Systems Calls - IPCs III

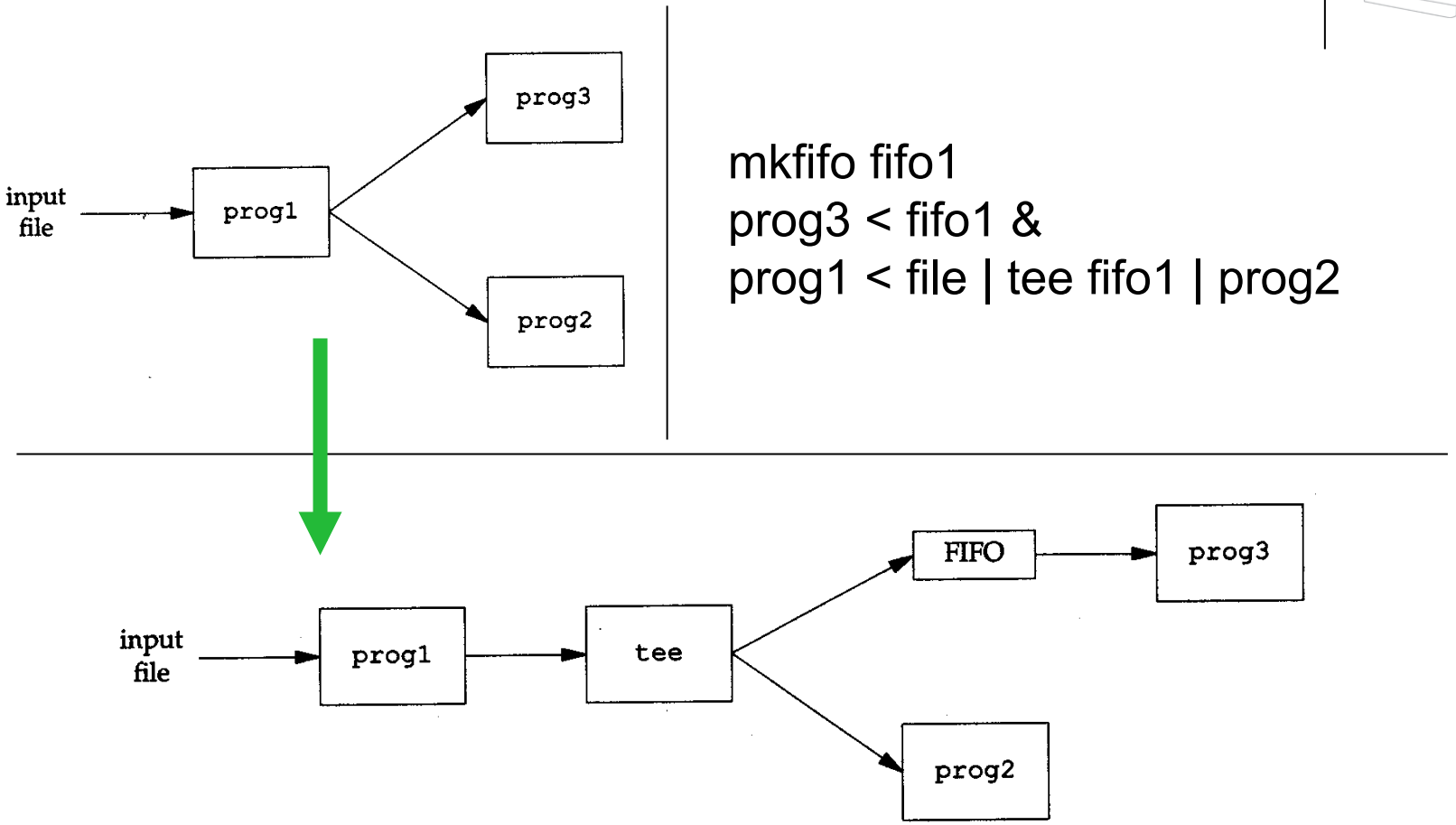
Alain NINANE – RSSI UCL

19 Avril 2016

Recap's

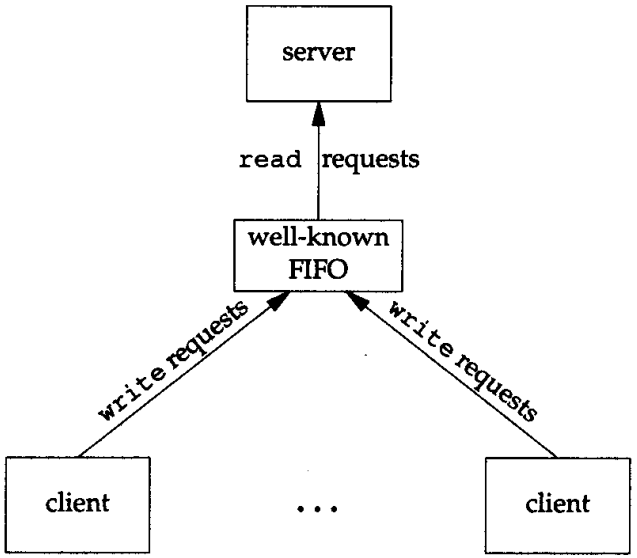
- Pipes
  - Exercice: chain/loop of processes (test01)
- Named pipes
  - Duplication of output streams (next slide)
  - Client/Server architecture (next slide)
- Signals
  - Different behavior between BSD and SYSV
- Sockets

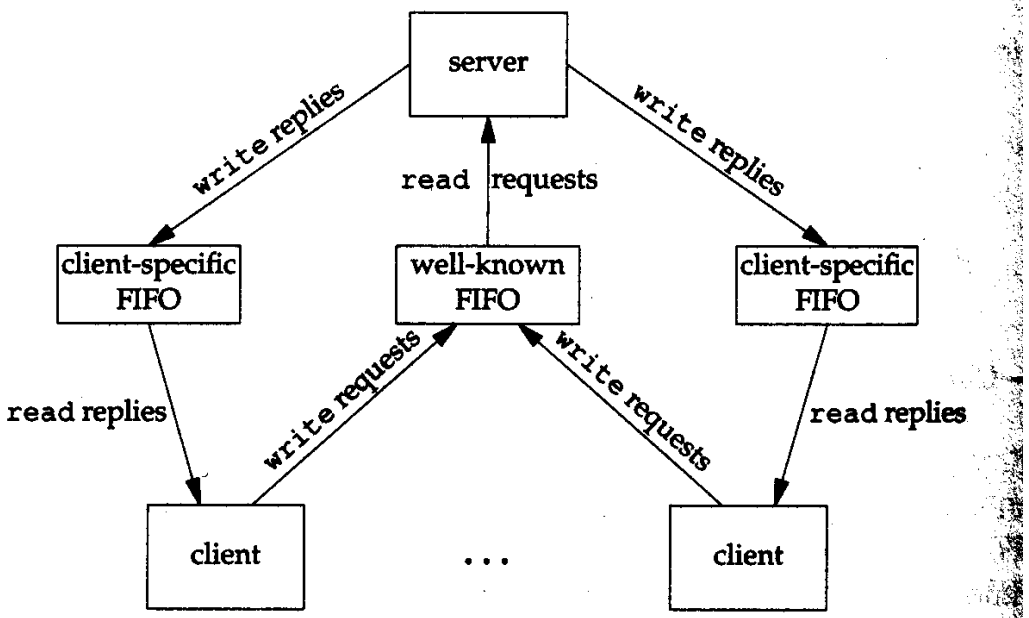
Named Pipes - 2 Filters




```
mkfifo fifo1
prog3 < fifo1 &
prog1 < file | tee fifo1 | prog2
```

Names Pipes - client/server





SysV IPCs - Introduction

- Different IPC structure
  - Message queues
  - Semaphores
  - Shared memory
- Originated in the 70s under internal version of UNIX
- All structures share common characteristics
  - ipc identifier
  - permission structure

SysV IPCs - IPC Identifier (I)

- The IPC identifier identifies the IPC structure
  - An allways incrementing integer (wrap around 0)
  - Returned by msget(), semget(), shmget()
    - int msgget(**key_t key**, **int msgflg**);
    - int semget(**key_t key**, int nsems, **int semflg**);
    - int shmget(**key_t key**, size_t size, **int shmflg**);
  - key allows processes to agrees on IPC structures
  - IPCV identifiers values are global in the UNIX system (systemwide)

SysV IPCs - IPC Identifier (II)

- Key
  - IPC_PRIVATE
  - Number key
- Flags
  - IPC_CREAT
    - if key available -> creates IPC object
  - IPC_EXCL
    - if key already used -> generates an error

SysV IPCs - IPC Identifier (III)

- IPC structure are system wide objects
- Can be used by unrelated processes
- They must agree on the IPC identifier
  - Common key in an include file
  - Server publish the IPC identifier to clients through files
  - Key can be generated by ftok()
  - ftok(const char *pathname, int project_id)

SysV IPCs - Perm. Structure (I)

- Each IPC structure has an associated ipc_perm stucture
  - struct ipc_perm
  - { uid_t uid; /* owner's effective user id */
    gid_t gid;
    uid_t cuid; /* creator's effective used id */
    gid_t cgid;
    mode_t mode; /* access mode */
    ...
    key_t key;
  - }

SysV IPCs - Perm. Structure (II)

- Fields of the ipc_perm structure are similar to the regular files ownership and access modes
- Changed by msgctl(), semctl() and shmctl()
  - Similar to chown, chmod

Message queues - Intro (I)

- Message queues - kernel internals
  - Linked list of messages stored in the kernel space
  - Queue is identified by a message queue id
  - Queue is created or opened by msgget()
  - Messages are sent by msgsnd()
    - Added at the end of message queue
  - Messages are read by msgrcv()
    - Messages have a type field for selected readout
    - Not necessarily first-in/first-out
  - Queue has a msqid_ds structure

Message queues - Intro (II)

- Message queue internal structure

```
struct msqid_ds {
  struct ipc_perm  msg_perm; /* see Section 14.6.2 */
  struct msg  *msg_first;  /* ptr to first message on queue */
  struct msg  *msg_last;   /* ptr to last message on queue */
  ulong        msg_cbytes; /* current # bytes on queue */
  ulong        msg_qnum;   /* # of messages on queue */
  ulong        msg_qbytes; /* max # of bytes on queue */
  pid_t        msg_lspid;  /* pid of last msgsnd() */
  pid_t        msg_lrpid;  /* pid of last msgrcv() */
  time_t       msg_stime;  /* last-msgsnd() time */
  time_t       msg_rtime;  /* last-msgrcv() time */
  time_t       msg_ctime;  /* last-change time */
};
```

Message queue - create/open

- int qid = msgget(IPC_PRIVATE,mode)
- int qid = msgget(KEY,mode|flags)
  - flags &= IPC_CREATE
    - create queue if queue doesn't exists already
    - open queue if queue exists already
  - flags &= IPC_EXCL
    - create queue if queue doesn't exists already
    - error if queue exists already
- Demo: test02a and test02b

Message queue - send

- int ret = msgsnd(int qid, const void *ptr,
  size_t nbytes, int flag);
  - qid: message queue identifier
  - ptr: pointer to a long integer followed by the message data
    - struct MyMessage
      { long type;
      char text[MAX_LENGTH];
      }
  - nbytes: size of text area (< MAX_LENGTH)
  - flag: IPC_NOWAIT
  - demo: test03a - test03b

Message queue - receive (I)

- int ret = msgrcv(int qid, void *ptr, size_t nbytes,
  long type, int flag);
  - qid: message queue identifier
  - ptr: pointer to a long int. followed by a data buffer where message will be read
  - struct MyMessage
    { long type;
      char buffer[BUF_MAX_LENGTH];
    }
  - nbytes: size of buffer area (= BUF_MAX_LENGTH)
    - message > BUF_MAX_LENGTH
      - message stays in queue - error = E2BIG
      - message truncated (if MSGNOERROR flag is set)

Message queue - receive (II)

- int ret = msgrcv(int qid, void *ptr,
  size_t nbytes, long type, int flag);
  - type
    - type = 0: first message in queue is read
    - type > 0: first message with type is read
    - type < 0: first message with type <= |type| is read
  - flag
    - IPC_NOWAIT: do not block if queue empty
    - MSG_NOERROR: truncate message
      if message > nbytes

Message queue - control

- int msgctl(int qid, int cmd, struct msqid_ds *ptr)
  - qid: the queue id
  - ptr: pointer to the msqid_ds IPC structure
  - cmd:
    - IPC_STAT: returns the control structure
    - IPC_SET: write the following fields to the control struct.
      msg_perm.uid msg_perm.gid
      msg_perm.mode msg_qbytes
    - IPC_RMID: remove the message queue
      - all clients will get errno EIDRM

Message queue - summary

- provides a bi-directionnal communication
- between unrelated processes
- messages are typed
- message boundaries are preserved

- message queues are:
  - outside of the file system hierarchy
  - outside of the processes arborescence

Semaphores - Introduction (I)

- Purpose
  - Synchronization of operations
- Semaphore
  - A positive short integer
  - Elementaty operations
    - increment or decrement the semaphore value
    - suspend the execution of calling process until the semaphore reaches a particular value (e.g. > 0)
  - Provides controlled/protected access to critical ressource
    - e.g. to manipulate doubly-linked lists

Semaphores - Introduction (II)

- Practical operations
  - Semaphore initialization
    - s = init. value (e.g. 1)
  - Process **take** semaphore
    - while(s <= 0); // Note '**;**' i.e. wait loop
    - s = s - 1;
  - Ressource is now locked (critical section)
  - Process **release** semaphore
    - s = s + 1

Semaphores - Introduction (III)

- Remark 1
  - Init. value can be ‘n’ > 1
  - As many as ‘n’ processes can access the ressource simultaneously
- Remark 2
  - There is a concurrency/failure point between
    - while(s <= 0); and
    - s = s - 1;
  - Semaphore operations must be atomic

Sys V IPC - Semaphores (I)

- System V IPCs provides semaphores
  - Really a **set** of semaphore
  - Multiple operations can be processed atomically

- System calls
  - semget(): get an identifier for a set of semaphores
  - semctl(): control operations on a set of semaphores
  - semop(): operations on a set of semaphores

Sys V IPC - Semaphores (II)

- Semaphore internal structure (in kernel space !)

```
struct semid_ds {
  struct ipc_perm  sem_perm;  /* see Section 14.6.2 */
  struct sem  *sem_base;  /* ptr to first semaphore in set */
  ushort       sem_nsems; /* # of semaphores in set */
  time_t       sem_otime; /* last-semop() time */
  time_t       sem_ctime; /* last-change time */
};
```

```
struct sem {
  ushort  semval;   /* semaphore value, always >= 0 */
  pid_t   sempid;   /* pid for last operation */
  ushort  semncnt;  /* # processes awaiting semval > currval */
  ushort  semzcnt;  /* # processes awaiting semval = 0 */
};
```

Semaphore - getting ...

- int semget(key, nsems, semflg)
  - key_t key: a number key
  - int semflg: mode or'ed with IPC_CREATE/IPC_EXCL
    - Exactly like msgget()
  - nsems
    - the number of semaphore in the set
  - returns : a semaphore (set) identifier
  - caveat: semaphores are created but not initialized

Semaphore - controls ... (I)

- int semctl(semid, semnum, cmd, arg)
  - int semid: a semaphore (set) identifier
  - int semnum: the semaphore number in the set (if meaningful)
  - int cmd: (next slide)
  - last argument '**arg**' is polymorphic depending on '**cmd**'
    - union
      ```
      {       int val;
              struct semid_ds *buf;
              ushort *array;
      } arg;
      ```

Semaphore - controls ... (II)

- GETVAL/SETVAL: value of a single sem.
- GETALL/SETALL: value of all sems.
- GETPID: last process who operated on sem.
- GETNCNT: number of procs. waiting on sem. value > current value
- GETZCNT: number of procs waiting on sem. value = 0
- IPC_STAT: returns semid_ds to pointer buf
- IPC_SET: modify effective uid/gid and modes through ipc_perm of semid_ds.
- IPC_RMID: remove the semaphore set

Semaphore - operations (I)

- int semop(semid, sops, nsops)
  - int semid; // semaphore set id
  - struct sembuf sops[ ]; // array of sem ops
  - int nsops; // nbr os sem ops
- sembuf describe a semaphore operation
  - struct sembuf

```
{       ushort sem_num;   // semaphore number
        short sem_op;     // semaphore operation
        short sem_flg;    // semaphore flag
};
```

- **Remark**: semop allows for a **set** of operations on a set of semaphores **atomically**

Semaphore - operations (II)

- short sem_op - semaphore operation
  - sem_op > 0
    - value is added to the current semaphore value
  - sem_op < 0
    - |sem_op| <= semaphore value
      - |sem_op| is subtracted from the current semaphore value
    - |sem_op| > semaphore value
      - semncnt++;
      - current process goes to sleep until |sem_op| <= semaphore
      - sem_op| is subtracted from the current semaphore value
  - sem_op == 0
    - semaphore value != 0
      - semzcnt++;
      - current process goes to sleep until semaphore value = 0

Semaphore - operations (III)

- short sel_flg
  - IPC_NOWAIT
    - if condition would cause the process to sleep, the system call returns immédiately with errno EAGAIN;
  - SEM_UNDO
    - all modifications of semaphore values in the current process are recorded to be “played back” at the end of current process
- demo: test05[abc]

Sys V IPCs - Shared Memory (I)

- Memory shared between processes
  - a way to exchange data between processes
- Fastest means between processes
  - No system calls involved
- Shared memory
  - must be created by a process
  - must be attached by processes (even creator)
  - must be detached when unused

Sys V IPCs - Shared Memory (II)

Shared Memory Internel Structure (kernel address space)

```
struct shmid_ds {
  struct ipc_perm  shm_perm; /* see Section 14.6.2 */
  struct anon_map *shm_amp;  /* pointer in kernel */
  int      shm_segsz;    /* size of segment in bytes */
  ushort   shm_lkcnt;    /* number of times segment is being locked
  pid_t    shm_lpid;     /* pid of last shmop() */
  pid_t    shm_cpid;     /* pid of creator */
  ulong    shm_nattch;   /* number of current attaches */
  ulong    shm_cnattch;  /* used only for shminfo */
  time_t   shm_atime;    /* last-attach time */
  time_t   shm_dtime;    /* last-detach time */
  time_t   shm_ctime;    /* last-change time */
};
```

Shared Memory - getting ...

- int shmget(key, size, shmflg)
  - key_t key; // as usual for msgget() and semget()
  - size; // size of shared memory segment
    - Really the minimum size of segment (can get more)
    - Only needed for shm creator process
  - shmflg // as usual
    - IPC_CREATE
    - IPC_EXCL

Shared Memory - controls ...

- int shmctl(shmid, cmd, buf)
  - int shmid; /* The shared memory identifier */
  - int cmd; /* The command to execute on shm */
  - struct shmid_ds *buf; /* If needed by cmd */
  - CMD:
    - SHM_STAT: fetch the shmid_ds structure from kernel
    - SHM_SET: set authorized fields of kernel shmid_ds
    - SHM_RMID: destroy the shared memory
      - caveat: the segment exists until last proc. detach it
the shmid is removed (to prevent further attach)
    - SHM_LOCK/UNLOCK: lock shm in physical memory

Shared Memory - attach/detach

- void *shmat (shmid, addr, flag)
  - int shmid; /* The shared memory identifier */
  - void *addr;
    - /* Should be NULL in today's applications !! */
  - int flag;
    - SHM_RDONLY;
  - returns: the user space address to be used by the calling process
- void shmdt(addr)

System V IPC - Discussion

- Pro and Con
- /dev/zero
  - od -bcv /dev/zero | more -c
- /dev/null
- Debugging
  - http://media.techtarget.com/searchEnterpriseLinux/downloads/Linux_Toolbox.pdf