Interprocess Communication

Bosky Agarwal

CS 518

Presentation Layout

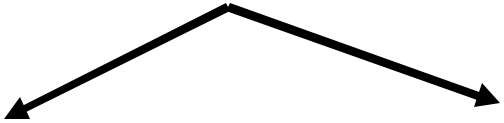
- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
    8. Example program
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
    3. Example program
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

Review

- Processes – running program
- $ ps aux – lists all processes running on the machine

- Characteristics

1. Unique identifier PID
2. Processes can share resources
3. Special processes



Swapper

- ✓ PID 0
- ✓ Scheduler

Init

- ✓ PID 1
- ✓ Invoked by kernel

4. Need some communicative methods

Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
    8. Example program
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
    3. Example program
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

Introduction

- Synchronization of actions
- Exchange of data
- Actors involved ⟶ User mode programs
- Dependence on kernel for communication
- Set of functionalities defined by OS
- Basic mechanisms

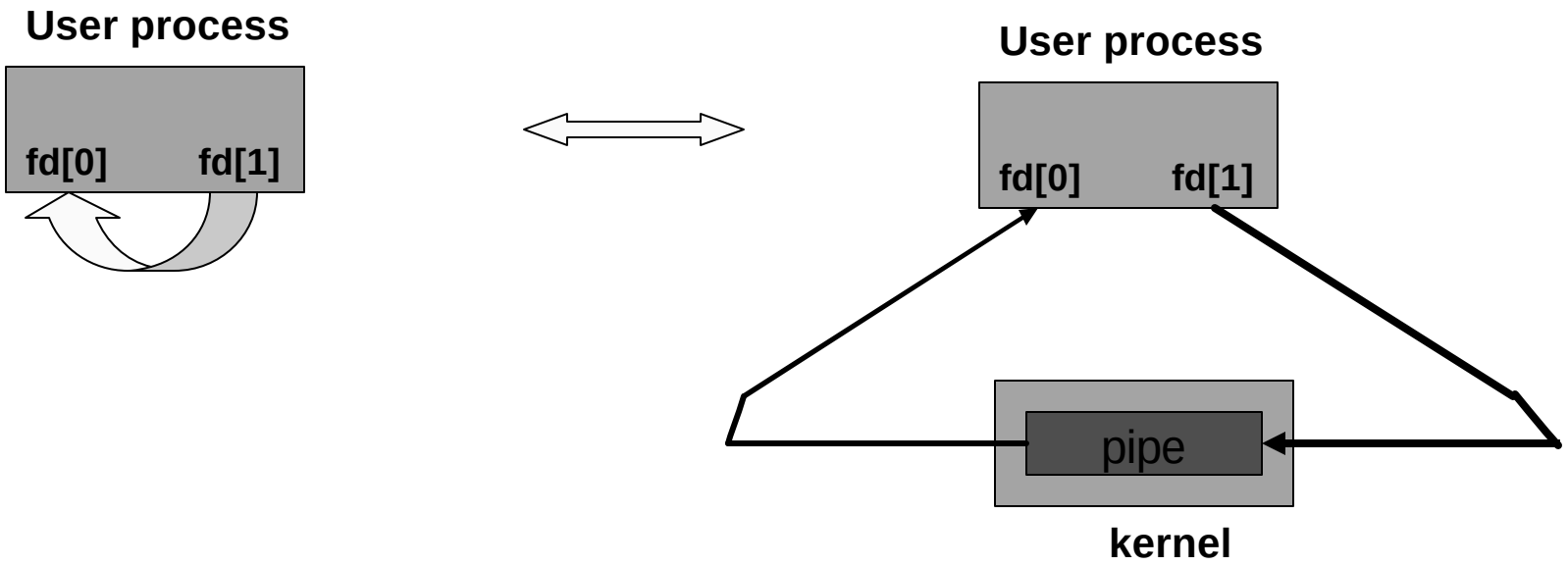
1. Pipes
2. FIFO's
3. System V IPC
    a. Semaphores
    b. Message Queues
    c. Shared Memory

Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

1. Pipes

- Provided in all flavors of Unix
- Unidirectional flow of data
- Data written to pipe is routed via kernel to another process
- Common ancestor necessary



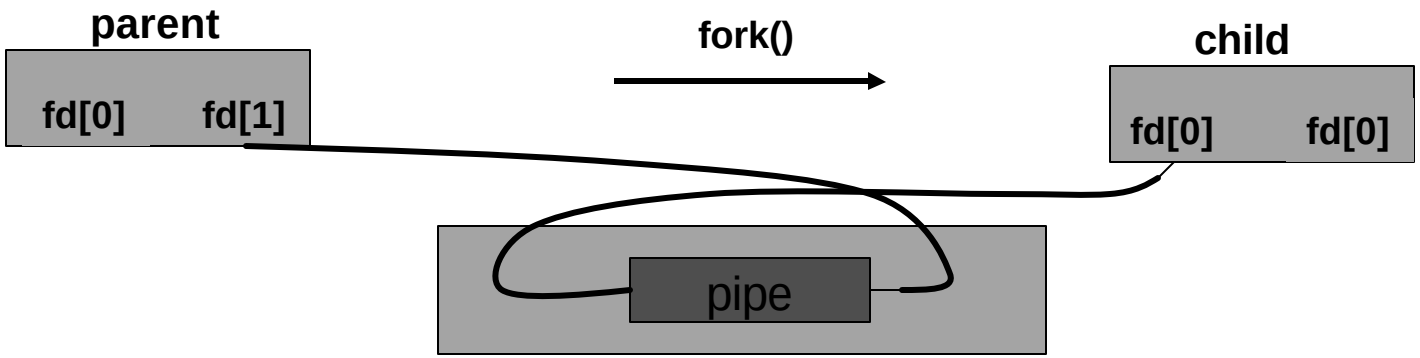
- Unix shell command – pipes created by ‘|’ e.g. $ ls | more
- Alternatively, $ ls > temp
  $ more < temp

Using Pipes

- Open files
- No image in mounted file system
- Creation of pipes:

```
#include <unistd.h>

int pipe(int fd[2]);

                                        Returns: 0 if OK, -1 on error
```

- Process passes file objects to children through fork()



Working of Pipes

e.g. $ ls | more

- Invocation of pipe()
- Invocation of fork() twice
- Invocation of close() twice

- Child1 performs:

    1. dup2(fd[1],1)
    2. Invoke close() twice
    3. Invoke execve() to execute ls

- Child2 performs:

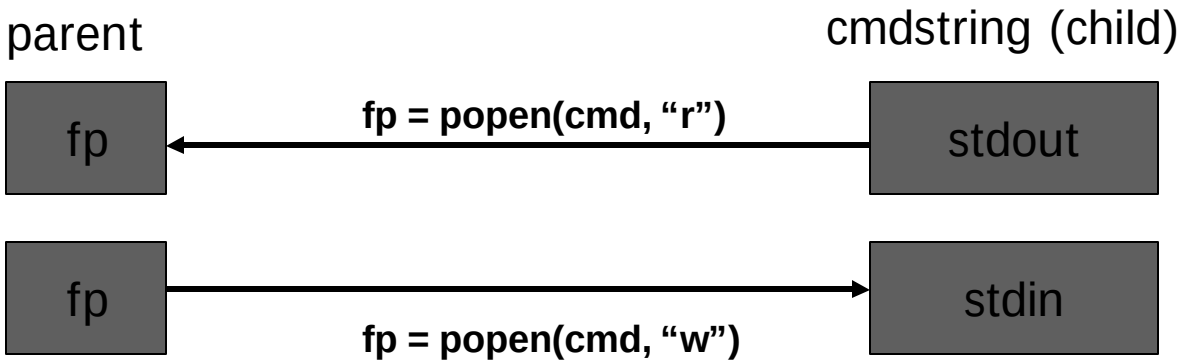
    1. dup2(fd[0],0)
    2. Invoke close() twice
    3. Invoke execve() to execute more

Working of Pipes (contd.)

- Use of wrapper functions included in C library
  a. popen()
  b. pclose()

```
#include <stdio.h>

FILE *popen(const char *cmdstring, const char *type);

                                        Returns: file pointer if OK, NULL on error
```

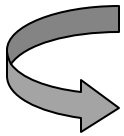
```
#include <stdio.h>

int pclose(FILE *fp);

                                Returns: termination status of cmdstring if OK, -1 on error
```



Pipe Data Structure

- Once created, we can use read() and write() of VFS
- For each pipe, kernel associates

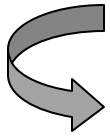
a. Inode object
b. Two file descriptors
c. Pipe buffer



- ✓ Page frame
- ✓ Data written into pipe , yet to be read
- ✓ Circular buffer accessed by both processes
- ✓ Use of i_sem semaphore included in *i_node* object
- ✓ Address, size and other parameters stored in *pipe_inode_info* structure

The pipefs special Filesystem

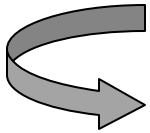
- Implemented as a VFS object – no disk image
- Organized into pipefs special filesystem
- Fully integrated into VFS layer
- Mounting of pipefs done in pipe.c



init_pipe_fs()

Creating and Destroying a Pipe

- Implemented as pipe() system call
- Serviced by sys_pipe()



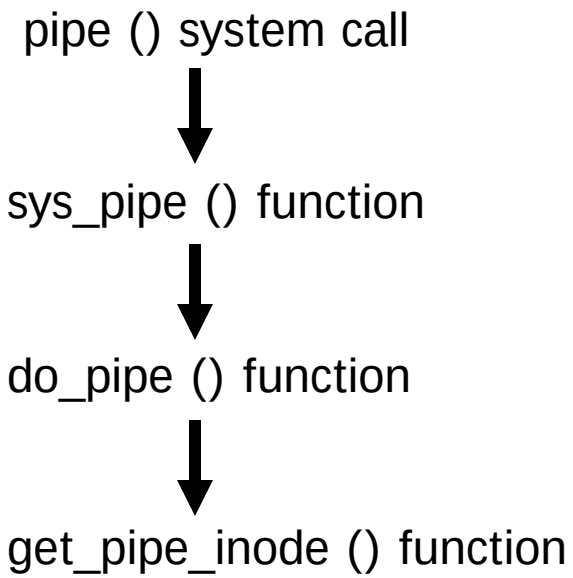
do_pipe()

- ✓ Invokes get_pipe_inode()
- ✓ Allocation and initialization of inode object
- ✓ Allocates pipe_inode_info structure
- ✓ Allocates page frame for pipe buffer
- ✓ Initializes start, len, waiting readers, waiting writers – 0
- ✓ Initializes r_counter and w_counter – 1
- ✓ Sets readers and writers – 1

If pipe's file object is opened by a process

Flow of Functions

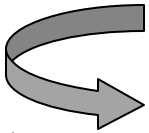
1. Structure pipe_inode_info
2. Mounting of pipefs special file system
3. Creation of a pipe



4. Reading from a pipe: pipe_read() function
5. Writing to a pipe: pipe_write() function

Reading from a Pipe

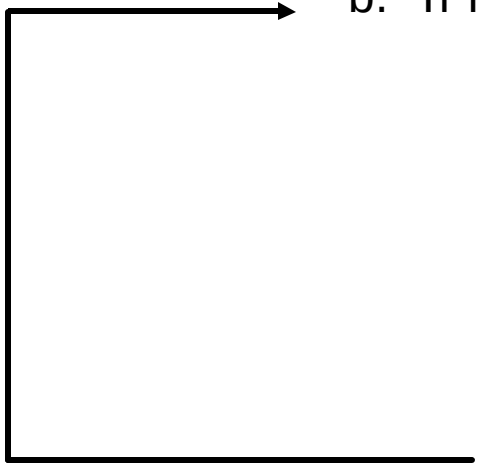
- Serviced by pipe_read()



- ✓ Acquires i_sem semaphore of inode.
- ✓ Determines len
  - a. If len = 0 determines the f_flags field in file object
  - b. If f_fileds = BLOCKED
    - i. Adds 1 to waiting readers
    - ii. Adds current to wait queue
    - iii. Sets TASK_INTERRUPTIBLE
    - iv. Invokes schedule()
    - v. Once awake, removes current from queue
    - vi. Acquires inode semaphore
    - vii. Decrements waiting_readers field
  - c. Copies requested number of bytes.
  - d. Updates start and len fields
  - e. Invokes wake_up_interrutible()
  - f. If not all requested bytes are copied
  - g. Return number of bytes read

Writing into a Pipe

- Serviced by pipe_write()
  - ✓ Acquires i_sem semaphore of inode.
  - ✓ Checks number of reading process,If 0
    - sends SIGPIPE , releases i_sem
  - ✓ If # bytes < buffer's size write operations – atomic
  - ✓ If # bytes > buffer size,check for free space
    - a. If no free space, and non-blocking
      - i. Release i_sem
      - ii. Return with -EAGAIN
    - b. If no free space and blocking,
      - i. Adds 1 to waiting writers
      - ii. Adds current to wait queue
      - iii. Releases inode semaphore
      - iv. Sets TASK_INTERRUPTIBLE
      - v. Invokes schedule()
      - vi. Once awake, removes current from queue
      - vii. Acquires inode semaphore
      - viii. Decrements waiting_writers field



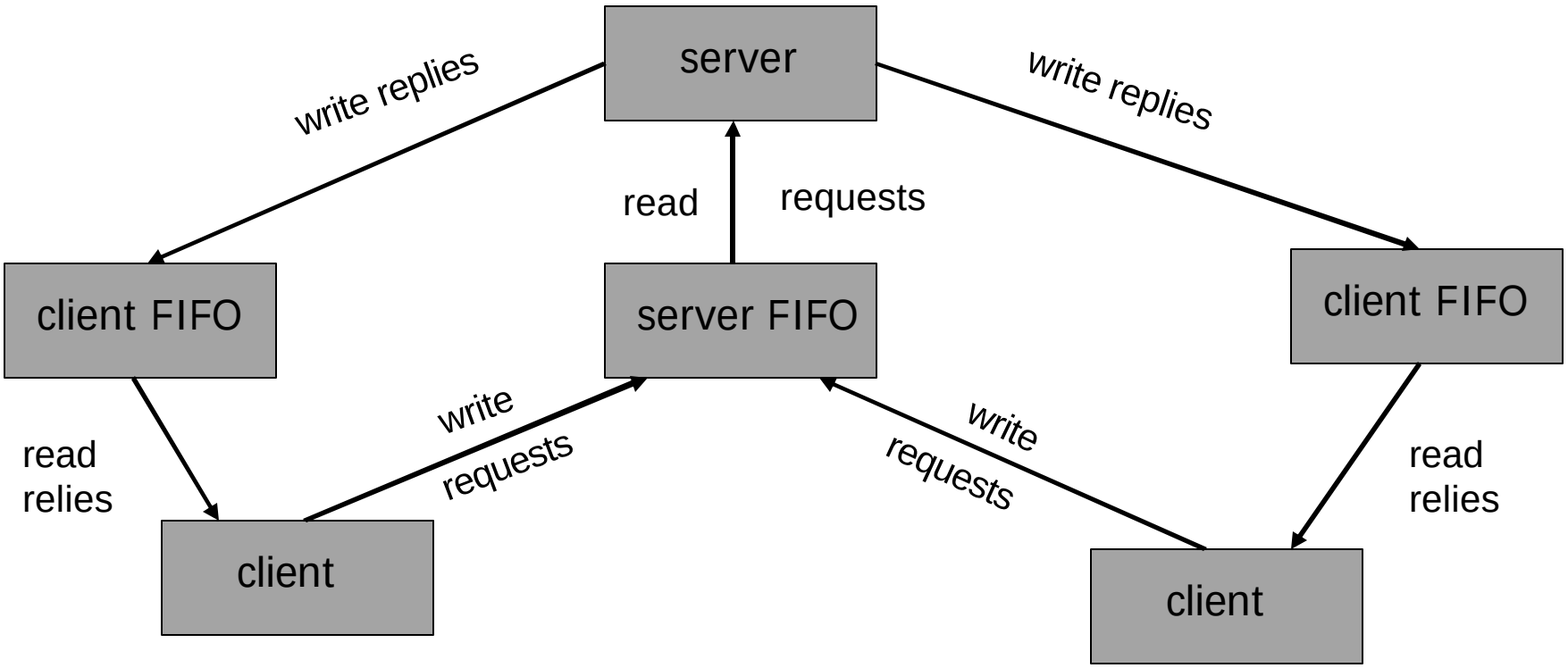
  - ✓ Writes requested bytes and other related work

Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

2. FIFOs (Named pipes)

- Removes the drawback of pipes
- First byte written will be first read
- FIFO filename included in system directory's tree
- FIFOs are bi-directional



Creating a FIFO

- POSIX introduced mkfifo() to create FIFO

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

int mkfifo(const char *pathname, mode_t mode);

                                Returns: 0 if OK, -1 on error
```

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

int mknod(char *pathname, mode_t mode);
                                Returns: 0 if OK, -1 on error
```

Opening a FIFO

- Once created, accessed using open(), read() write() etc
- Serviced by fifo_open()
  - ✓ Acquires i_sem
  - ✓ Checks i_pipe field
    - If NULL, allocates and initializes a new pipe_inode_info
  - ✓ If access mode is read-only or read-write
    - readers++, r_counter++
    - If no other reading process, wakes up any writing process
  - ✓ If access mode is write-only or read-write
    - writers++, w_counter++
    - If no other writing process, wakes up any reading process
  - ✓ If no readers or writers,
    - It either blocks
    - Terminates with error codes
  - ✓ Releases i_sem
    - Terminates
    - Returns 0 for success

Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

3. System V IPC

- First appeared in Columbus Unix
- Set of mechanisms allowing User Mode Processes to
  - Synchronize itself with other processes via Semaphores
  - Send and receive messages
  - Share a memory area with other processes
- Persistent until system shutdown
- Can be shared by any process
- Resource identified through
  - 32 bit IPC key – chosen by programmers
  - 32 bit IPC identifier – assigned by kernel and unique within a system

Using IPC Resources

- Kernel derives IPC identifier from IPC key
- If key not already associated,
  - New resource created
  - Return of positive IPC identifier
- Error code returned on IPC identifier request

Error Code	Description
EACCESS	Improper Access rights
EEXIST	Key already exists
EIDRM	Resource marked for deletion
ENOMEM	No storage space left
ENOSPC	Maximum limit on number of resources exceeded

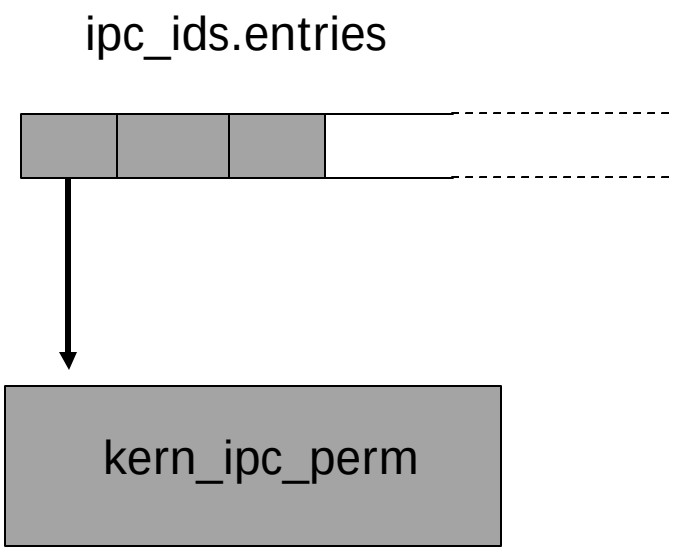
Using IPC Resources (contd.)

- Sharing of IPC resource done by
  - Processes agree on some fixed key
  - Specifying IPC_PRIVATE or
  - Key is not currently used and specify IPC_CREAT and/or IPC_EXCL as flag
- Incorrect referencing of wrong resource avoided by
  - Not recycling IPC identifiers as soon as they become free
  - IPC id assigned > previously allocated ID, exception on overflow
  - IPC id = s * M + i

    s = slot usage sequence number
    M = upper bound on number of allocatable resources
    i = slot index such that 0 <=i<=M

System V Data Structure

- Structure ipc_ids
- Structure kern_ipc_perm

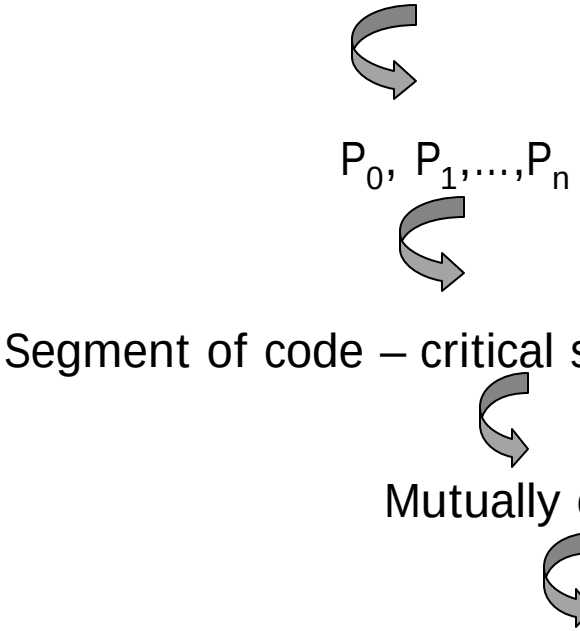


Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

Review of Semaphores

- Need a good understanding of Critical Region Problem



Change shared variables

Write to database

Write to a file etc

Review of Semaphores (contd.)

- Critical section { Entry section
  Exit section

repeat

Entry section

Critical section

Exit section

Remainder section

until false;

Review of Semaphores (contd.)

- Semaphore S – synchronization tool
- Counters to provide access to shared data

Integer variable

Accessed via atomic operations

wait(S)

- ✓ when request to enter critical section
- ✓ decrements
- ✓ while S.val <= 0 do no_op;
  S.val --;

signal(S)

- ✓ when it wants to leave
- ✓ increments
- ✓ S.val ++;

Semaphores in Linux

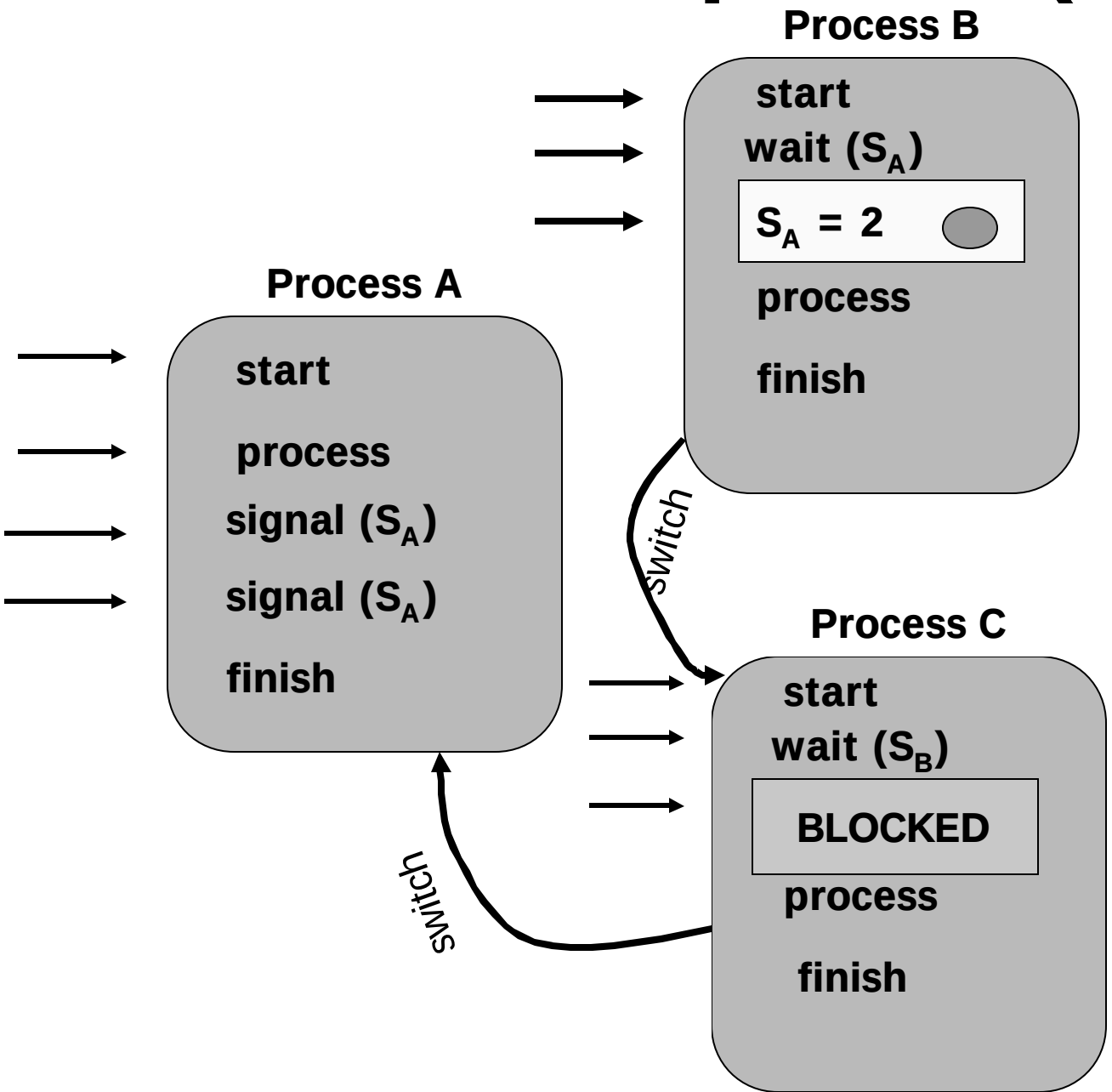
- Sharing depends on value of semaphore

  - if +ve, then protected resource is available
  - if 0, protected resource unavailable

- Process decrements semaphore value
- Kernel blocks process until semaphore value becomes +ve
- After use, it increments the semaphore value

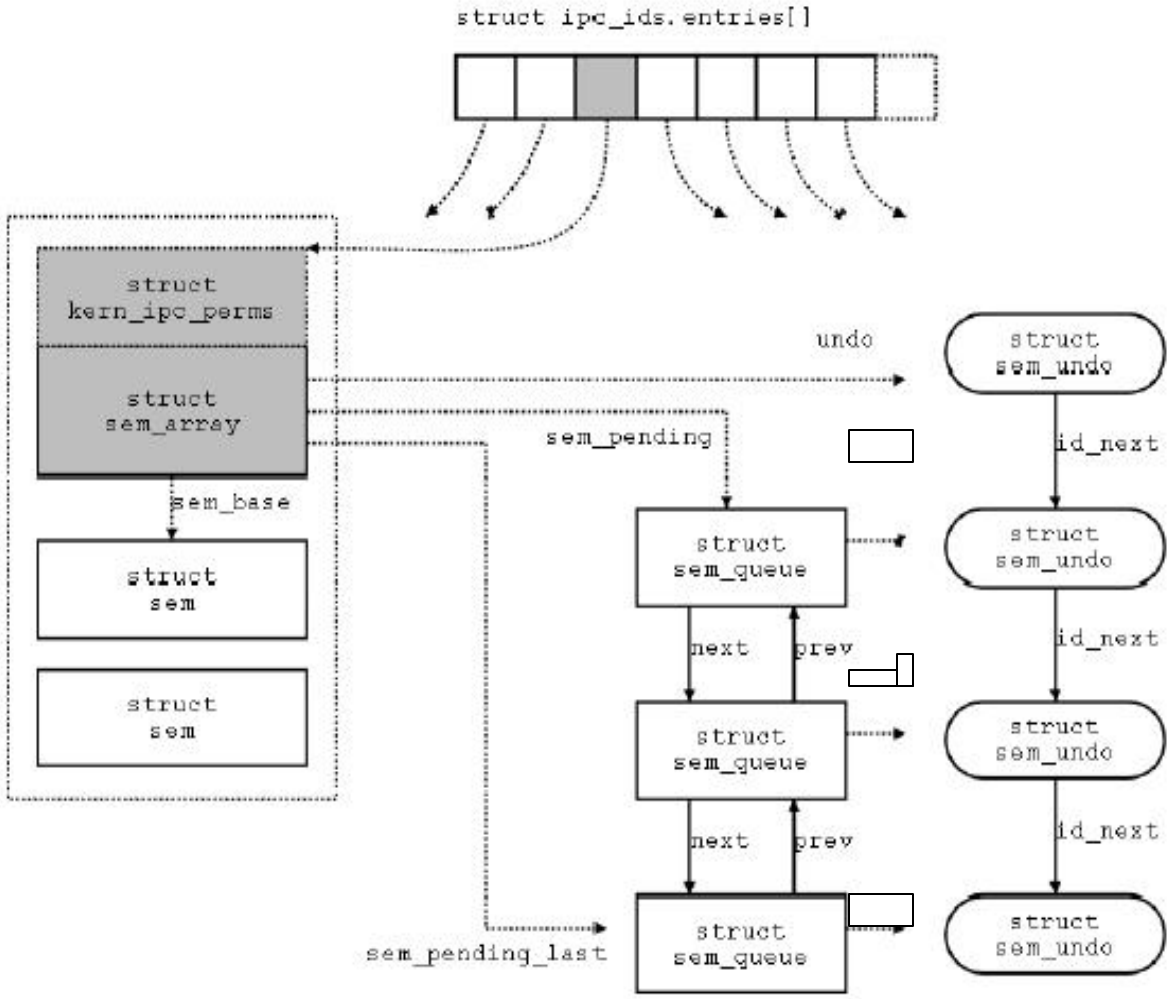
  Other processes wake up

Review of Semaphores (contd.)



Semaphores Data Structures

- Structure sem_array
- Structure sem



Semaphores Data Structures (contd.)

/proc/sys/kernel/sem or ipcs -ls

This file contains 4 numbers defining limits for System V IPC semaphores

SEMMSL -The maximum semaphores per semaphore set

SEMMNS - A system-wide limit on the number of semaphores in all semaphore sets

SEMOPM - The maximum number of operations that may be specified in a **semop()**

SEMMNI - A system-wide limit on the maximum number of semaphore identifiers.

Tuning

$ echo 250 32000 100 128 > /proc/sys/kernel/sem

$ echo "kernel.sem=250 32000 100 128" >> /etc/sysctl.conf

Semaphore Functions

- Semaphore creation – serviced by semget()

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

int semget(key_t key, int nsems, int flag);

                                        Returns: semaphore id if OK, -1 on error
```

- Semaphore handling – serviced by semctl()

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

int semctl(int semid, int semnum, int cmd,union semun arg);
```

cmd = IPC_STAT, IPC_SET, IPC_RMID, GETVAL, SETVAL, GETPID, GETALL, SETALL

Semaphore Functions (contd.)

- Semaphore operations (contd.) – serviced by semop()

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

int semopl(int semid, struct sembuf semoparray[], size_t nops);
```

nops = number of operations in the array

Presentation Layout

- Review
- Introduction
- Pipes

    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe

- FIFOs

    1. Creating a FIFO
    2. Opening a FIFO

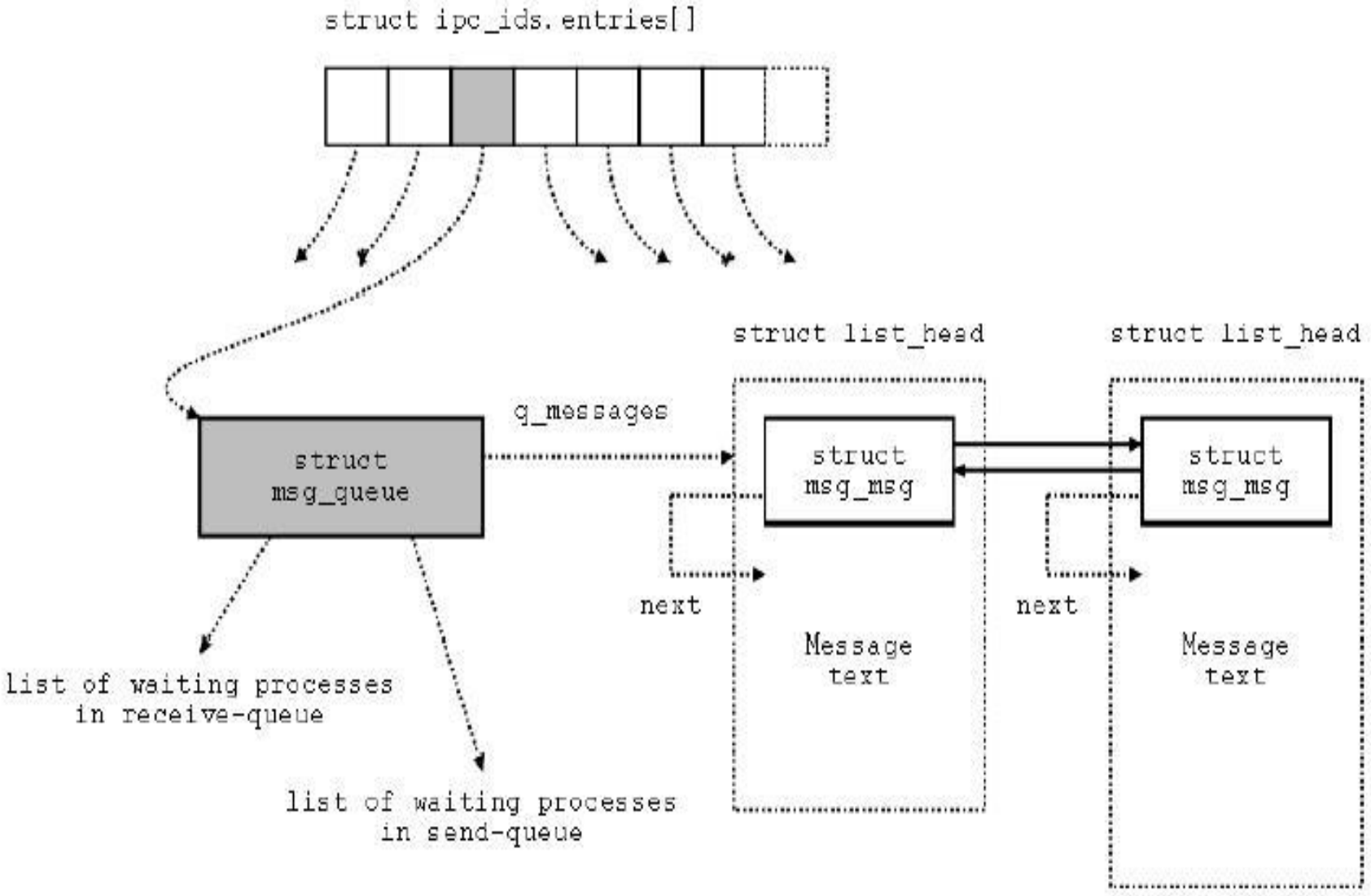
- System V IPC

    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

Message Queues

- Linked list of messages stored within kernel
- Identified by Message identifier
- Message = message header + text
- New message queue is created or opened using msgget()
- Messages are fetched from queue via msgrcv()
- Messages are sent to queue via msgsnd()
- Message Queues data structures
  - Structure msg_queue
  - Structure msg_msg

Message Queues Data Structures



Message Queues Data Structures (contd.)

- ***/proc/sys/kernel/msgmni***

  The maximum number of messages

- ***proc/sys/kernel/msgmax***

  Size of each message

- ***proc/sys/kernel/msgmnb***

  The total size of message queue

Message Queues Functions

- Queue creation – serviced by msgget()

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

int msgget(key_t key, int flag);

                                        Returns: message queue id if OK, -1 on error
```

- Message Queue operations – serviced by msgctl()

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

int msgctl(int msqid, int cmd,struct msqid_ds *buf);

                                        Returns: 0 if OK, -1 on error
```

cmd = IPC_STAT, IPC_SET, IPC_RMID

Message Queues Functions (contd.)

- Sending messages – serviced by msgsnd()

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

int msgsnd(int msqid, const void *ptr, size_t nbytes, int flag);

                                        Returns: 0 if OK, -1 on error
```

ptr= pointer to structure msgbuf
flag = IPC_NOWAIT

Message Queues Functions (contd.)

- Receiving messages – serviced by msgrcv()

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

int msgrcv(int msqid, const void *ptr, size_t nbytes, long type,int flag);

                        Returns: size of data portion of message if OK, -1 on error
```

ptr= pointer to structure msgbuf
flag = IPC_NOWAIT
type = specifies message to fetch

type == 0 The first message is returned
type > 0 The first message whose message type = type
type < 0 The first message whose message type <= |type|

Presentation Layout

- Review
- Introduction
- Pipes

    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe

- FIFOs

    1. Creating a FIFO
    2. Opening a FIFO

- System V IPC

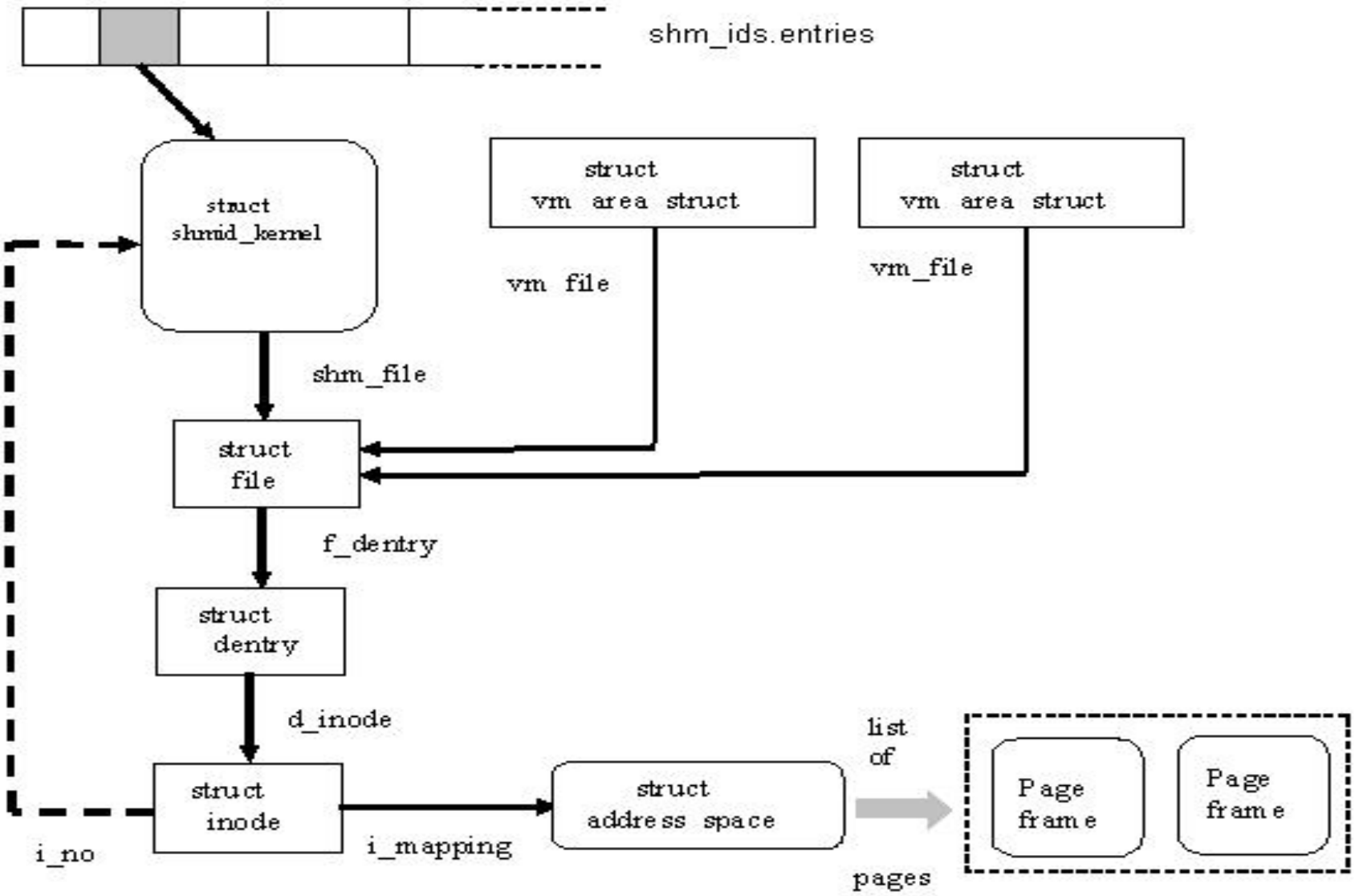
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions

Shared Memory

- Allow processes to share given piece of memory
- Synchronization is very important here
- Obtaining shared memory via shmget()
- Shared memory operations via shmctl()
- Shared memory attachment via shmat()
- Detach shared memory via shmdt()

- Shared memory data structures
  - Structure shm_id_ds

Shared Memory Data Structures



Shared Memory Data Structures (contd.)

- ***/proc/sys/kernel/shmmni***

  The number of IPC shared memory regions

- ***proc/sys/kernel/shmmax***

  Size of each region

- ***proc/sys/kernel/shmall***

  The total size ofall memory regions

Shared Memory Functions

- Obtaining shared memory – serviced by shmget()

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

int shmget(key_t key, int size int flag);

                                        Returns: shared memory id if OK, -1 on error
```

- Shared memory operations – serviced by shmctl()

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

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

                                        Returns: 0 if OK, -1 on error
```

cmd = IPC_STAT, IPC_SET, IPC_RMID, SHM_LOCK, SHM_UNLOCK

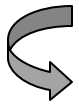
Shared memory Functions (contd.)

- Attaching shared memory regions – serviced by shmat()

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

void *shmat(int shmid, void *addr, int flag);

                    Returns: pointer to shred memory if OK, -1 on error
```

flag = SHM_RND (RND = round)
addr = specifies address in calling process

addr == 0 Segment attached to first available address selected by kernel
addr != 0 Segment attached to address given by addr if SHM_RND not specified
addr != 0 Segment attached to address (addr – (addr mod SHMLBA))



SHM_LBA = low boundary address multiple

Shared memory Functions (contd.)

- Detaching shared memory regions – serviced by shmdt()

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

int  shmat(void *addr);

                        Returns: 0 if OK, -1 on error
```

Presentation Layout

- Review
- Introduction
- Pipes
    1. Using pipes
    2. Working of pipes
    3. Pipe Data Structure
    4. Special pipefs File System
    5. Creating and destroying pipes
    6. Reading from a pipe
    7. Writing to a pipe
- FIFOs
    1. Creating a FIFO
    2. Opening a FIFO
- System V IPC
    1. Using IPC Resources
    2. Data Structures
    3. Semaphores
        a. Data Structures
        b. Functions
        c. Example program
    4. Message Queues
        a. Data structures
        b. Functions
    5. Shared Memory
        a. Data structures
        b. Functions
- Summary
- References

Summary

- Demonstration of methods for processes to communicate
- Synchronization is widely handled all the time by kernel processes
- This presentation dealt with User mode processes
- Basic working idea of the main IPC was provided

References

- The design of the UNIX Operating System, Maurice J. Bach
- Advanced Programming in the Unix Environment W. Richard Stevens
- Understanding Linux Kernel Daniel P. Bovet & Marco Cesati
- http://lxr.linux.no/source/
- http://www.cs.cf.ac.uk/Dave/C/