```
:::::::::::::: selectdemo.c ::::::::::::::

/* selectdemo.c : watch for input on two devices AND timeout
 *         usage: selectdemo dev1 dev2 timeout
 *        action: reports on input from each file, and
 *                reports timeouts
 */

#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>

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

main(int ac, char *av[])
{
        int    fd1, fd2;           /* the fds to watch      */
        struct timeval timeout;    /* how long to wait      */
        fd_set readfds;            /* watch these for input */
        int    maxfd;             /* max fd plus 1        */
        int    retval;            /* return from select    */

        if ( ac != 4 ){
                fprintf(stderr,"usage: %s file file timeout", *av);
                exit(1);
        }

        /** open files **/
        if ( (fd1 = open(av[1],O_RDONLY)) == -1 )
                oops(av[1], 2);
        if ( (fd2 = open(av[2],O_RDONLY)) == -1 )
                oops(av[2], 3);
        maxfd = 1 + (fd1>fd2?fd1:fd2);

        while(1) {
                /** make a list of file descriptors to watch **/
                FD_ZERO(&readfds);        /* clear all bits  */
                FD_SET(fd1, &readfds);     /* set bit for fd1 */
                FD_SET(fd2, &readfds);     /* set bit for fd2 */

                /** set timeout value **/
                timeout.tv_sec = atoi(av[3]);  /* set seconds */
                timeout.tv_usec = 0;           /* no useconds */

                /** wait for input **/
                retval = select(maxfd,&readfds,NULL,NULL,&timeout);
                if( retval == -1 )
                        oops("select",4);
                if ( retval > 0 ){
                        /** check bits for each fd **/
                        if ( FD_ISSET(fd1, &readfds) )
                                showdata(av[1], fd1);
                        if ( FD_ISSET(fd2, &readfds) )
                                showdata(av[2], fd2);
                }
                else
                        printf("no input after %d seconds\n", atoi(av[3]));
        }
}
showdata(char *fname, int fd)
{
        char buf[BUFSIZ];
        int  n;

        printf("%s: ", fname, n);
        fflush(stdout);
        n = read(fd, buf, BUFSIZ);
        if ( n == -1 )
                oops(fname,5);
        write(1, buf, n);
        write(1, "\n", 1);
}
```

```
:::::::::::::: file_tc.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.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 ::::::::::::::
/* 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);
}
```

```
:::::::::::::: shm_tc2.c ::::::::::::::
/* shm_tc2.c - time client 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       <sys/shm.h>
#include       <time.h>
#include       <sys/types.h>
#include       <sys/ipc.h>
#include       <sys/sem.h>

#define TIME_MEM_KEY   99             /* kind of like a port number */
#define TIME_SEM_KEY   9900           /* like a filename           */
#define SEG_SIZE       ((size_t)100)          /* size of segment     */
#define oops(m,x)  { perror(m); exit(x); }
union semun { int val ; struct semid_ds *buf ; ushort *array; };

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 */
}

/*
 * build and execute a 2-element action set:
 *    wait for 0 on n_writers AND increment n_readers
 */
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
 */
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);
}
```

```
:::::::::::::: shm_ts2.c ::::::::::::::
/* 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       <sys/shm.h>
#include       <time.h>
#include       <sys/types.h>
#include       <sys/sem.h>
#include       <signal.h>

#define TIME_MEM_KEY   99                    /* like a filename      */
#define TIME_SEM_KEY   9900
#define SEG_SIZE       ((size_t)100)         /* size of segment     */
#define oops(m,x)  { perror(m); exit(x); }

union semun { int val ; struct semid_ds *buf ; ushort *array; };
int     seg_id, semset_id;                   /* global for cleanup()       */
void    cleanup(int);

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 )
                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);
}

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);
}
:::::::::::::: fifo_tc ::::::::::::::
#!/bin/sh
# time client using fifos

	cat /tmp/current_date


:::::::::::::: fifo_ts ::::::::::::::
#!/bin/sh
# time server using fifos

	while true
	do
		rm -f /tmp/current_date
		date > /tmp/current_date
		sleep 1
	done
	cat /tmp/current_date
```