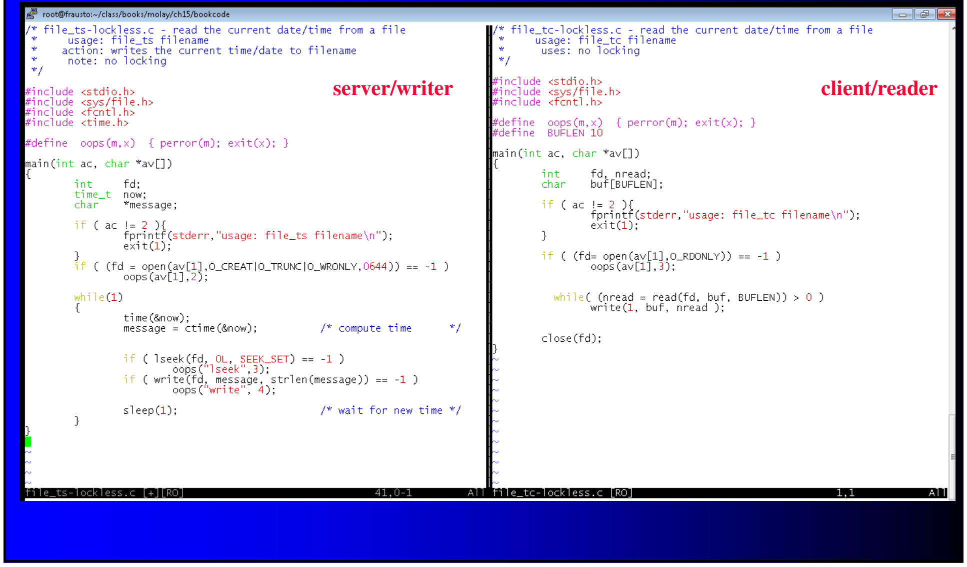
root@frausto:~/class/ipc-processSync/files

```
[root@frausto files]# cat file_ts.sh
#!/bin/bash
# time server -file version - Molay p496

while true
do
        date > /tmp/current_date
        sleep 1
done
[root@frausto files]# cat file_tc.sh
#!/bin/bash
# time client -file version - Molay p497

cat /tmp/current_date

[root@frausto files]# ./file_ts.sh &
[1] 2394
[root@frausto files]# ./file_tc.sh; sleep 10; ./file_tc.sh
Sun Apr 12 21:47:40 PDT 2015
Sun Apr 12 21:47:50 PDT 2015
[root@frausto files]#
```

server writes time to file every second

client reads from file

File-based time service – a C version



root@frausto:~/class/books/molay/ch15/bookcode

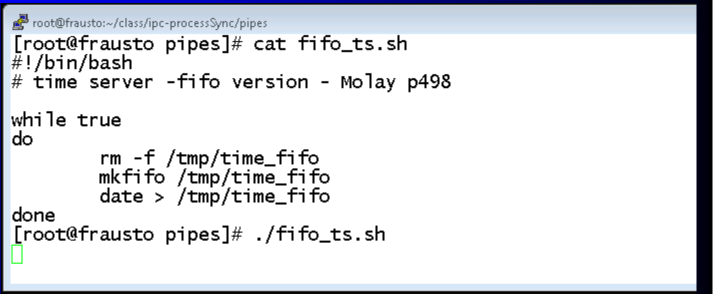
server/writer

```c
/* file_ts-lockless.c - read the current date/time from a file
 *     usage: file_ts filename
 *   action: writes the current time/date to filename
 *     note: no locking
 */

#include <stdio.h>
#include <sys/file.h>
#include <fcntl.h>
#include <time.h>

#define  oops(m,x)  { perror(m); exit(x); }

main(int ac, char *av[])
{
        int     fd;
        time_t  now;
        char    *message;

        if ( ac != 2 ){
                fprintf(stderr,"usage: file_ts filename\n");
                exit(1);
        }
        if ( (fd = open(av[1],O_CREAT|O_TRUNC|O_WRONLY,0644)) == -1 )
                oops(av[1],2);

        while(1)
        {
                time(&now);
                message = ctime(&now);          /* compute time      */


                if ( lseek(fd, 0L, SEEK_SET) == -1 )
                        oops("lseek",3);
                if ( write(fd, message, strlen(message)) == -1 )
                        oops("write", 4);

                sleep(1);                       /* wait for new time */
        }
}

~
~
~
~
file_ts-lockless.c [+][RO]                                   41,0-1        All
```

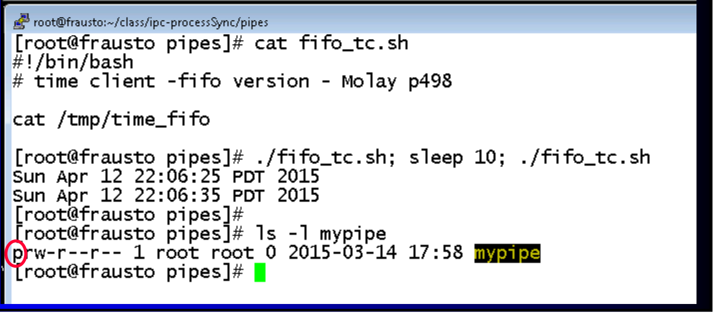
client/reader

```c
/* file_tc-lockless.c - read the current date/time from a file
 *      usage: file_tc filename
 *       uses: no locking
 */

#include <stdio.h>
#include <sys/file.h>
#include <fcntl.h>

#define  oops(m,x)  { perror(m); exit(x); }
#define  BUFLEN 10

main(int ac, char *av[])
{
        int     fd, nread;
        char    buf[BUFLEN];

        if ( ac != 2 ){
                fprintf(stderr,"usage: file_tc filename\n");
                exit(1);
        }

        if ( (fd= open(av[1],O_RDONLY)) == -1 )
                oops(av[1],3);


          while( (nread = read(fd, buf, BUFLEN)) > 0 )
                write(1, buf, nread );


        close(fd);
}
~
~
~
~
~
~
~
~
~
~
~
~
~
file_tc-lockless.c [RO]                                      1,1           All
```

fifo/pipe-based time service



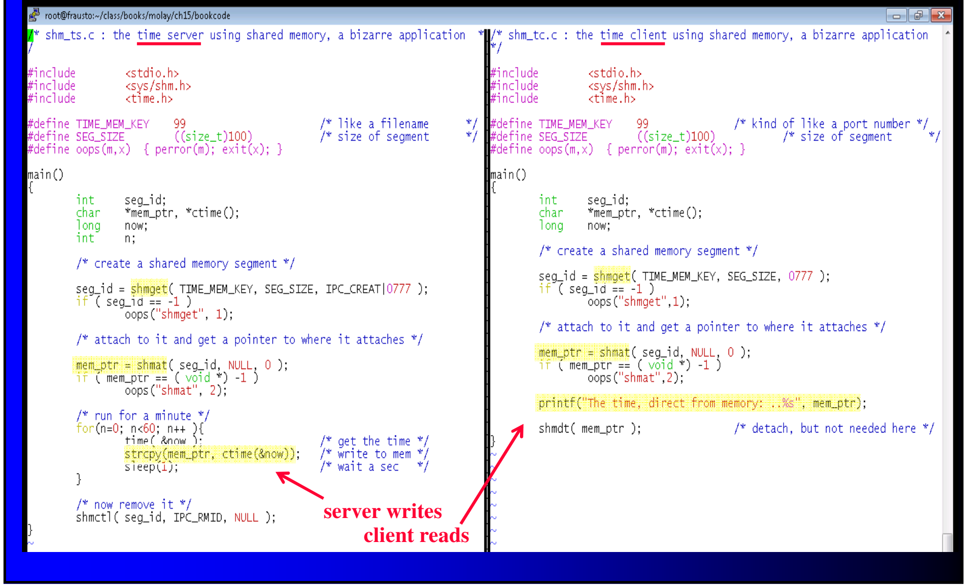
```
[root@frausto pipes]# cat fifo_ts.sh
#!/bin/bash
# time server -fifo version - Molay p498

while true
do
        rm -f /tmp/time_fifo
        mkfifo /tmp/time_fifo
        date > /tmp/time_fifo
done
[root@frausto pipes]# ./fifo_ts.sh
```



```
[root@frausto pipes]# cat fifo_tc.sh
#!/bin/bash
# time client -fifo version - Molay p498

cat /tmp/time_fifo

[root@frausto pipes]# ./fifo_tc.sh; sleep 10; ./fifo_tc.sh
Sun Apr 12 22:06:25 PDT 2015
Sun Apr 12 22:06:35 PDT 2015
[root@frausto pipes]#
[root@frausto pipes]# ls -l mypipe
prw-r--r-- 1 root root 0 2015-03-14 17:58 mypipe
[root@frausto pipes]#
```

- fifo appears in filesystem, but
- fifo is a memory buffer
- fifo content is in memory, not disk

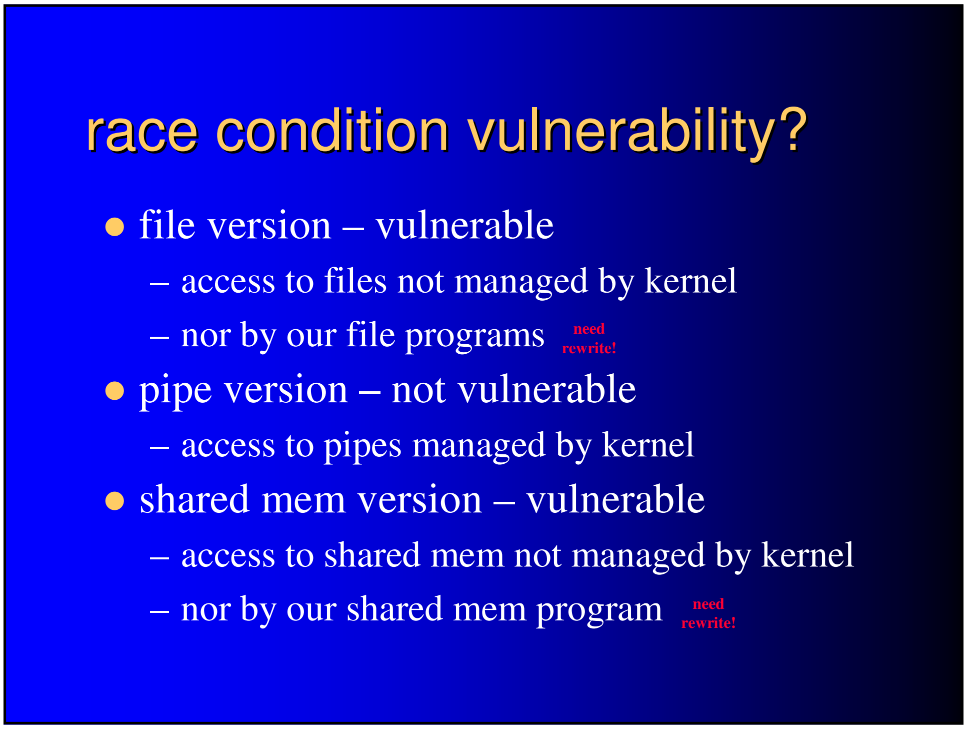
Shared-memory-based time service



```
/* shm_ts.c : the time server using shared memory, a bizarre application  *
/

#include        <stdio.h>
#include        <sys/shm.h>
#include        <time.h>

#define TIME_MEM_KEY    99                      /* like a filename      */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

main()
{
        int     seg_id;
        char    *mem_ptr, *ctime();
        long    now;
        int     n;

        /* create a shared memory segment */

        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, IPC_CREAT|0777 );
        if ( seg_id == -1 )
                oops("shmget", 1);

        /* attach to it and get a pointer to where it attaches */

        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat", 2);

        /* run for a minute */
        for(n=0; n<60; n++ ){
                time( &now );                   /* get the time */
                strcpy(mem_ptr, ctime(&now));   /* write to mem */
                sleep(1);                       /* wait a sec   */
        }

        /* now remove it */
        shmctl( seg_id, IPC_RMID, NULL );
}
~
```

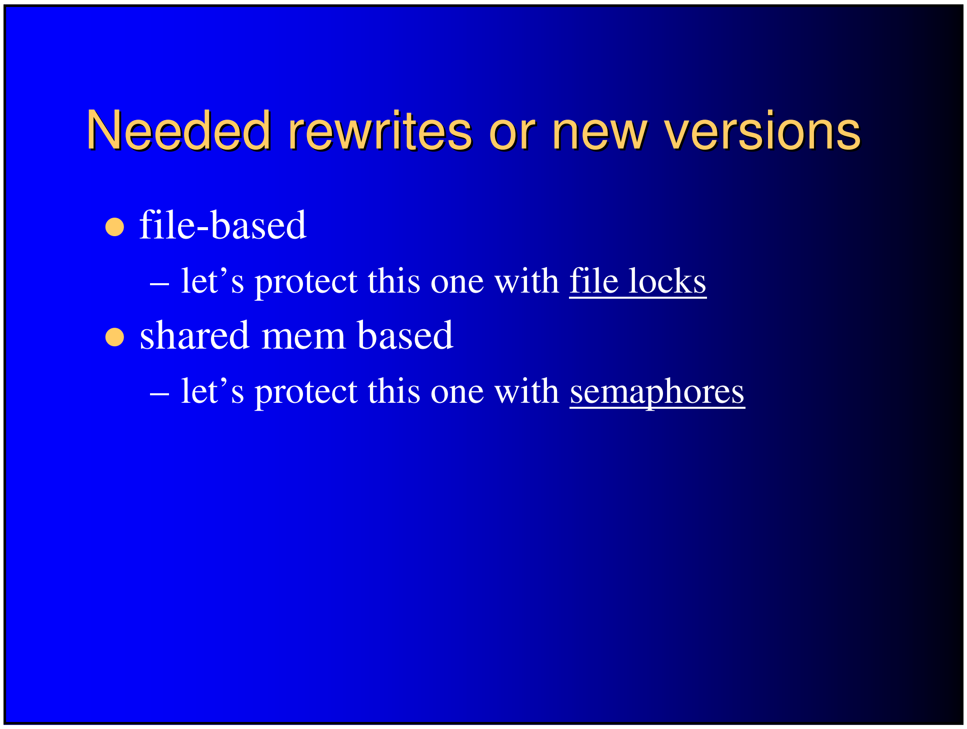
```
/* shm_tc.c : the time client using shared memory, a bizarre application
*/

#include        <stdio.h>
#include        <sys/shm.h>
#include        <time.h>

#define TIME_MEM_KEY    99              /* kind of like a port number */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

main()
{
        int     seg_id;
        char    *mem_ptr, *ctime();
        long    now;

        /* create a shared memory segment */

        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, 0777 );
        if ( seg_id == -1 )
                oops("shmget",1);

        /* attach to it and get a pointer to where it attaches */

        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat",2);

        printf("The time, direct from memory: ..%s", mem_ptr);

        shmdt( mem_ptr );               /* detach, but not needed here */
}
~
~
~
~
~
~
~
~
```

server writes
client reads



race condition vulnerability?

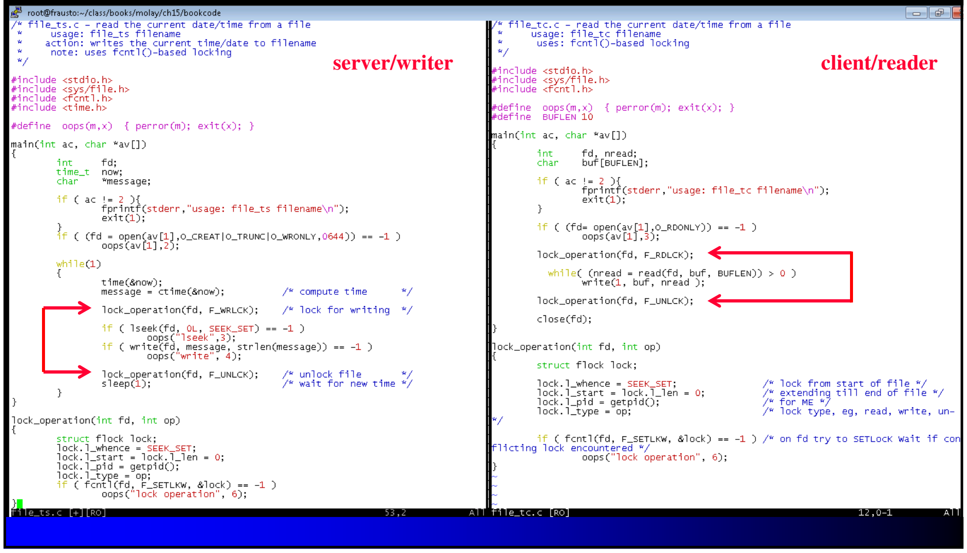
- file version – vulnerable
  - access to files not managed by kernel
  - nor by our file programs need rewrite!
- pipe version – not vulnerable
  - access to pipes managed by kernel
- shared mem version – vulnerable
  - access to shared mem not managed by kernel
  - nor by our shared mem program need rewrite!



Needed rewrites or new versions

- file-based
  - let's protect this one with file locks
- shared mem based
  - let's protect this one with semaphores

File-based version
– protected with file locks



root@frausto:~/class/books/molay/ch15/bookcode

server/writer

```c
/* file_ts.c - read the current date/time from a file
 *     usage: file_ts filename
 *    action: writes the current time/date to filename
 *     note: uses fcntl()-based locking
 */

#include <stdio.h>
#include <sys/file.h>
#include <fcntl.h>
#include <time.h>

#define  oops(m,x)  { perror(m); exit(x); }

main(int ac, char *av[])
{
        int     fd;
        time_t  now;
        char    *message;

        if ( ac != 2 ){
                fprintf(stderr,"usage: file_ts filename\n");
                exit(1);
        }
        if ( (fd = open(av[1],O_CREAT|O_TRUNC|O_WRONLY,0644)) == -1 )
                oops(av[1],2);

        while(1)
        {
                time(&now);
                message = ctime(&now);          /* compute time      */

                lock_operation(fd, F_WRLCK);    /* lock for writing  */

                if ( lseek(fd, 0L, SEEK_SET) == -1 )
                        oops("lseek",3);
                if ( write(fd, message, strlen(message)) == -1 )
                        oops("write", 4);

                lock_operation(fd, F_UNLCK);    /* unlock file       */
                sleep(1);                       /* wait for new time */
        }
}

lock_operation(int fd, int op)
{
        struct flock lock;
        lock.l_whence = SEEK_SET;
        lock.l_start = lock.l_len = 0;
        lock.l_pid = getpid();
        lock.l_type = op;
        if ( fcntl(fd, F_SETLKW, &lock) == -1 )
                oops("lock operation", 6);
}
file_ts.c [+][RO]                                    53,2          All
```

client/reader

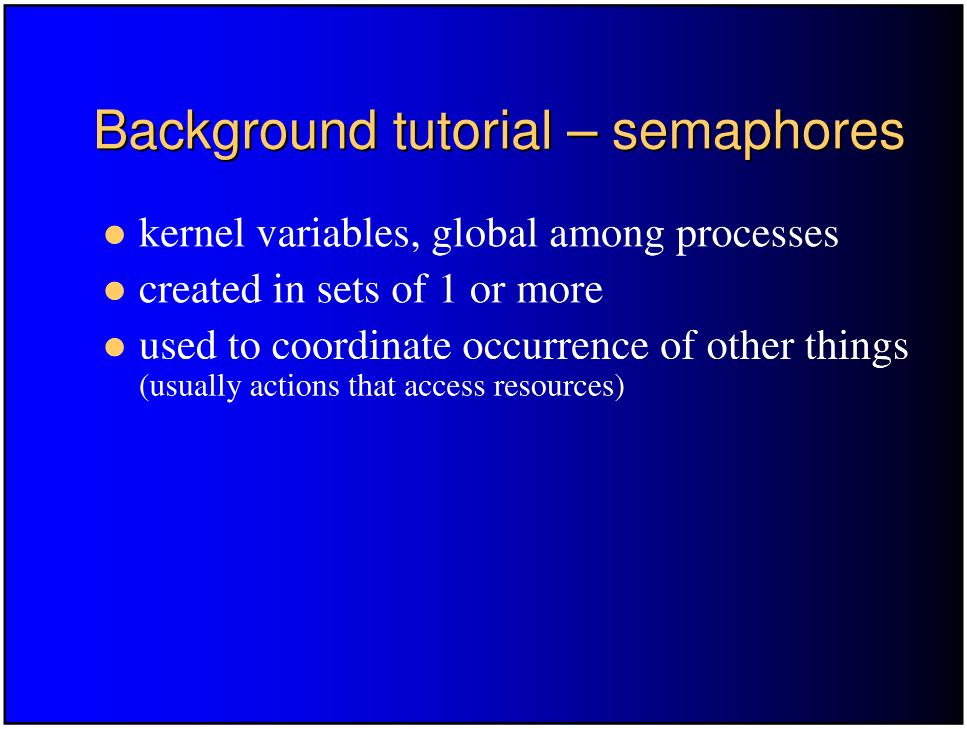
```c
/* file_tc.c - read the current date/time from a file
 *     usage: file_tc filename
 *      uses: fcntl()-based locking
 */

#include <stdio.h>
#include <sys/file.h>
#include <fcntl.h>

#define  oops(m,x)  { perror(m); exit(x); }
#define  BUFLEN 10

main(int ac, char *av[])
{
        int     fd, nread;
        char    buf[BUFLEN];

        if ( ac != 2 ){
                fprintf(stderr,"usage: file_tc filename\n");
                exit(1);
        }

        if ( (fd= open(av[1],O_RDONLY)) == -1 )
                oops(av[1],3);

        lock_operation(fd, F_RDLCK);

          while( (nread = read(fd, buf, BUFLEN)) > 0 )
                write(1, buf, nread );

        lock_operation(fd, F_UNLCK);

        close(fd);
}

lock_operation(int fd, int op)
{
        struct flock lock;

        lock.l_whence = SEEK_SET;                       /* lock from start of file */
        lock.l_start = lock.l_len = 0;                  /* extending till end of file */
        lock.l_pid = getpid();                          /* for ME */
        lock.l_type = op;                               /* lock type, eg, read, write, un-
*/

        if ( fcntl(fd, F_SETLKW, &lock) == -1 ) /* on fd try to SETLock Wait if con
flicting lock encountered */
                oops("lock operation", 6);
}
~
~
~
~
file_tc.c [RO]                                     12,0-1        All
```

Background tutorial – semaphores
One if by land! Two if by sea!!

Listen, my children, and you shall hear
Of the midnight ride of Paul Revere,
On the eighteenth of April, in Seventy-Five:
Hardly a man is now alive
Who remembers that famous day and year.

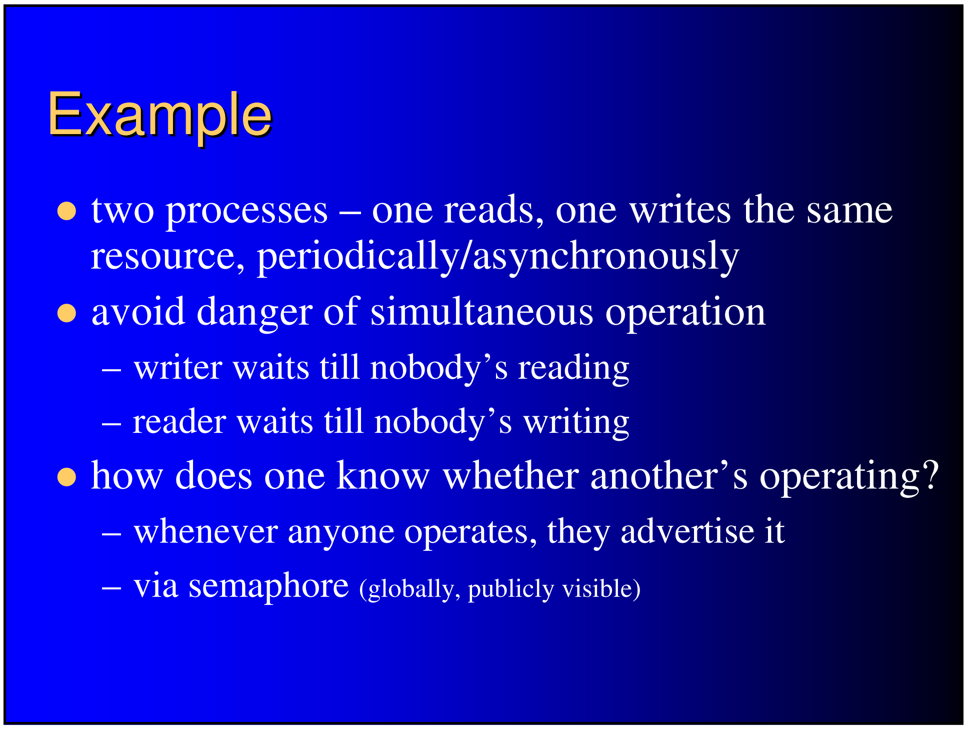
He said to his friend, “If the British march
By land or sea from the town to-night,
Hang a lantern aloft in the belfry-arch
Of the North-Church-tower, as a signal-light,--
One if by land, and two if by sea;
And I on the opposite shore will be,
Ready to ride and spread the alarm
Through every Middlesex village and farm,
For the country-folk to be up and to arm.”

Henry Wadsworth Longfellow



Background tutorial – semaphores

- kernel variables, global among processes
- created in sets of 1 or more
- used to coordinate occurrence of other things (usually actions that access resources)



Example

- two processes – one reads, one writes the same resource, periodically/asynchronously
- avoid danger of simultaneous operation
  - writer waits till nobody’s reading
  - reader waits till nobody’s writing
- how does one know whether another’s operating?
  - whenever anyone operates, they advertise it
  - via semaphore (globally, publicly visible)

mechanism

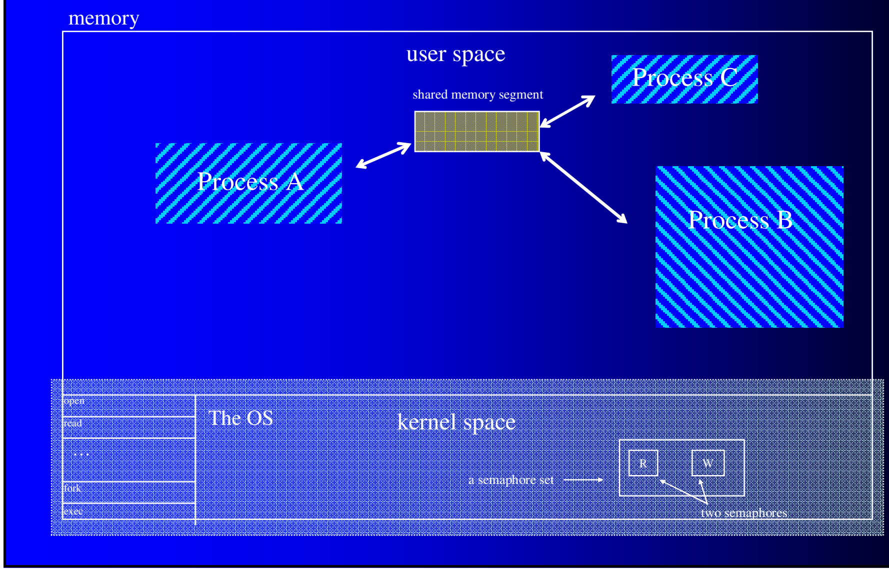
- create a 2-semphore semaphore set
  - utilize one (R) to represent the number of readers
  - and the other (W) to represent the number of writers
- writers must:
  - wait for R to become 0, and increment W
  - proceed to write the resource
  - decrement W

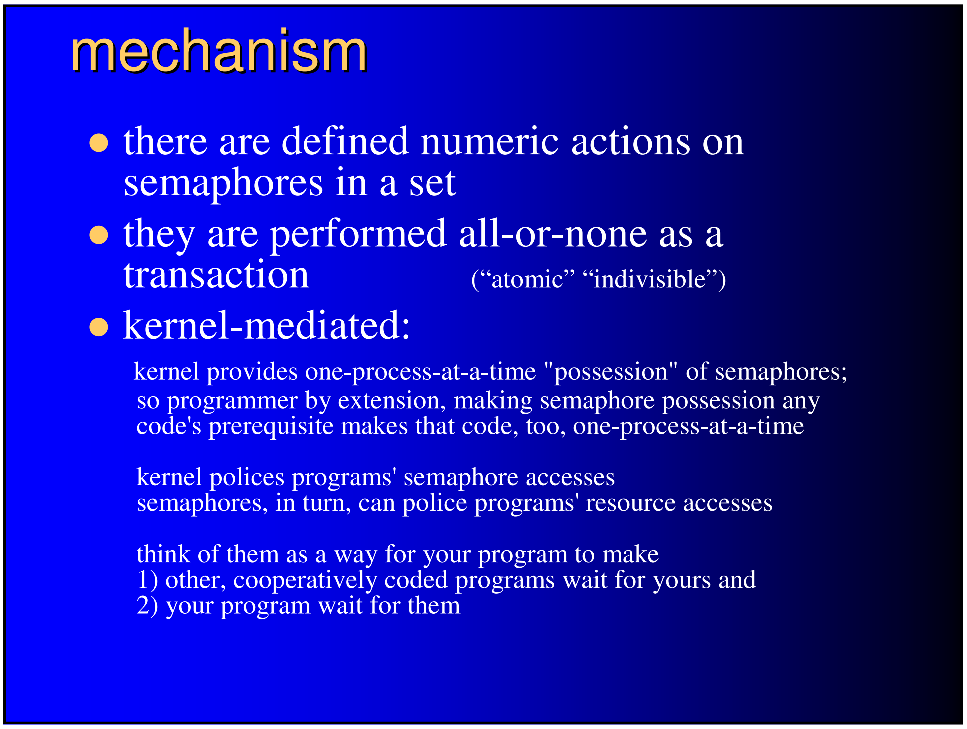
  atomic action set
- readers must:
  - wait for W to become 0, and increment R
  - proceed to read the resource
  - decrement R

  atomic action set

A semaphore set

it can mediate access to shared memory





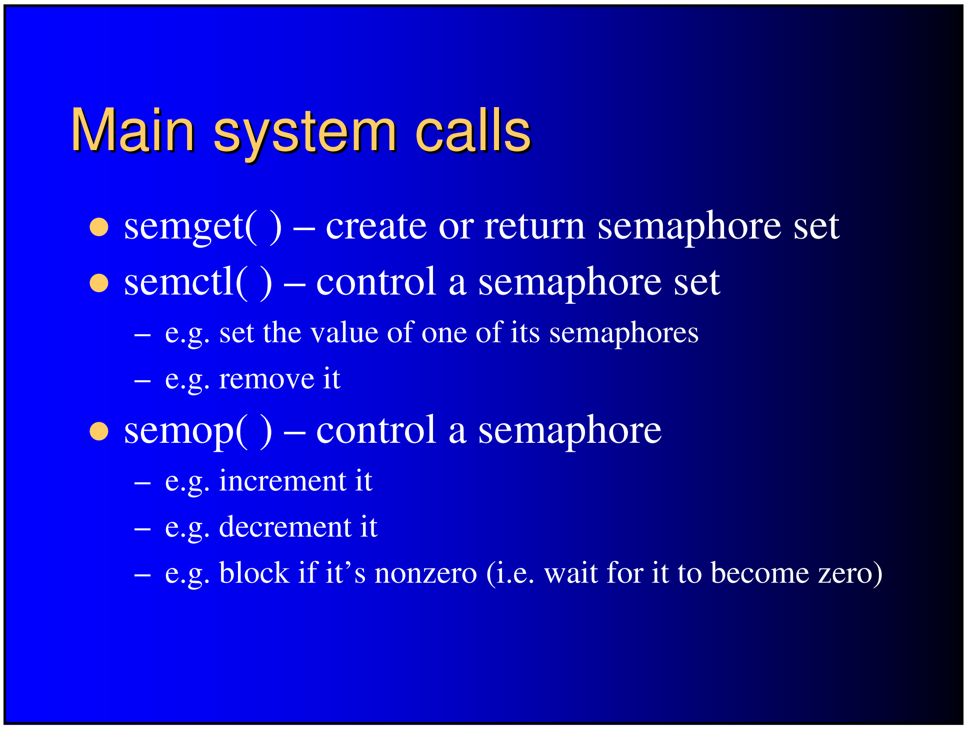
mechanism

- there are defined numeric actions on semaphores in a set
- they are performed all-or-none as a transaction (“atomic” “indivisible”)
- kernel-mediated:

kernel provides one-process-at-a-time "possession" of semaphores;
so programmer by extension, making semaphore possession any
code's prerequisite makes that code, too, one-process-at-a-time

kernel polices programs' semaphore accesses
semaphores, in turn, can police programs' resource accesses

think of them as a way for your program to make
1) other, cooperatively coded programs wait for yours and
2) your program wait for them



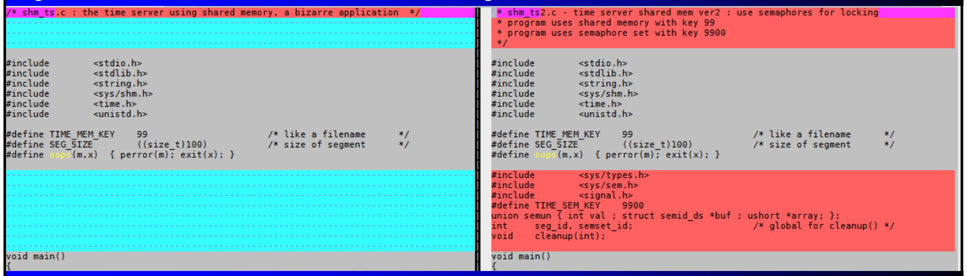
Main system calls

- semget( ) – create or return semaphore set
- semctl( ) – control a semaphore set
  - e.g. set the value of one of its semaphores
  - e.g. remove it
- semop( ) – control a semaphore
  - e.g. increment it
  - e.g. decrement it
  - e.g. block if it’s nonzero (i.e. wait for it to become zero)

Shared-mem-based version

– semaphore-protected server, setup

unprotected version: **protected verion:**



unprotected version:

```
/* shm_ts.c : the time server using shared memory, a bizarre application  */

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <sys/shm.h>
#include        <time.h>
#include        <unistd.h>

#define TIME_MEM_KEY    99                      /* like a filename      */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

void main()
{
```

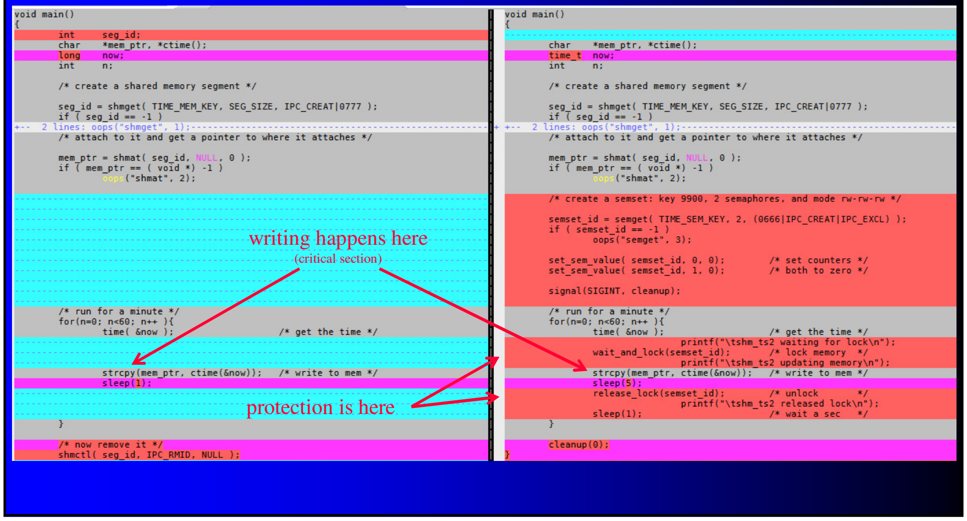
protected verion:

```
 * shm_ts2.c - time server shared mem ver2 : use semaphores for locking
 * program uses shared memory with key 99
 * program uses semaphore set with key 9900
 */

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <sys/shm.h>
#include        <time.h>
#include        <unistd.h>

#define TIME_MEM_KEY    99                      /* like a filename      */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

#include        <sys/types.h>
#include        <sys/sem.h>
#include        <signal.h>
#define TIME_SEM_KEY    9900
union semun { int val ; struct semid_ds *buf ; ushort *array; };
int     seg_id, semset_id;                      /* global for cleanup() */
void    cleanup(int);

void main()
{
```

Shared-mem-based version

– semaphore-protected server, main()

unprotected version: **protected version:**



unprotected version:

```
void main()
{
        int     seg_id;
        char    *mem_ptr, *ctime();
        long    now;
        int     n;

        /* create a shared memory segment */

        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, IPC_CREAT|0777 );
        if ( seg_id == -1 )
+--  2 lines: oops("shmget", 1);----------------------------------------
        /* attach to it and get a pointer to where it attaches */

        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat", 2);

        /* run for a minute */
        for(n=0; n<60; n++ ){
                time( &now );                   /* get the time */

                strcpy(mem_ptr, ctime(&now));   /* write to mem */
                sleep(1);

        }

        /* now remove it */
        shmctl( seg_id, IPC_RMID, NULL );
```

protected version:

```
void main()
{

        char    *mem_ptr, *ctime();
        time_t  now;
        int     n;

        /* create a shared memory segment */

        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, IPC_CREAT|0777 );
        if ( seg_id == -1 )
+--  2 lines: oops("shmget", 1);----------------------------------------
        /* attach to it and get a pointer to where it attaches */

        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat", 2);

        /* create a semset: key 9900, 2 semaphores, and mode rw-rw-rw */

        semset_id = semget( TIME_SEM_KEY, 2, (0666|IPC_CREAT|IPC_EXCL) );
        if ( semset_id == -1 )
                oops("semget", 3);

        set_sem_value( semset_id, 0, 0);        /* set counters */
        set_sem_value( semset_id, 1, 0);        /* both to zero */

        signal(SIGINT, cleanup);

        /* run for a minute */
        for(n=0; n<60; n++ ){
                time( &now );                   /* get the time */
                                printf("\tshm_ts2 waiting for lock\n");
                wait_and_lock(semset_id);       /* lock memory  */
                                printf("\tshm_ts2 updating memory\n");
                strcpy(mem_ptr, ctime(&now));   /* write to mem */
                sleep(5);
                release_lock(semset_id);        /* unlock       */
                                printf("\tshm_ts2 released lock\n");
                sleep(1);                       /* wait a sec   */
        }

        cleanup(0);
}
```

writing happens here
(critical section)

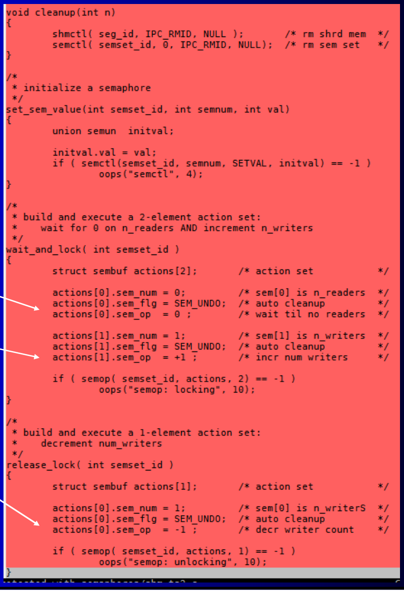
protection is here

Shared-mem-based version
– semaphore-protected server, functions

semop() on a semaphore
 if sem_op
  is zero
   block till semaphore equals 0

  is positive
   increment the semaphore

  is negative
   decrement the semaphore

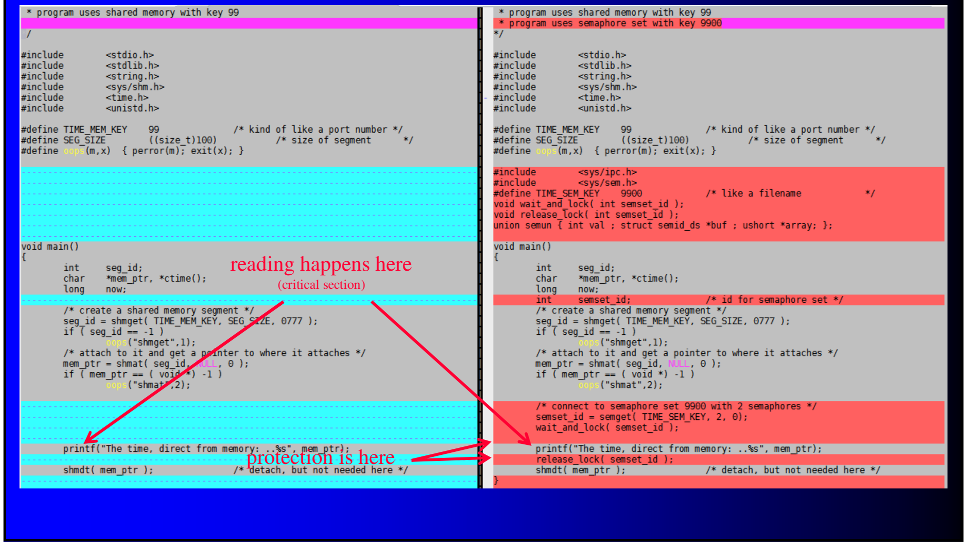


```c
void cleanup(int n)
{
        shmctl( seg_id, IPC_RMID, NULL );       /* rm shrd mem  */
        semctl( semset_id, 0, IPC_RMID, NULL);  /* rm sem set   */
}

/*
 * initialize a semaphore
 */
set_sem_value(int semset_id, int semnum, int val)
{
        union semun  initval;

        initval.val = val;
        if ( semctl(semset_id, semnum, SETVAL, initval) == -1 )
                oops("semctl", 4);
}

/*
 * build and execute a 2-element action set:
 *    wait for 0 on n_readers AND increment n_writers
 */
wait_and_lock( int semset_id )
{
        struct sembuf actions[2];       /* action set           */

        actions[0].sem_num = 0;         /* sem[0] is n_readers  */
        actions[0].sem_flg = SEM_UNDO;  /* auto cleanup         */
        actions[0].sem_op  = 0 ;        /* wait til no readers  */

        actions[1].sem_num = 1;         /* sem[1] is n_writers  */
        actions[1].sem_flg = SEM_UNDO;  /* auto cleanup         */
        actions[1].sem_op  = +1 ;       /* incr num writers     */

        if ( semop( semset_id, actions, 2) == -1 )
                oops("semop: locking", 10);
}

/*
 * build and execute a 1-element action set:
 *    decrement num_writers
 */
release_lock( int semset_id )
{
        struct sembuf actions[1];       /* action set           */

        actions[0].sem_num = 1;         /* sem[0] is n_writerS  */
        actions[0].sem_flg = SEM_UNDO;  /* auto cleanup         */
        actions[0].sem_op  = -1 ;       /* decr writer count    */

        if ( semop( semset_id, actions, 1) == -1 )
                oops("semop: unlocking", 10);
}
```

Shared-mem-based version
– semaphore-protected client

unprotected version:

protected version:



unprotected version:

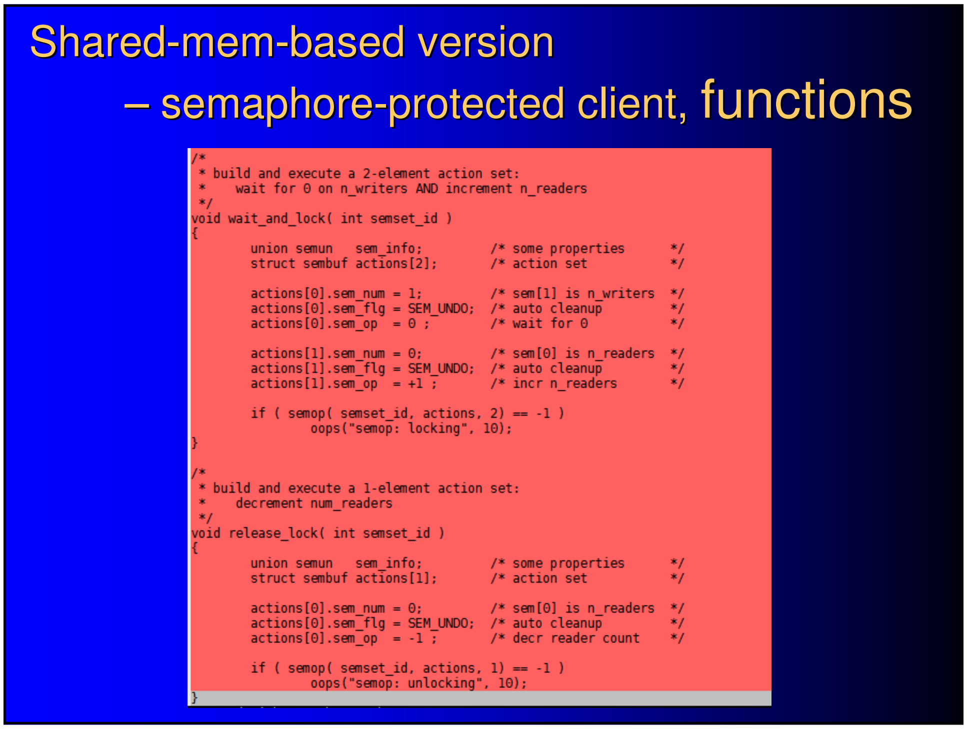
```c
 * program uses shared memory with key 99

*/

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <sys/shm.h>
#include        <time.h>
#include        <unistd.h>

#define TIME_MEM_KEY    99              /* kind of like a port number */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

void main()
{
        int     seg_id;
        char    *mem_ptr, *ctime();
        long    now;

        /* create a shared memory segment */
        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, 0777 );
        if ( seg_id == -1 )
                oops("shmget",1);
        /* attach to it and get a pointer to where it attaches */
        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat",2);

        printf("The time, direct from memory: ..%s", mem_ptr);

        shmdt( mem_ptr );               /* detach, but not needed here */
```

protected version:

```c
 * program uses shared memory with key 99
 * program uses semaphore set with key 9900
*/

#include        <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <sys/shm.h>
#include        <time.h>
#include        <unistd.h>

#define TIME_MEM_KEY    99              /* kind of like a port number */
#define SEG_SIZE        ((size_t)100)           /* size of segment      */
#define oops(m,x)  { perror(m); exit(x); }

#include        <sys/ipc.h>
#include        <sys/sem.h>
#define TIME_SEM_KEY    9900            /* like a filename            */
void wait_and_lock( int semset_id );
void release_lock( int semset_id );
union semun { int val ; struct semid_ds *buf ; ushort *array; };

void main()
{
        int     seg_id;
        char    *mem_ptr, *ctime();
        long    now;
        int     semset_id;              /* id for semaphore set */
        /* create a shared memory segment */
        seg_id = shmget( TIME_MEM_KEY, SEG_SIZE, 0777 );
        if ( seg_id == -1 )
                oops("shmget",1);
        /* attach to it and get a pointer to where it attaches */
        mem_ptr = shmat( seg_id, NULL, 0 );
        if ( mem_ptr == ( void *) -1 )
                oops("shmat",2);

        /* connect to semaphore set 9900 with 2 semaphores */
        semset_id = semget( TIME_SEM_KEY, 2, 0);
        wait_and_lock( semset_id );

        printf("The time, direct from memory: ..%s", mem_ptr);
        release_lock( semset_id );
        shmdt( mem_ptr );               /* detach, but not needed here */
}
```

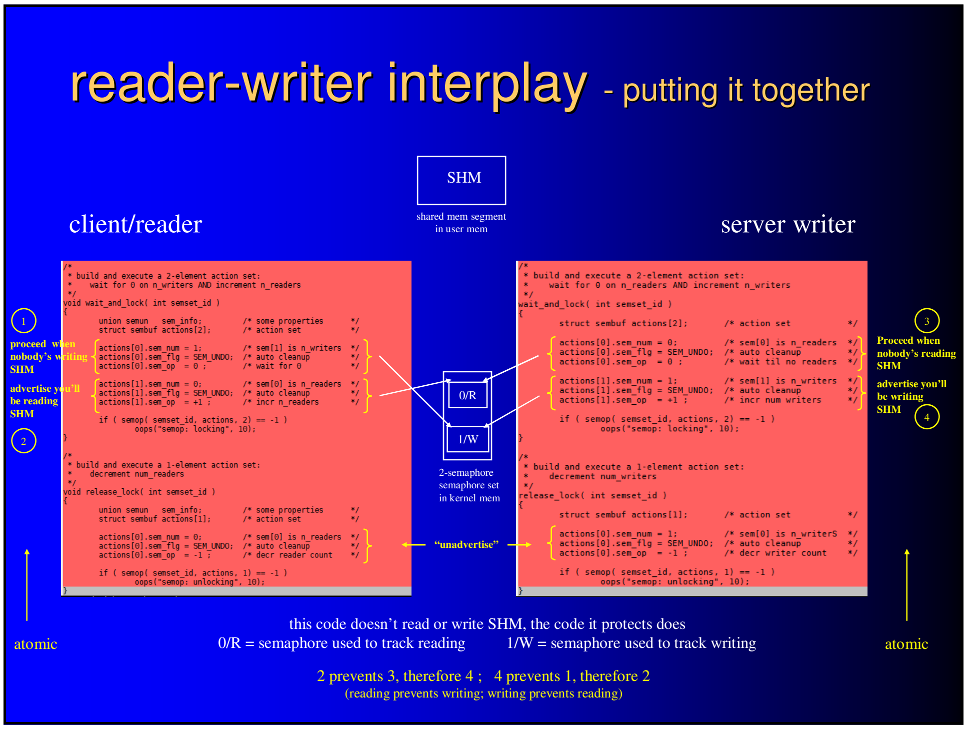
reading happens here
(critical section)

protection is here



Shared-mem-based version
– semaphore-protected client, functions

```
/*
 * build and execute a 2-element action set:
 *    wait for 0 on n_writers AND increment n_readers
 */
void wait_and_lock( int semset_id )
{
        union semun   sem_info;          /* some properties     */
        struct sembuf actions[2];        /* action set          */

        actions[0].sem_num = 1;          /* sem[1] is n_writers */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = 0 ;         /* wait for 0          */

        actions[1].sem_num = 0;          /* sem[0] is n_readers */
        actions[1].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[1].sem_op  = +1 ;        /* incr n_readers      */

        if ( semop( semset_id, actions, 2) == -1 )
                oops("semop: locking", 10);
}

/*
 * build and execute a 1-element action set:
 *    decrement num_readers
 */
void release_lock( int semset_id )
{
        union semun   sem_info;          /* some properties     */
        struct sembuf actions[1];        /* action set          */

        actions[0].sem_num = 0;          /* sem[0] is n_readers */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = -1 ;        /* decr reader count   */

        if ( semop( semset_id, actions, 1) == -1 )
                oops("semop: unlocking", 10);
}
```



reader-writer interplay - putting it together

SHM

shared mem segment
in user mem

client/reader

server writer

1

proceed when nobody's writing SHM

advertise you'll be reading SHM

2

```
/*
 * build and execute a 2-element action set:
 *    wait for 0 on n_writers AND increment n_readers
 */
void wait_and_lock( int semset_id )
{
        union semun   sem_info;          /* some properties     */
        struct sembuf actions[2];        /* action set          */

        actions[0].sem_num = 1;          /* sem[1] is n_writers */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = 0 ;         /* wait for 0          */

        actions[1].sem_num = 0;          /* sem[0] is n_readers */
        actions[1].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[1].sem_op  = +1 ;        /* incr n_readers      */

        if ( semop( semset_id, actions, 2) == -1 )
                oops("semop: locking", 10);
}

/*
 * build and execute a 1-element action set:
 *    decrement num_readers
 */
void release_lock( int semset_id )
{
        union semun   sem_info;          /* some properties     */
        struct sembuf actions[1];        /* action set          */

        actions[0].sem_num = 0;          /* sem[0] is n_readers */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = -1 ;        /* decr reader count   */

        if ( semop( semset_id, actions, 1) == -1 )
                oops("semop: unlocking", 10);
}
```

0/R

1/W

2-semaphore
semaphore set
in kernel mem

"unadvertise"

```
/*
 * build and execute a 2-element action set:
 *    wait for 0 on n_readers AND increment n_writers
 */
wait_and_lock( int semset_id )
{
        struct sembuf actions[2];        /* action set          */

        actions[0].sem_num = 0;          /* sem[0] is n_readers */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = 0 ;         /* wait til no readers */

        actions[1].sem_num = 1;          /* sem[1] is n_writers */
        actions[1].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[1].sem_op  = +1 ;        /* incr num writers    */

        if ( semop( semset_id, actions, 2) == -1 )
                oops("semop: locking", 10);
}

/*
 * build and execute a 1-element action set:
 *    decrement num_writers
 */
release_lock( int semset_id )
{
        struct sembuf actions[1];        /* action set          */

        actions[0].sem_num = 1;          /* sem[0] is n_writerS */
        actions[0].sem_flg = SEM_UNDO;   /* auto cleanup        */
        actions[0].sem_op  = -1 ;        /* decr writer count   */

        if ( semop( semset_id, actions, 1) == -1 )
                oops("semop: unlocking", 10);
}
```

3

Proceed when nobody's reading SHM

advertise you'll be writing SHM

4

atomic

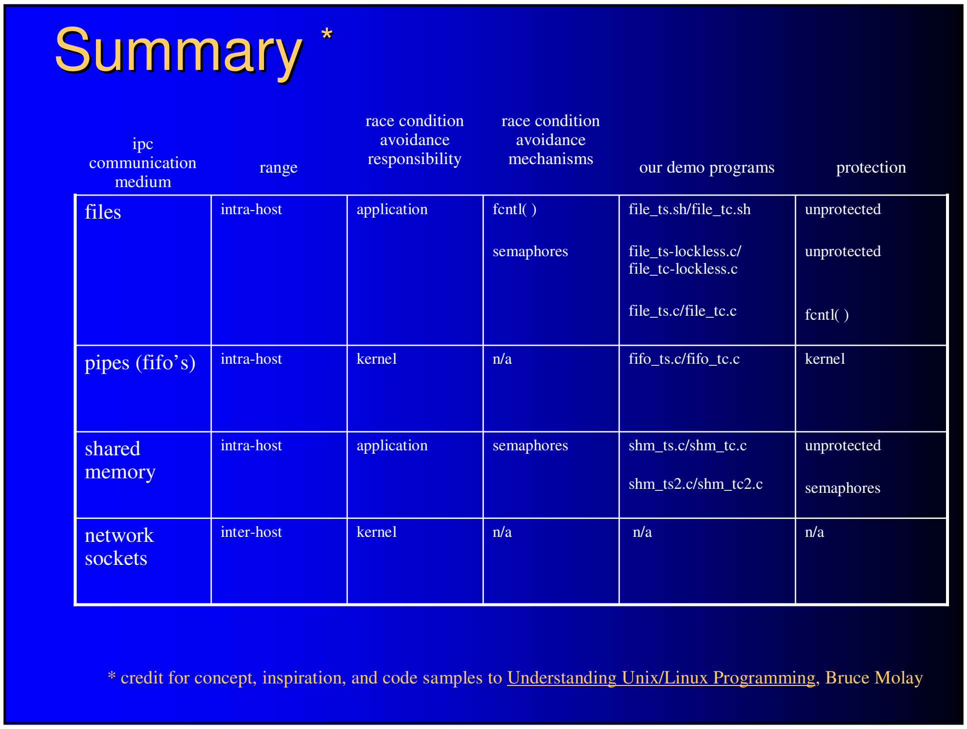
this code doesn't read or write SHM, the code it protects does

0/R = semaphore used to track reading 1/W = semaphore used to track writing

atomic

2 prevents 3, therefore 4 ; 4 prevents 1, therefore 2

(reading prevents writing; writing prevents reading)



Summary *

ipc communication medium	range	race condition avoidance responsibility	race condition avoidance mechanisms	our demo programs	protection
files	intra-host	application	fcntl( ) semaphores	file_ts.sh/file_tc.sh file_ts-lockless.c/ file_tc-lockless.c file_ts.c/file_tc.c	unprotected unprotected fcntl( )
pipes (fifo's)	intra-host	kernel	n/a	fifo_ts.c/fifo_tc.c	kernel
shared memory	intra-host	application	semaphores	shm_ts.c/shm_tc.c shm_ts2.c/shm_tc2.c	unprotected semaphores
network sockets	inter-host	kernel	n/a	n/a	n/a

* credit for concept, inspiration, and code samples to Understanding Unix/Linux Programming, Bruce Molay