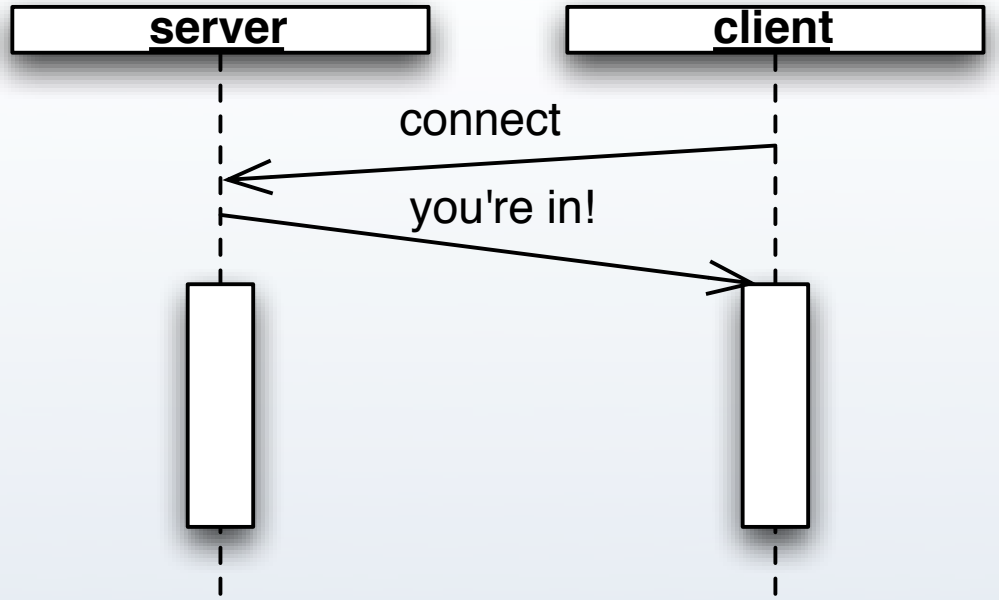
Introduction to multiprogramming

Ștefan Gună

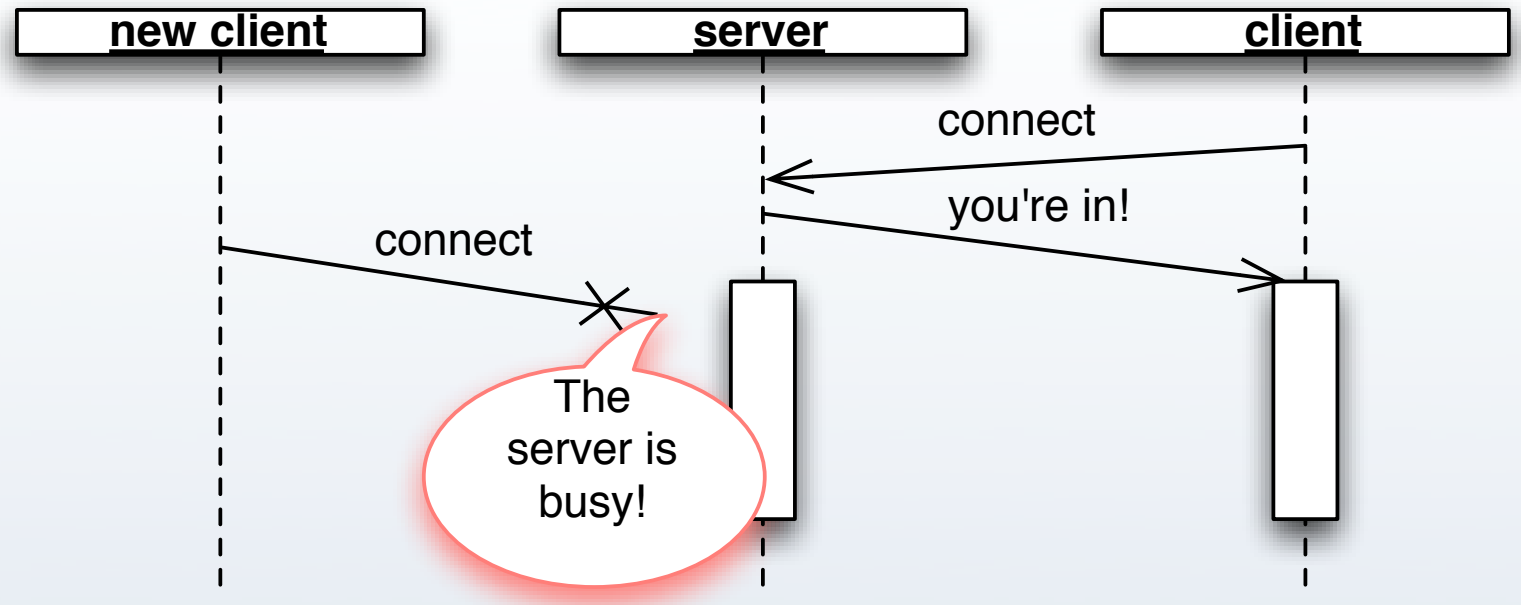
guna@disi.unitn.it

Processes

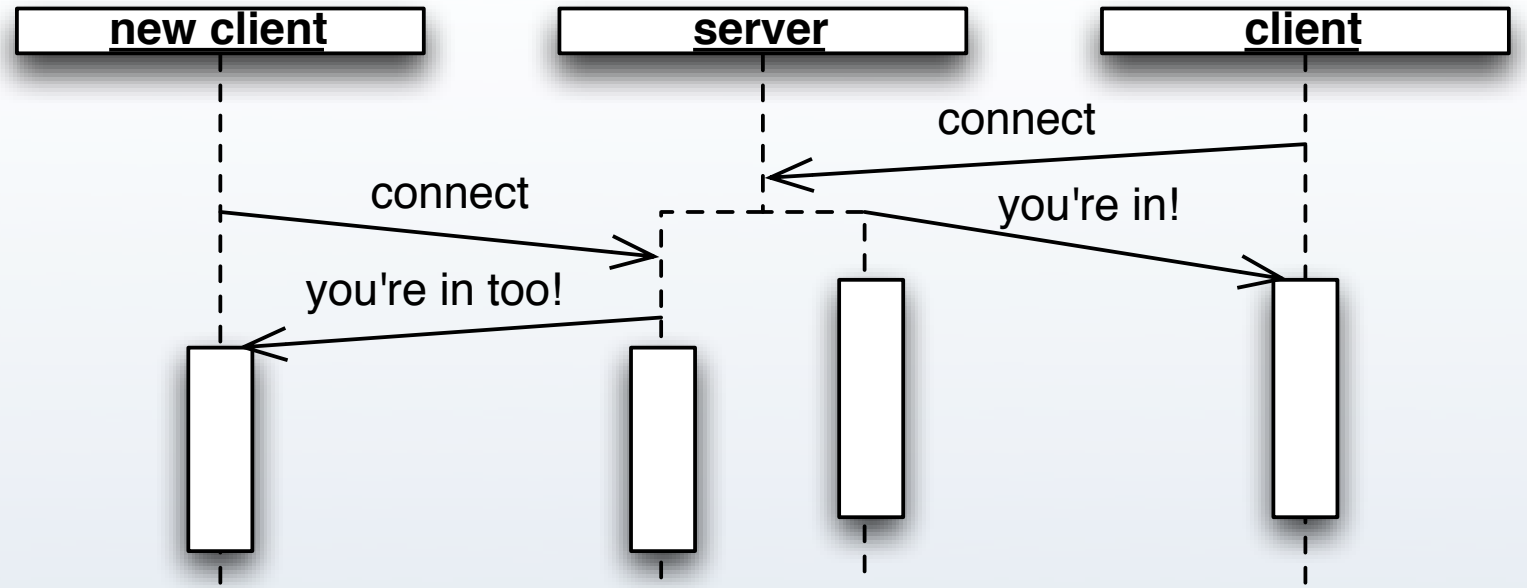
The problem



The problem

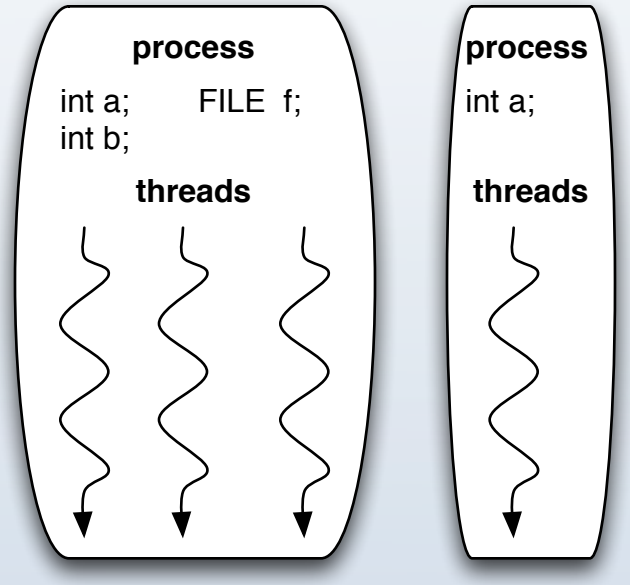

new client
server
client
connect
you're in!
connect
The server is busy!

The problem

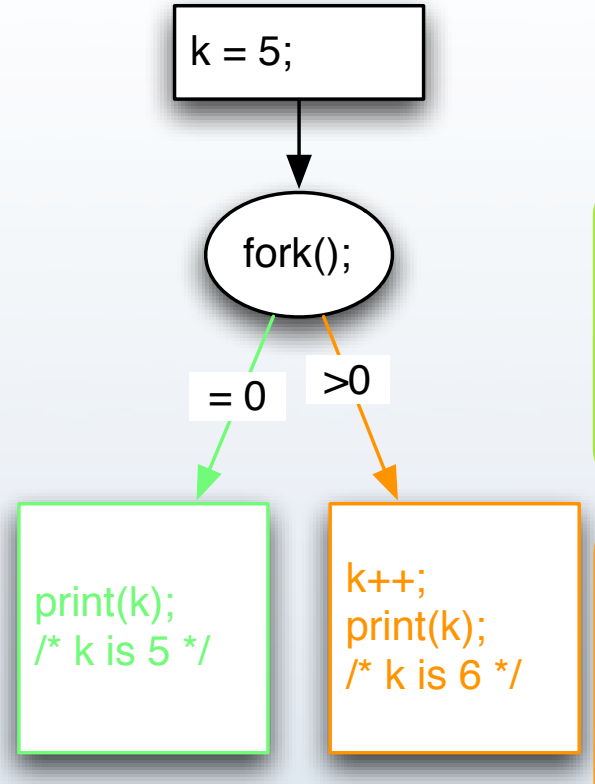

new client
server
client
connect
connect
you're in!
you're in too!

Processes and threads

- parallel execution can be obtained using *processes* and *threads*
- processes have their own resources and are identified by their PID
- processes are creating by spawning child processes from other processes
- a process has one or more execution *threads*
- threads of the same process share resources



Spawning using `fork`



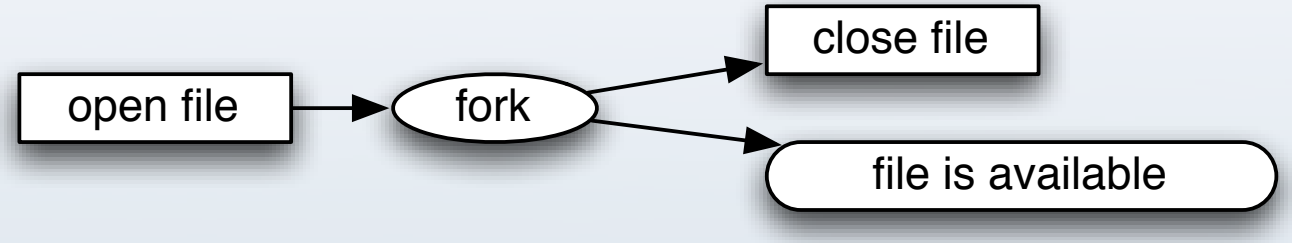

```
int k = 5;
pid_t pid = fork();
```

```
if (pid == 0) {
  sleep(5);
  printf("Child: k=%d\n", k);
}
```

```
if (pid > 0) {
  k++;
  wait(NULL);
  printf("Parent: k=%d!\n", k);
}
```

What you should know

- resources are duplicated when processes are spawned
- any resource before the fork is available afterwards
- resource changes are not visible in other processes
- example:



- resources = variable, files, sockets

Inter-process communication

Introduction

- concurrent processes often must:
  - share data
  - synchronize their execution
- synchronization is often required for regulating access to shared data

Introduction

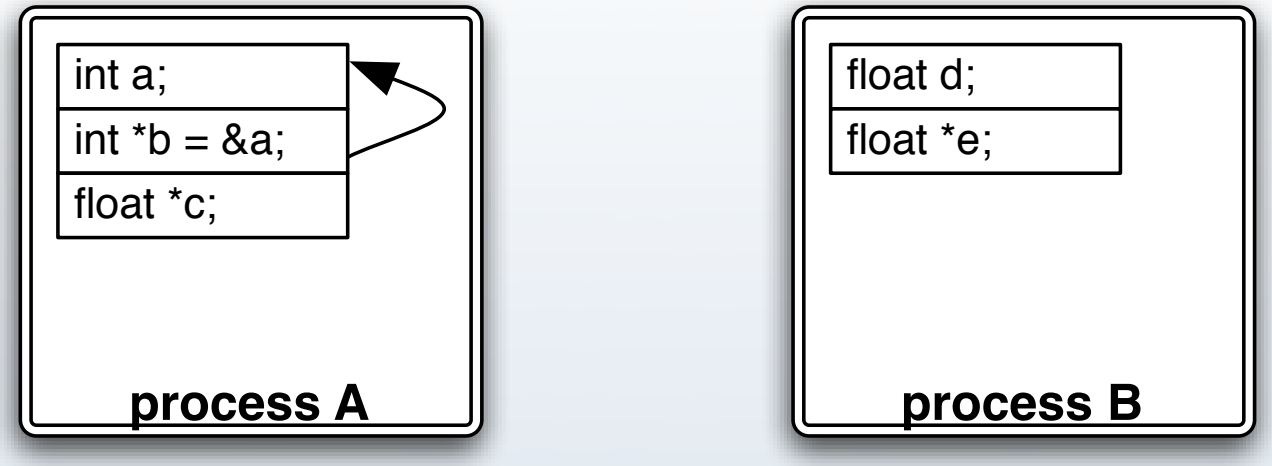
Data sharing

- pipes
- named pipes
- message queues
- shared memory
- sockets

Synchronization

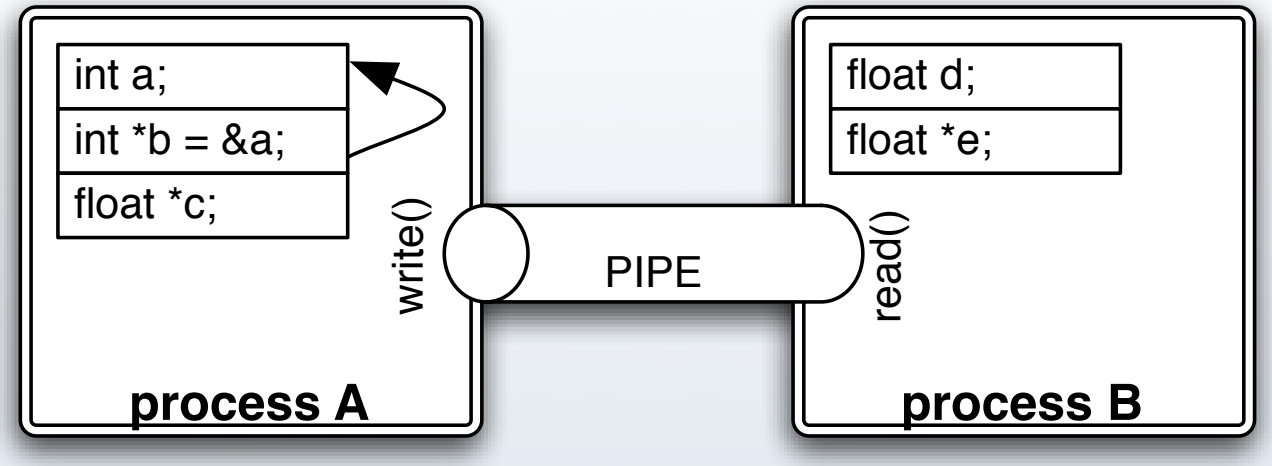
- signals
- semaphores
- monitors

shared memory



- how can processes A and B share information?

shared memory



- a file or a pipe could be used
- but you need to **write** or **read**

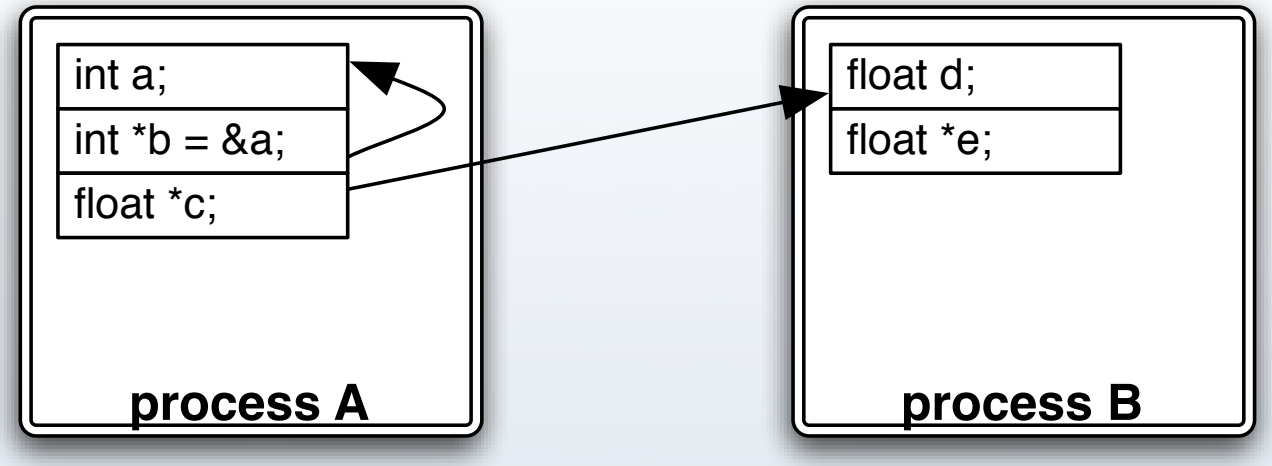
shared memory

Execute…

```
(*c)++;
```

… with the effect

```
d++;
```



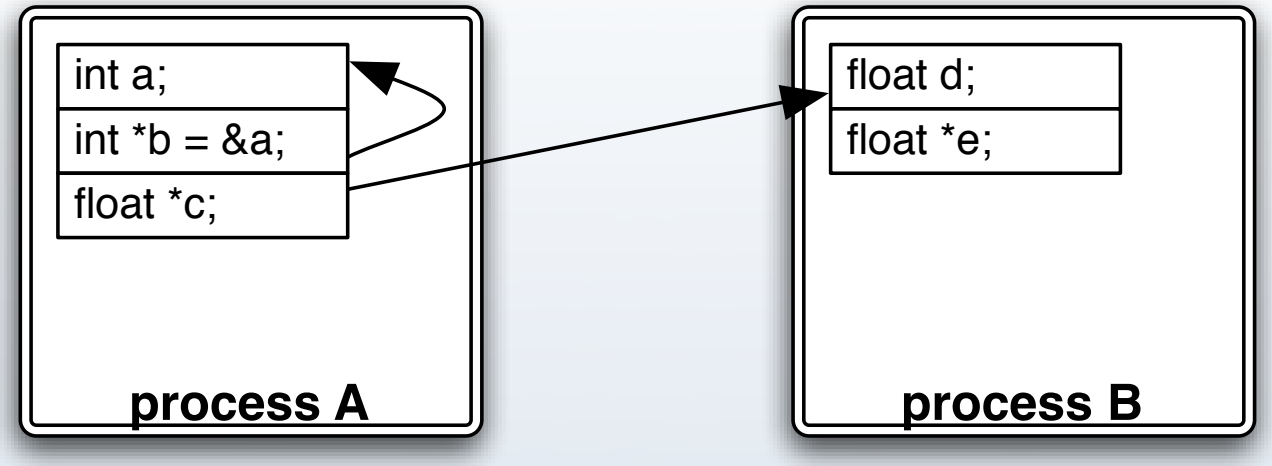
shared memory

Execute…

```
(*c)++;
```

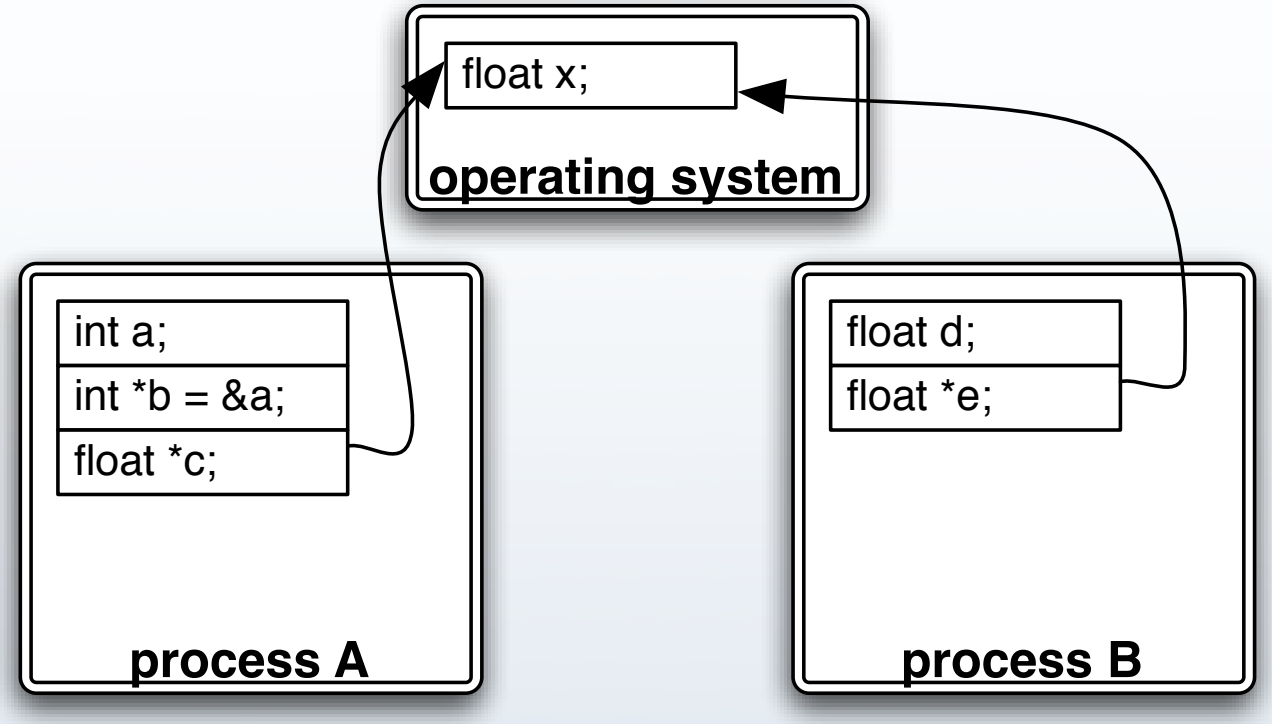
… with the effect

```
d++;
```



NOT ALLOWED!

shared memory



- shared memory block allocated by the operating system
- mapped into the address space of a process

shared memory

```
key_t key;
int shm_id;
float *c;

key = ftok("~/some_file", 7);

shm_id = shmget(key, sizeof(float),
                0666 | IPC_CREATE);

c = shmat(shm_id, NULL, 0);
```

size of the memory block

address where you would want **c** to be

Flags to set access rights. Can be SHM_RDONLY.

A tiny detail

- don't forget to delete the shared memory block!
- un-map it from address space:

```
int shmdt(c);
```

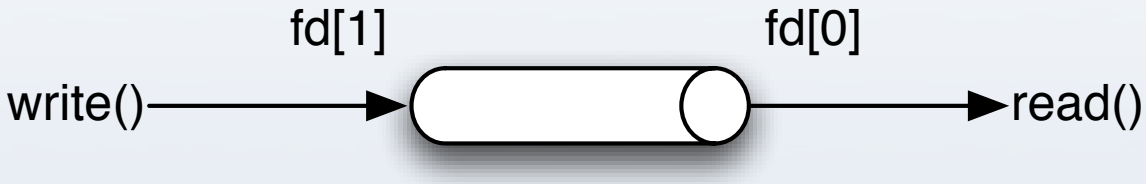
- use the shared memory control function:

```
int shmctl(shm_id, IPC_RMID);
```

- you can use the following command line tools:
  - `ipcs`: for displaying all shared memory areas
  - `ipcrm`: for removing shared memory areas

pipes

- creates a communication channel between two processes
- managed by two file descriptors: one for *write* operators, the other for *read* operations



pipe example

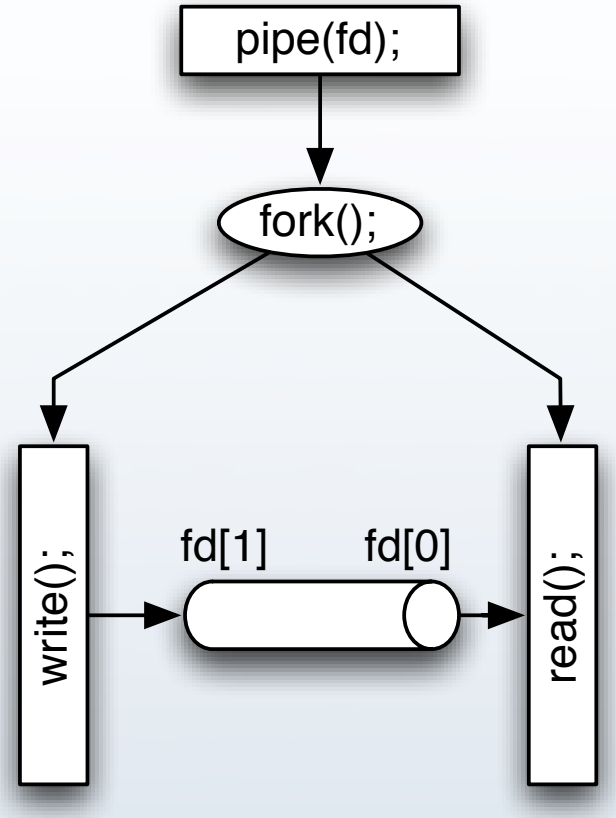
```
char buf[20];
int pipe_fd[2];
pipe(pipe_fd);

if (fork() == 0) {

  /* child writing to pipe */
  write(pipe_fd[1],
          "Hello World!", 13);

  exit(0);
} else {

  /* parent reading from pipe */
  read(pipe_fd[0], buf, 20);
}
```



semaphores

overview

- in real-life, used in intersections to regulate car traffic
- in computer science, it regulates access to shared data
- implemented using an integer *S* and two operations: *up* and *down*
  - if *S* > 0, *down* decrements *S*
  - if *S* = 0, *down* **blocks** until S > 0 and then decrements *S*
  - *up* increments S

Synchronizing access

PROCESS 1

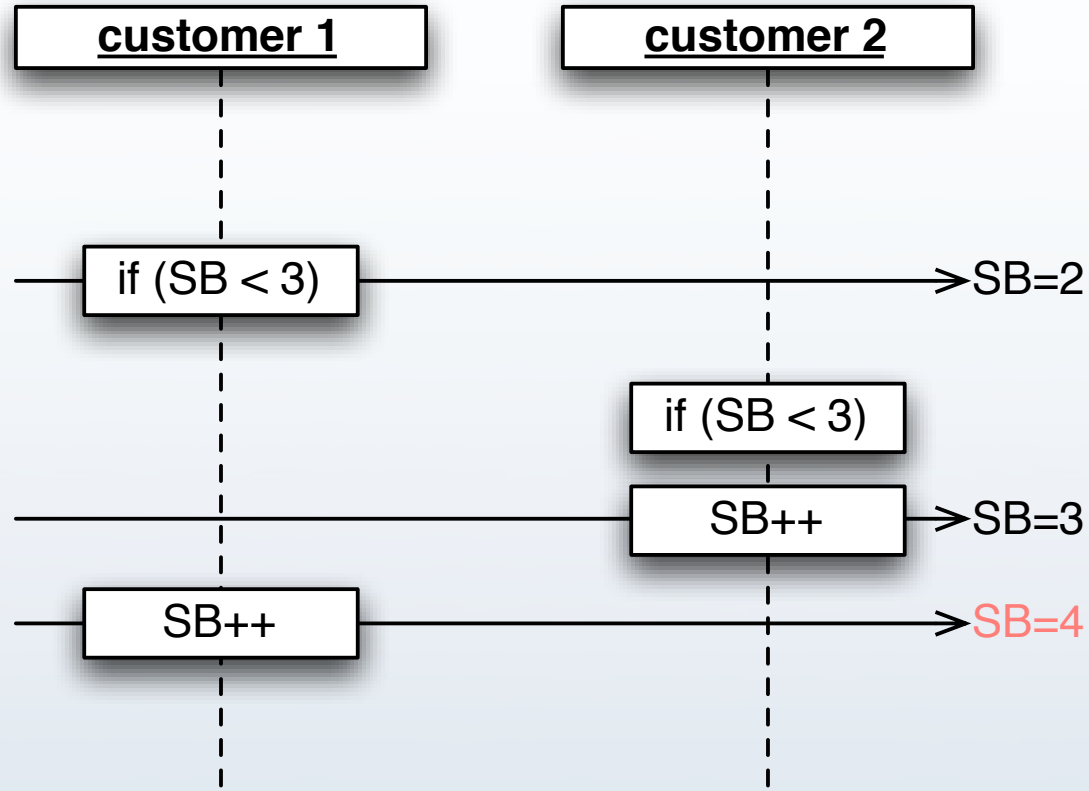
```
SB = 2;
if (SB < 3)
  SB++;
```

PROCESS 2

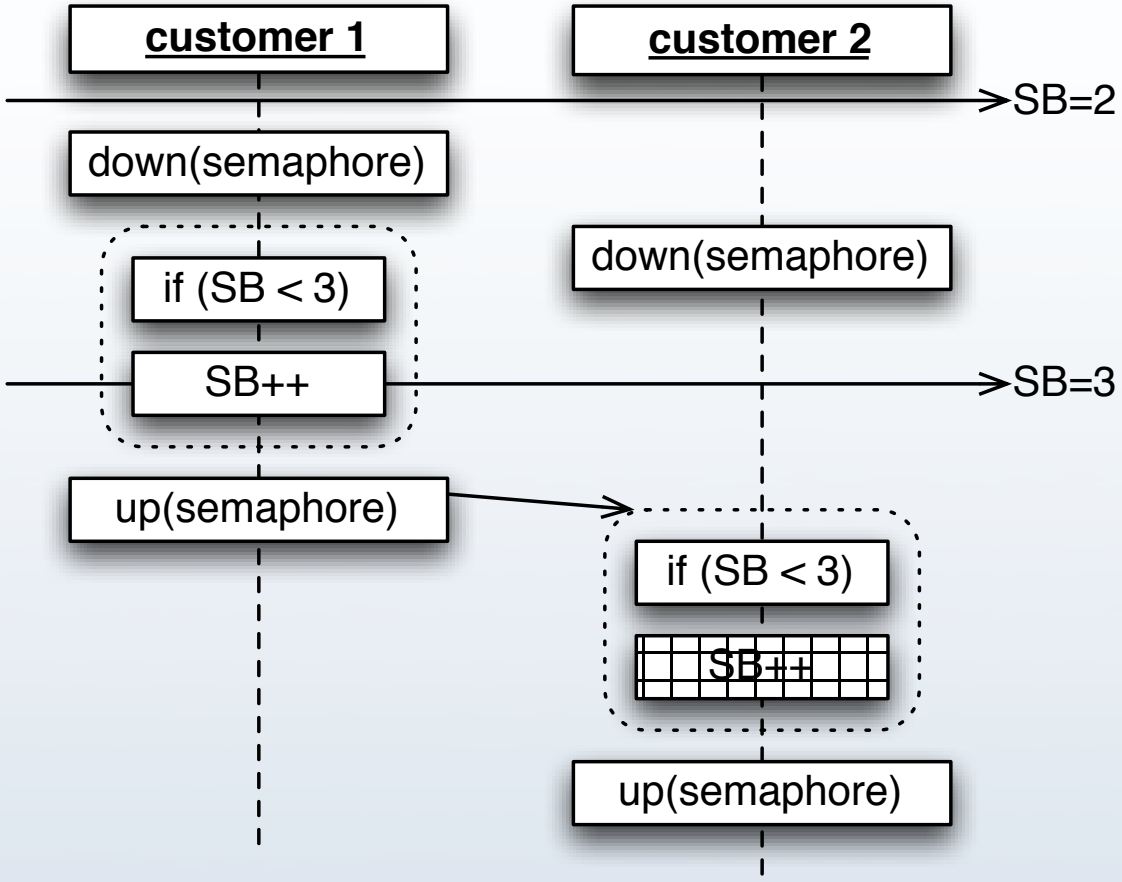
```
SB = 2;
if (SB < 3)
  SB++;
```

What is the value of SB at the end of the execution?

Synchronizing access



Synchronizing access



(sets of) semaphores in C

```
key_t key;
int sem_id;

key =
  ftok("/home/batman/some_file", 7);

sem_id = semget(key, 10, 0666 | IPC_CREATE);
```

creates a key based on an **existing** file and a number

access rights

number of semaphores in the set

flags: create a new semaphore set or use an existing semaphore set

Controlling semaphores

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

- `semid`: obtained by `semget`
- `sops`: array of commands on semaphores

```
struct sembuf {
  u_short sem_num; // the semaphore to control
  short sem_op; // value to increment / decrement
  short sem_flg;
};
```

If `sem_op`= 0, the call blocks until the semaphore gets to be 0.

- `nsops`: number of commands in the array

Example

Producer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 1,
        0666 | IPC_CREATE);

 getchar();

 struct sembuf sops = {
        .sem_num = 0,
        .sem_op = 1};
 semop(sem_id, &sops, 1);
}
```

Consumer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 0, 0);

 struct sembuf sops = {
        .sem_num = 0,
        .sem_op = -1};

 semop(sem_id, &sops, 1);

 printf("key pressed\n");
}
```

The keys are the same

Example

Producer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 1,
        0666 | IPC_CREATE);

 getchar();

 struct sembuf sops = {
        .sem_num = 0,
        .sem_op = 1};
 semop(sem_id, &sops, 1);
}
```

Consumer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 0, 0);

 struct sembuf sops = {
        .sem_num = 0,
        .sem_op = -1};

 semop(sem_id, &sops, 1);

 printf("key pressed\n");
}
```

The keys are the same

Example

The semaphore set is created once

Producer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 1,
       0666 | IPC_CREATE);

 getchar();

 struct sembuf sops = {
       .sem_num = 0,
       .sem_op = 1};
 semop(sem_id, &sops, 1);
}
```

Consumer

```
int main()
{
 key_t key; int sem_id;

 key = ftok("~/a", 1);

 sem_id = semget(key, 0, 0);

 struct sembuf sops = {
       .sem_num = 0,
       .sem_op = -1};

 semop(sem_id, &sops, 1);

 printf("key pressed\n");
}
```

A tiny detail

- don't forget to delete the semaphore set!
- use the semaphore control function:

```
int semctl(int semid, int semnum,
           int cmd, ...);
```

with the following arguments:

```
int semctl(semid, 0, IPC_RMID);
```

- you can use the following command line tools:
  - `ipcs`: for displaying all semaphores
  - `ipcrm`: for removing semaphores