Interprocess communication in UNIX

Pipes

- The `pipe()` function:

```
#include <unistd.h>

int pipe(int fd[2]);

    Returns: 0 if OK, –1 on error
```

- After calling `pipe()`:

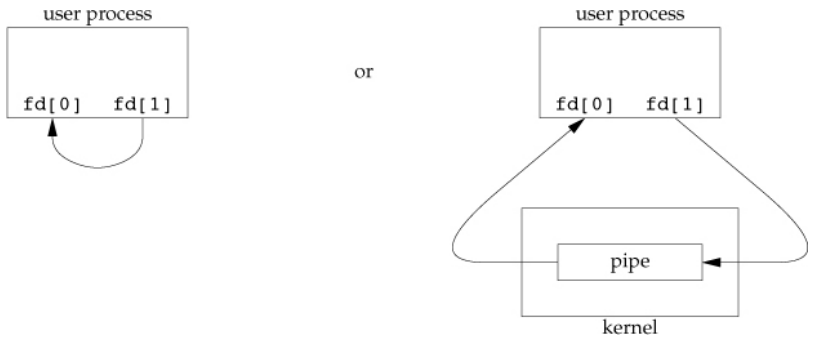

Figure 15.2, APUE

- After calling `pipe()` and then `fork()`:

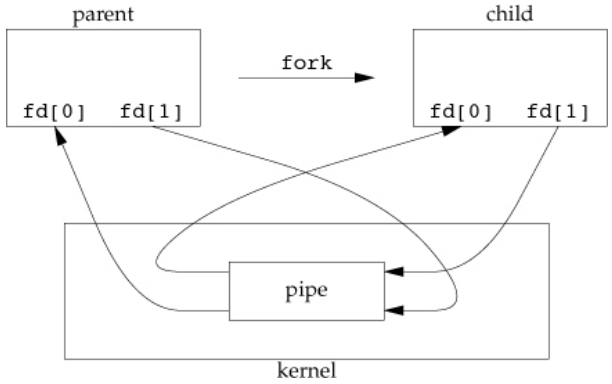

Figure 15.3, APUE

- Example: copying file to a pager program

```
#include "apue.h"
#include <sys/wait.h>

#define DEF_PAGER "/bin/more" /* default pager program */

int main(int argc, char *argv[])
{
    int         n;
    int         fd[2];
    pid_t       pid;
    char        *pager, *argv0;
    char        line[MAXLINE];
    FILE        *fp;

    if (argc != 2)
        err_quit("usage: a.out <pathname>");

    if ((fp = fopen(argv[1], "r")) == NULL)
        err_sys("can't open %s", argv[1]);
    if (pipe(fd) < 0)
        err_sys("pipe error");

    if ((pid = fork()) < 0) {
        err_sys("fork error");

    } else if (pid > 0) {       /* parent */

        close(fd[0]);           /* close read end */

        /* parent copies argv[1] to pipe */
        while (fgets(line, MAXLINE, fp) != NULL) {
            n = strlen(line);
            if (write(fd[1], line, n) != n)
                err_sys("write error to pipe");
        }
        if (ferror(fp))
            err_sys("fgets error");

        close(fd[1]);   /* close write end of pipe for reader */

        if (waitpid(pid, NULL, 0) < 0)
            err_sys("waitpid error");
        exit(0);

    } else {            /* child */

        close(fd[1]);   /* close write end */

        if (fd[0] != STDIN_FILENO) {
            if (dup2(fd[0], STDIN_FILENO) != STDIN_FILENO)
                err_sys("dup2 error to stdin");
```

```
            close(fd[0]);       /* don't need this after dup2 */
        }

        /* get arguments for execl() */
        if ((pager = getenv("PAGER")) == NULL)
            pager = DEF_PAGER;
        if ((argv0 = strrchr(pager, '/')) != NULL)
            argv0++;            /* step past rightmost slash */
        else
            argv0 = pager;      /* no slash in pager */

        if (execl(pager, argv0, (char *)0) < 0)
            err_sys("execl error for %s", pager);
    }
    exit(0);
}
```

XSI IPC

They share common naming and interface scheme:

- XSI Message queues

```
int msgget(key_t key, int flag);
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
int msgsnd(int msqid, const void *ptr, size_t nbytes, int flag);
ssize_t msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag);
```

- XSI Semaphores

```
int semget(key_t key, int nsems, int flag);
int semctl(int semid, int semnum, int cmd, ... /* union semun arg */ );
int semop(int semid, struct sembuf semoparray[], size_t nops);
```

- XSI Shared memory

```
int shmget(key_t key, size_t size, int flag);
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
void *shmat(int shmid, const void *addr, int flag);
int shmdt(const void *addr);
```

And they all suck…

1. Hard to clean-up because there is no reference counting
    - pipes get automatically removed when last process terminates
    - data left in a FIFO is removed when last process terminates

2. Hard to use
    - complex and inelegant interfaces that don't fit into UNIX file system paradigm
    - stupid naming scheme: IPC identifiers, keys, and *project IDs – are you serious?*

They have been widely used for lack of alternatives. Fortunately we do have alternatives these days:

- Instead of XSI message queues, use:
    - UNIX domain sockets
    - POSIX message queues (still not widely available, so not covered in APUE; see `man 7 mq_overview`)
- Instead of XSI semaphores, use:
    - POSIX semaphores
- Instead of XSI shared memory, use:
    - memory mapping using `mmap()`

Memory-mapped I/O

- `mmap()` function:

```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flag, int fd, off_t off);

    Returns: starting address of mapped region if OK, MAP_FAILED on error
```

- Example of a memory-mapped file:

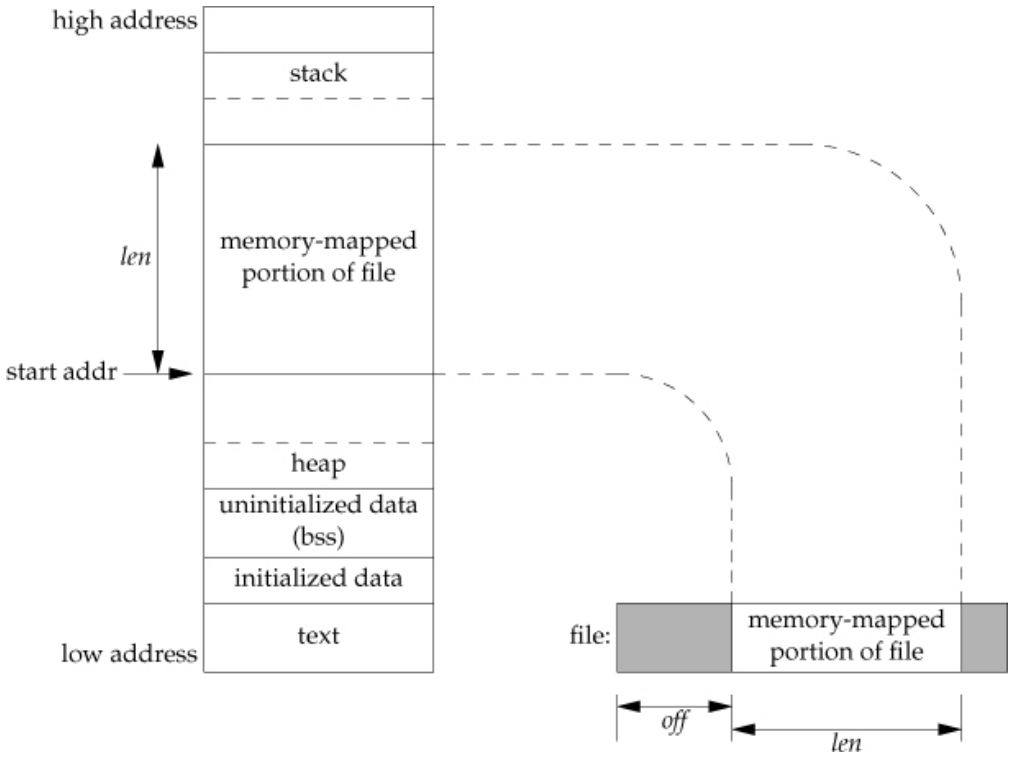

Figure 14.26, APUE

- Memory mapping `/dev/zero` for shared memory (Figure 15.33, APUE):

```
#include "apue.h"
#include <fcntl.h>
#include <sys/mman.h>

#define NLOOPS          1000
#define SIZE            sizeof(long)    /* size of shared memory area */

static int
update(long *ptr)
{
        return((*ptr)++);       /* return value before increment */
}

int
main(void)
{
        int             fd, i, counter;
        pid_t   pid;
        void    *area;

        if ((fd = open("/dev/zero", O_RDWR)) < 0)
                err_sys("open error");
        if ((area = mmap(0, SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) ==
                err_sys("mmap error");
        close(fd);              /* can close /dev/zero now that it's mapped */

        TELL_WAIT();

        if ((pid = fork()) < 0) {
                err_sys("fork error");
        } else if (pid > 0) {                   /* parent */
                for (i = 0; i < NLOOPS; i += 2) {
                        if ((counter = update((long *)area)) != i)
                                err_quit("parent: expected %d, got %d", i, coun

                        TELL_CHILD(pid);
                        WAIT_CHILD();
                }
        } else {                                                /* child */
                for (i = 1; i < NLOOPS + 1; i += 2) {
                        WAIT_PARENT();

                        if ((counter = update((long *)area)) != i)
                                err_quit("child: expected %d, got %d", i, counte

                        TELL_PARENT(getppid());
                }
        }

        exit(0);
}
```

- Anonymous memory mapping

  Same as `/dev/zero` mapping, but more portable and more convenient.

  Change Figure 15.33 as follows:

  1. remove `open("/dev/zero", ...)` and `close(fd)`
  2. change `mmap` call to:

```
if ((area = mmap(0, SIZE, PROT_READ | PROT_WRITE,
      MAP_ANON | MAP_SHARED, -1, 0)) == MAP_FAILED)
```

POSIX Semaphores

- What is semaphore?
  - Binary vs. Counting semaphores
- Creating, opening, closing, and removing **named** POSIX semaphores:

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ... /* mode_t mode,
                unsigned int value  */ );
        Returns: Pointer to semaphore if OK, SEM_FAILED on error

int sem_close(sem_t *sem);
        Returns: 0 if OK, –1 on error

int sem_unlink(const char *name);
        Returns: 0 if OK, –1 on error
```

- Initializing and destroying **unnamed** POSIX semaphores:

```
#include <semaphore.h>

int sem_init(sem_t *sem, int pshared, unsigned int value);
        Returns: 0 if OK, –1 on error

int sem_destroy(sem_t *sem);
        Returns: 0 if OK, –1 on error
```

- Using POSIX semaphores:

  Decrement the value of semaphores:

```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
        Both return: 0 if OK, −1 on error
```

Decrement with bounded waiting:

```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
                  const struct timespec *restrict tsptr);
        Returns: 0 if OK, −1 on error
```

Increment the value of semaphores:

```
#include <semaphore.h>

int sem_post(sem_t *sem);
        Returns: 0 if OK, −1 on error
```

Last updated: 2014–02–11