XSI IPC

System V IPC

XSI IPC

- Message queue
- Shared memory
- Semaphores

Why?

The interprocess communication tools considered earlier do not allow to organize an exchange between processes running at different time

The identifiers and keys

- Each XSI IPC corresponds to an identifier in the system

- The external identifier is a special key that is used in system calls

Key

<sys/types.h>

key_t - signed long integer

The kernel converts the key to an internal identifier

Creating the key

```
#include <sys/ipc.h>
key_t ftok(const char *path, int id);
```

1. Possible conflicts with other developers
2. The id parameter allows you to specify multiple ip associated with a single path in the FS

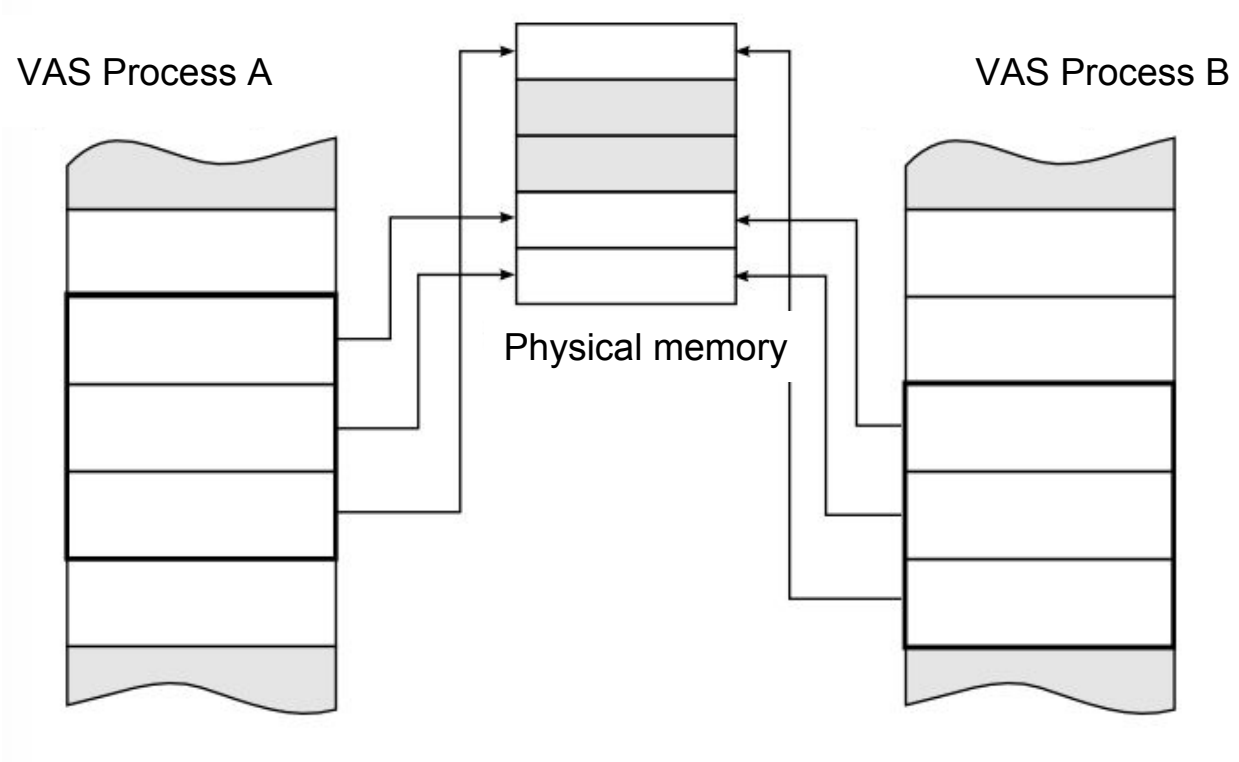
The file is needed only for key generation!

Shared memory

Each process exists in its own isolated address space

Shared memory overcomes this limitation

Shared memory



The behavior when fork, exec

What happens to shared memory with **fork** and **exec**?

The behavior when fork, exec

fork:

Shared memory is inherited

exec:

Shared memory is excluded

Creating shared memory

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

int shmget(key_t key, size_t size, int shmflg);
```

1 argument: key_t key

1. The result of calling the function ftok()

2. The special value **IPC_PRIVATE**, which results in an attempt to create a new shared memory with a key whose value does not match any key of existing IPC objects, and which cannot be obtained using the **ftok()** function in any combination of its parameters

2 argument: size_t size

The size argument specifies the size of the memory segment to be created.

If a memory segment with the specified key already exists and the specified size does not match the size of the existing segment, an error is thrown.

3 argument: int shmflg

It is important only when creating new shared memory and determines the access rights to this memory for different users

IPC_CREAT — if the SHM for the specified key does not exist, it must be created;

IPC_EXCL — used in conjunction with IPC_CREATE flag. At their joint usage and existence of SHM with the specified key access to SHM is not made and the error is thrown

Attach to the process

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

void *shmat(int shmid, const void *shmaddr, int
shmflg);
```

Returns the address of the memory segment in the address space of the process in case of success, -1 in case of error

Argument *shmaddr

The *shmaddr argument is a pointer to the memory area to which you want to attach the new segment. A NULL value can be entered instead of a specific memory pointer, in which case the memory will be attached to the first free memory in the process address space.

Argument shmflg

The **shmflg** argument can take many different meanings, but within the framework of the information exchange between processes we are interested in — we will be interested in only two values

0 — read and write to a memory segment

SHM_RDONLY — only read from the memory segment

Separation of SHM from the process

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

int shmdt(char *shmaddr);
```

Returns 0 if successful, -1 if successful errors

The removal of the SHM

No reference counter! Manual deletion.

#include <sys/ipc.h>

#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);

Example:

shmctl(sd, IPC_RMID, NULL);

Manual deletion

Interpreter commands

ipcs -a view all tools IPC

ipcrm - remove the IPC from the system

Example

Shared memory usage between processes existing at different times in the system

shm-idle.c

Example

The use of shared memory between parent and child process

shm-fork.c

The problems with SHM

- Data races (Data Race, Race Conditions)
- The problem of readers and writers

The problem of readers and writers

We have a 64-bit machine and we want to copy a 128-bit number

C code: **n1 = n2**

mov <first 64-bits>

mov <second 64-bits>

Parallel reader can get "broken" data

Another example

```
struct User {
  int id,
  int role
}
```

struct User user1, user2

user1 = user2

Exercise 3a

Create parallel version of vectors multiplication. Vector size is divided by the number of processes. Each child process computes multiplication of its own part of vector and stores result in shared memory.

Questions:

1. Compare execution time of sequential and parallel versions.
2. Compare execution time of parallel version with different numbers of processes.

Template: vector-multiplication.c

Semaphores

Semaphores are defined by some integer nonnegative type, and **down** and **up** operations are possible on them.

down - compares the value of a variable with zero, if the value is greater than zero, then it is reduced by one and control is returned to the program, otherwise the process is blocked until the semaphore value is greater than zero, that is, until another process performs the up operation.

up - increases the semaphore value by one.

Readers & writers

1. A semaphore is created with an initial value of one.

2. Before writing to shared memory, the process performs a down operation on the semaphore, and then if the semaphore is 1, the process can safely write, and if the value is 0, it means that some process is already working with memory and the process will be put into a standby state.

3. After the memory operations are complete, the process performs an up operation on the semaphore.

Semaphores XSI IPC

1. **up**(S,n) — increase the semaphore value by n.

2.**down**(S,n) — Process lock if semaphore value s<n, then S=S-n.

3.**z(**S) — the process is blocked until the value of semaphore S becomes 0.

Create a semaphore

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

int semget(key_t key, int nsems, int semflg);
```

Arguments

The **nsems** argument specifies the number of semaphores in the array to be created, or the number of semaphores in an already created array, if accessed by a known key.

semflg similar **shmflg**

When you create the semaphore, always set to 0

Perform up, down, z operations

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

int semop(int semid, struct sembuf *sops, size_t nsops);
```

Returns 0 if successful, -1 if successful errors

Аргумент *sops

```
struct sembuf {
        short sem_num; /* The number of the semaphore in the array, semaphores start at 0 */
        short sem_op; /* Operation */
        short sem_flg; /* Flags to perform operations */
};
```

sem_op

The values of the **sem_op** structure element are defined as follows:

–To perform the up(S,n) operation, the value must be **n**
–To perform the down(S,n) operation, the value must be **-n**
–To perform operation z (S), the value must be **0**

The example of the operation

```
semd = semget(IPC_PRIVATE, 1, 0666 | IPC_CREAT);
struct sembuf mybuf;

// set the initial value of the semaphore to 1

mybuf.sem_num = 0;
mybuf.sem_op = 1;
mybuf.sem_flg = 0;

semop(semd,&mybuf,1);
```

Deletion of semaphores

Similar to shared memory:

semctl(semid, **0**, IPC_RMID, NULL);

Message queue

Can be thought of as a mailbox in which different processes send messages by marking them with some type.

Messages from this mailbox are fetched by processes of their own choice in the following ways:

1. Regardless of the message type, in order of FIFO (First in First Out).
2. Within a certain type in FIFO order.
3. The oldest message with the specific type is selected first.

Usage

This type of IPC is considered obsolete and is not recommended for use in real systems.

Disadvantages

- XSI IPC bound to the kernel, not the process
- Do not have a reference counter
- Do not have a name in the file system (there is a need to use special system calls)

Exercise 3b

Create parallel program for computing Pi number. Each child process appends its partial result to the same single shared memory var - sizeof(double). Access to this var should be synchronized by semaphore.

Questions:

1. What happens when we don’t synchronize access to shared var?
2. Compare execution time of template program and current program.

Template: pi-shared-memory.c