Other Topics

An Advanced Introduction to Unix/C Programming

John Dempsey
COMP-232 Programming Languages
California State University, Channel Islands

What to cover

- Read/Write/Execute
- Signals
- Fork
- Shared Memory
- Message Queues
- Unix File System
- Threads
- Semaphores / Mutex
- Client/Server Networking

d rwx rwx rwx

rwx	rwx	rwx
user	group	world

d rwx rwx rwx

File Mode Bits	
Bit	Meaning
d	Directory
r	Read Access
w	Write Access
x	Execute

RWX Groupings	
First rwx	User Group Access (owner of file)
Second rwx	Group Access (if in group, have access)
Third rwx	World Access (everyone on system)

For Files:

r	**– You can read the file**
w	**– You can modify the file**
x	**– You run execute (run) file**

For Directories:

r	**– You can see the file name in the directory.**
w	**– You can add, remove, and rename the file in the directory.**
x	**– You can use the directory name in a file path and change into directory.**

d rwx rwx rwx

File Mode Bits	
Bit	Meaning
d	Directory
r	Read Access
w	Write Access
x	Execute

RWX Groupings	
First rwx	User Group Access (owner of file)
Second rwx	Group Access (if in group, have access)
Third rwx	World Access (everyone on system)

```
% ls –l
drwxr-x--- 1 john staff  4096 Dec 24 14:15 my.dir
-rwxr-x--- 1 john staff 16728 Dec 25 15:51 a.out
-rw-r--r-- 1 john staff   232 Dec 24 14:17 continue.c
```

You need r-x access to cd into a directory.

chmod – change file mode bits

% **chmod 644 myfile.txt** ← Sets myfile.txt to rw-r--r—
0x644 = 110 100 100

% **chmod 755 a.out** ← Sets a.out to rwxr-xr-x
0x755 = 111 101 101

% **chmod 775 a.out** ← Sets a.out to rwxrwxr-x
0x775 = 111 111 101

% **chmod 400 id_rsa** ← Sets file id_rsa to be readonly, r--------
0x400 = 100 000 000

chmod – change file mode bits

Can also use …

```
chmod ugo +-= rwx filename/directory
```

where

ugo specifies user, group, other
+-= specifies add (+), subtract (-), set (=)
rwx specifies read, write, execute

% **chmod g+w myfile** ← Adds group write access to file myfile.

% **chmod o-w myfile** ← Remove other (world) write access from myfile.

% **chmod g=rwx myfile** ← Sets group to rwx for file myfile.

Signals

```
signal( SIGALRM, timeout);        ← If alarm goes off, call timeout().
alarm (10);                       ← Alarm will go off in 10 seconds.
// Do something.
alarm(0);                         ← Turn off alarm.


void timeout()
{
    printf(“ERROR: Timeout occurred.\n”);
}
```

Signals

The signals currently defined by <signal.h> are as follows:

Name	Value	Default	Event
SIGHUP	1	Exit	Hangup (see termio(4I))
SIGINT	2	Exit	Interrupt (^C)
SIGQUIT	3	Core	Quit (see termio(4I))
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace or Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status Changed
SIGPWR	19	Ignore	Power Fail or Restart
SIGWINCH	20	Ignore	Window Size Change

There are more signals defined too…

Program can ignore or handle signals:

```
sigset(SIGINT, SIG_IGN);
sigset(SIGHUP, SIG_IGN);
signal(SIGSYS, error_seen);
signal( SIGTERM, error_seen);
signal( SIGPWR, error_seen);
signal( SIGILL, error_seen);
signal( SIGFPE, error_seen);
signal( SIGBUS, error_seen);
signal( SIGSEGV, error_seen);
signal( SIGUSR1, SIG_IGN);
signal( SIGUSR2, error_seen);

// Program continues on …
```

fork

- When a process calls fork, it is deemed the parent process and the newly created process is its child.
- After the fork, the parent and child process don't know if they are the parent or child until the process id returned from the fork call is checked.
- fork() call from the parent process will return the process id of the child.
- fork() call from the child process will return 0. (The age of the child is zero.)
- After the fork, both processes resume execution starting at the fork call.
- Based on the return value, the parent and child can perform different functions.

fork example

```
switch (pid = fork()) {
        case -1:
                printf("----> ERROR: Unable to create child process.\n");
                exit(0);
        case 0:
                go_do_child_stuff();
                break;
        default:
               printf("PARENT started CHILD process id %d\n”, pid);
               break;
        }
        … Parent process continues
```

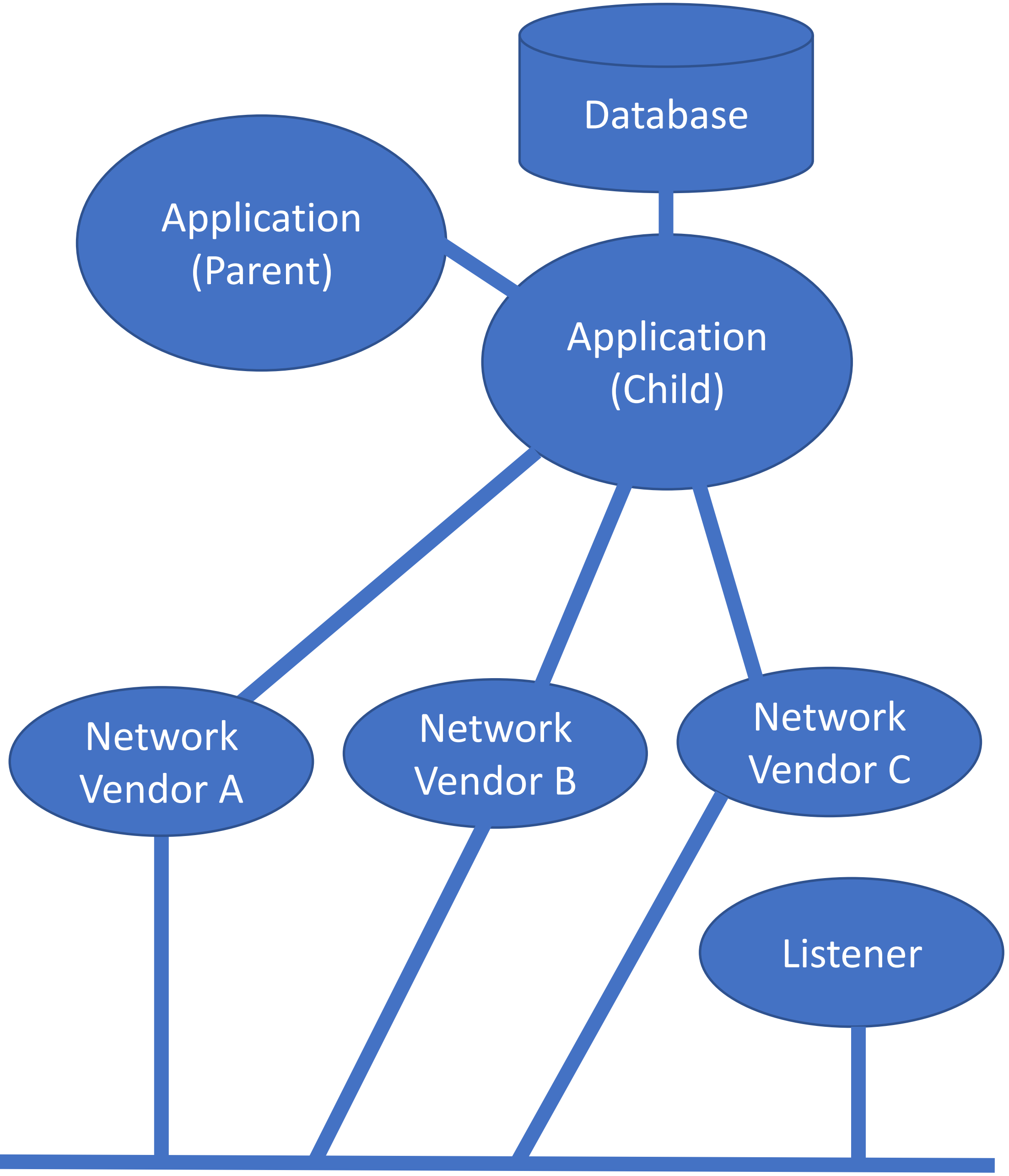
fork Usage

Application Level

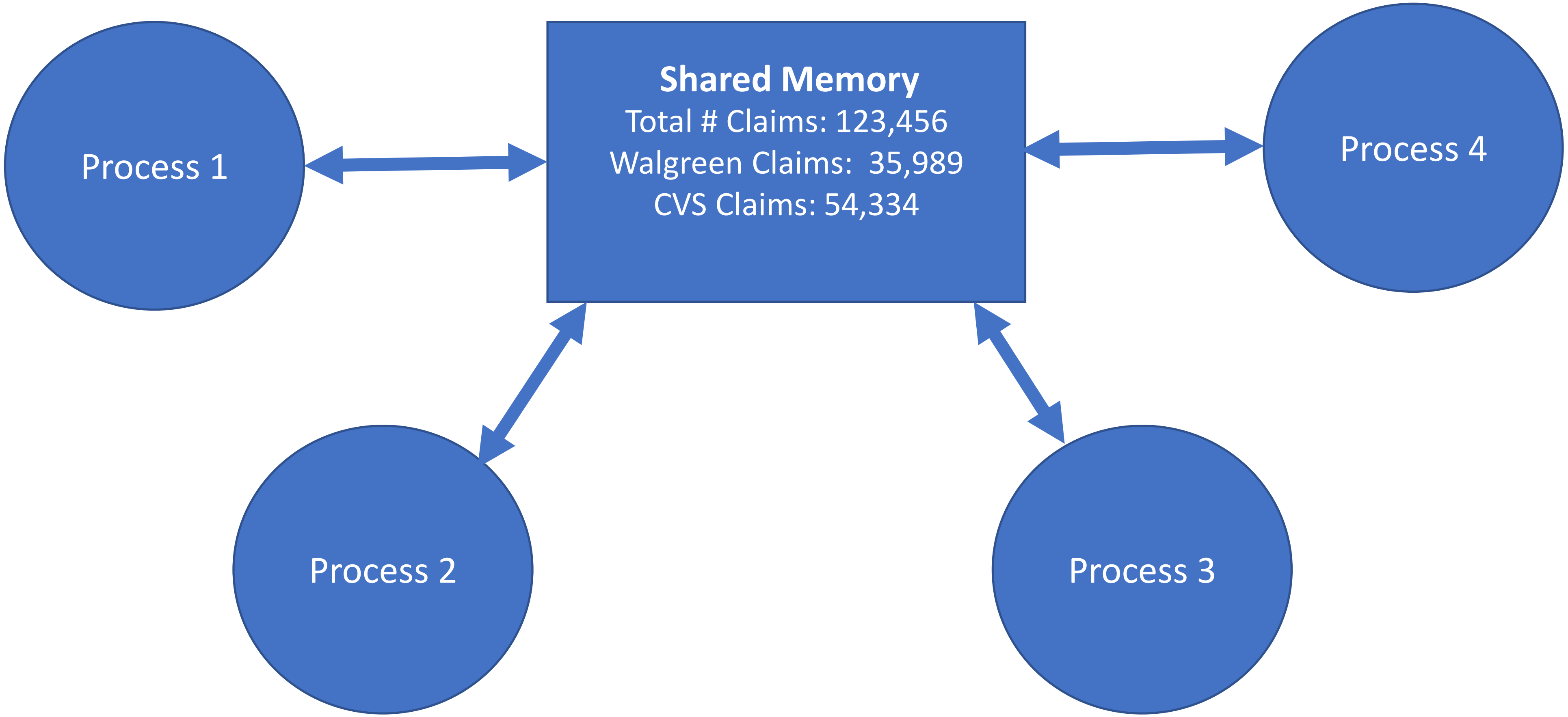
- Start application.
- Parent spawns a child process.
- Parent monitors child.
- If child process dies, parent respawns child process to keep application up and running.

Network Level

- Listener listens to IP Address/Port Number for incoming connection request.
- When a network connection is requested for IP/Port Number, parent spawns a child process.
- Child process handles networking at IP/Generated Port Number.



Shared Memory



Shared Memory
Total # Claims: 123,456
Walgreen Claims: 35,989
CVS Claims: 54,334

Shared Memory Calls - shmget

NAME

shmget - get shared memory segment identifier

SYNOPSIS

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

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

RETURN VALUES

Upon successful completion, a non-negative integer representing a shared memory identifier is returned. Otherwise, -1 is returned and errno is set to indicate the error.

shmat – Shared Memory Attach

NAME

shmop, shmat, shmdt - shared memory operations

SYNOPSIS

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

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

RETURN VALUES

Upon successful completion, shmat() returns the data segment start address of the attached shared memory segment. Otherwise, SHM_FAILED (-1) is returned, the shared memory segment is not attached, and errno is set to indicate the error.

Create Shared Memory Segment

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

#define SHM_SIZE 1024
```

```
int main()
{
    key_t key = 1234;
    int shmid;
    // Create shared memory.
    // Tricky Note: IPC_CREAT = 0x200 hex, 0644 is octal.
    if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1)
    {
        printf("ERROR: shmget failed.\n");
        exit(1);
    }
    return 0;
}
```

Copy b.c to b1.c, b2.c, b3.c. Update memory.

```
#define SHM_SIZE 1024
// Usage: b “text to write into shared memory”
int main(int argc, char *argv[])
{
    key_t key = 1234;
    int shmid;
    char *shm_ptr;
    int mode;

// Get shared memory.
    if ((shmid = shmget(key, SHM_SIZE, 0644)) == -1) {
        printf("ERROR: shmget failed.\n"); exit(1);
    }
// Attach to shared memory. shmat returns pointer.
    shm_ptr = shmat(shmid, (void *)0, 0);
    if (shm_ptr == (char *)(-1)) {
        printf("ERROR: shmat failed.\n"); exit(1);
    }
```

```
    // Read from shared memory.
    printf("Read from shared memory: \"%s\"\n", shm_ptr);

    // Write to shared memory.
    printf("Write to shared memory: \"%s\"\n", argv[1]);
    strncpy(shm_ptr, argv[1], SHM_SIZE);

    // Read from shared memory.
    printf("Read from shared memory: \"%s\"\n", shm_ptr);

    // Detach from shared memory.
    if (shmdt(shm_ptr) == -1) {
        printf("ERROR: shmdt failed.\n"); exit(1);
    }
    return 0;
}
```

Delete Shared Memory

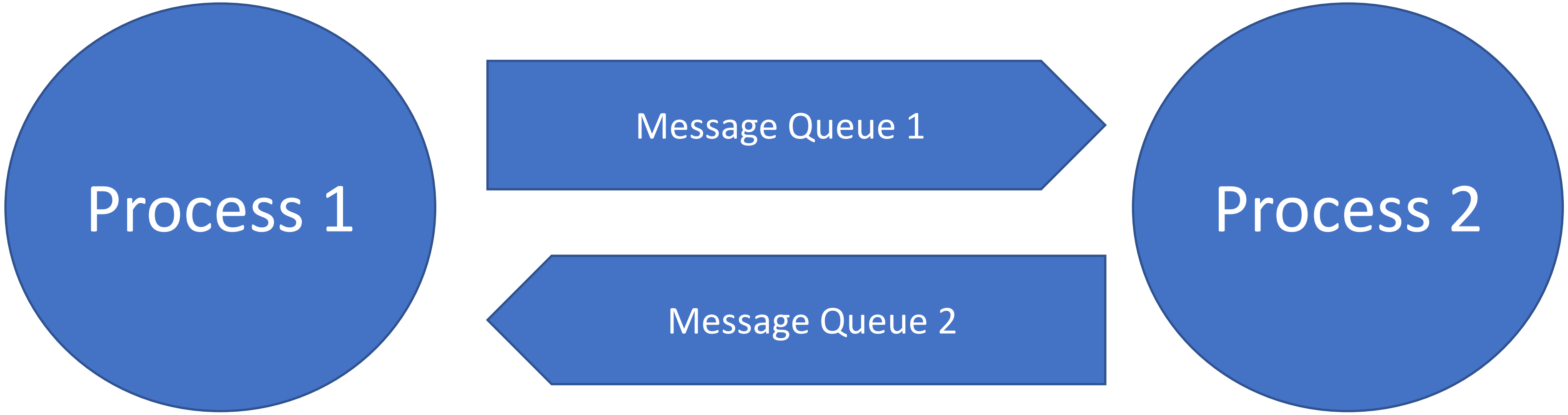
```
#define SHM_SIZE 1024

int main(int argc, char *argv[])
{
   key_t key = 1234;
   int shmid;

if ((shmid = shmget(key, SHM_SIZE, 0644)) == -1) {          // Get shared memory id.
      printf("ERROR: shmget failed.\n");
      exit(1);
   }
   shmctl(shmid, IPC_RMID, NULL);                             // Delete shared memory.
   return 0;
}
```

Sample Run for Shared Memory

```
john@oho:~/SHM$ b1 "Hello World"
ERROR: shmget failed.

john@oho:~/SHM$ create_shm

john@oho:~/SHM$ b1 "Hello World"
Read from shared memory: ""
Write to shared memory: "Hello World"
Read from shared memory: "Hello World“

john@oho:~/SHM$ b2 "This is a test"
Read from shared memory: "Hello World"
Write to shared memory: "This is a test"
Read from shared memory: "This is a test"
```

```
john@oho:~/SHM$ b3 "Shared Memory Works!"
Read from shared memory: "This is a test"
Write to shared memory: "Shared Memory Works!"
Read from shared memory: "Shared Memory Works!“

john@oho:~/SHM$ delete_shm

john@oho:~/SHM$ b1 "Hello again"
ERROR: shmget failed.

john@oho:~/SHM$ ls
b.c  b1  b1.c  b2  b2.c  b3  b3.c  create_shm  create_shm.c
delete_shm  delete_shm.c
```

Message Queues



msgget

```
NAME
    msgget - get message queue

SYNOPSIS
    #include <sys/msg.h>
    int msgget(key_t key, int msgflg);

DESCRIPTION
    The msgget() argument returns the message  queue  identifier associated with key.
```

```
% more c.c
#include <stdio.h>
#include <sys/msg.h>

int main()
{
      int    mq;

      mq = msgget(1000, 0666|IPC_CREAT);          ← Creates the message queue 1000.

      printf("message queue 1000 created.\n");
}
```

msgsnd / msgrcv

NAME

msgsnd - message send operation

SYNOPSIS

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *msgp,  size_t  msgsz,  int msgflg);
```

DESCRIPTION

The msgsnd() function is used to send a message to the queue associated with the message queue.

NAME

msgrcv - message receive operation

SYNOPSIS

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long int msgtyp, int msgflg);
```

DESCRIPTION

The msgrcv() function reads a message from the queue associated with the message queue identifier specified by msqid and places it in the user-defined buffer pointed to by msgp.

msgsnd

```
% cat s.c
#include <stdio.h>
#include <sys/msg.h>

struct my_message {
        long    mtype;
        char    mtext[100];
} my_msg;

int main()
{
        int     mq;
        int     return_val = 0;

        mq = msgget(1000, 0666);

        my_msg.mtype = 100;
        strcpy(my_msg.mtext, "Hello!");

        if ((return_val = msgsnd(mq, &my_msg,
                                  sizeof(struct my_message), 0666)) < 0) {
              printf("msgsnd failed.\n");
        }
        if ((return_val = msgsnd(mq, &my_msg,
                                 sizeof(struct my_message), 0666)) < 0) {
              printf("msgsnd failed.\n");
        }

       my_msg.mtype = 200;
       strcpy(my_msg.mtext, "Bye!");

       if ((return_val = msgsnd(mq, &my_msg,
                            sizeof(struct my_message), 0666)) < 0) {
              printf("msgsnd failed.\n");
       }

       printf("msgsnd worked.\n");
}
```

msgrcv – mtype = 100

```
% cat r100.c
#include <stdio.h>
#include <sys/msg.h>

struct my_message {
        long    mtype;
        char    mtext[100];
} my_msg;

int main()
{
        int     mq;
        int     return_val = 0;

        mq = msgget(1000, 0666);
        if ((return_val = msgrcv(mq, &my_msg, sizeof(struct my_message), 100, 0666)) < 0) {        // Return message with type = 100
          printf("msgrcv failed.\n");
        }

        printf("msgrcv: my_msg.mtype = %d, my_msg.mtext = %s\n", my_msg.mtype, my_msg.mtext);
        printf("msgrcv worked.\n");
}
```

msgrcv – mtype = 200

```
% cat r200.c
#include <stdio.h>
#include <sys/msg.h>
struct my_message {
        long    mtype;
        char    mtext[100];
} my_msg;

int main()
{
        int     mq;
        int     return_val = 0;

        mq = msgget(1000, 0666);
        if ((return_val = msgrcv(mq, &my_msg, sizeof(struct my_message), 200, 0666)) < 0) {        // mtype = 200
          printf("msgrcv failed.\n");
        }

        printf("msgrcv: my_msg.mtype = %d, my_msg.mtext = %s\n", my_msg.mtype, my_msg.mtext);
        printf("msgrcv worked.\n");
}
```

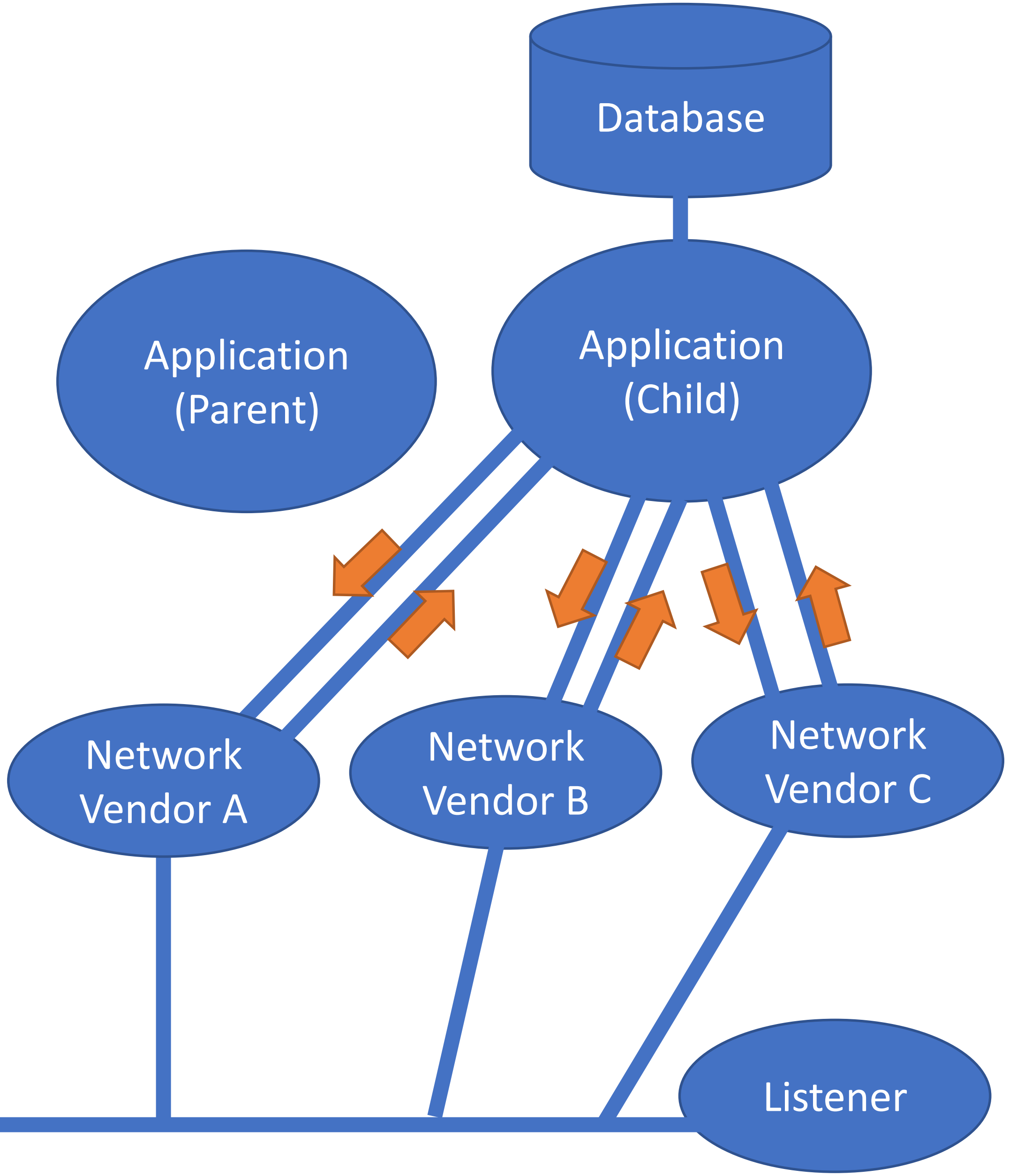
msgget, msgsnd, msgget Example

```
% c
message queue 1000 created.

% s                    ← Puts {100, “Hello!”}, {100, “Hello!”}, {200, “Bye!”} on queue.
msgsnd worked.

%r100
msgrcv: my_msg.mtype = 100, my_msg.mtext = Hello!
msgrcv worked.

%r200
msgrcv: my_msg.mtype = 200, my_msg.mtext = Bye!
msgrcv worked.

%r100
msgrcv: my_msg.mtype = 100, my_msg.mtext = Hello!
msgrcv worked.

%r100                        ← No more 100 message types on queue.  r100 blocks/waits for next 100 type.
^C
%r200                        ← No more 200 message types on queue.  r200 blocks/waits for next 200 type.
^C
```

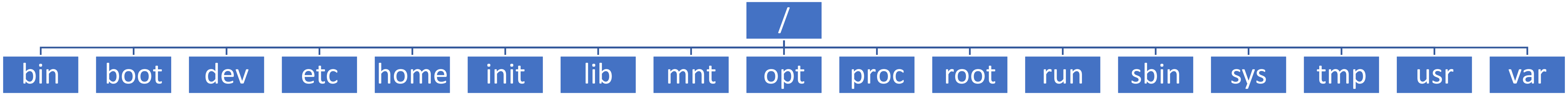
Message Queues

Message queues are used to send incoming transactions from the Network Vendor process to the Application Process.

Separate message queues are used to send responses from the Application to the Network Vendor process.



Unix File System



The top node “/” is called the root directory.

```
john@oho:~$ ls /
bin   dev  home  lib    lib64   media  opt   root  sbin  srv  tmp  var
boot  etc  init  lib32  libx32  mnt    proc  run   snap  sys  usr
```

COMP-232 Programming Languages

/bin and /sbin Directories

/bin contains most Unix commands that you can run.
Examples are:

ls, more, cat, vi

/sbin contains Unix commands for system administration.

Examples are:

adduser, fsck, mkfs, reboot

To locate where a Unix command resides, run: **which <command>**

COMP-232 Programming Languages

/home Directories

←

```
john@oho:/home$ cd            ← Change directory into /home/john
john@oho:~$ ls -a1            ← List all files in one column
.
..
.bashrc
.profile
.ssh
.vim
.viminfo
.vimrc
COMP232
a.out
c.c
```

COMP 232 Programming Languages

/tmp Directory

/tmp contains temporary files which are removed when the system is rebooted.

Everyone on the system can write to /tmp.

Makes for a great place to create/view temporary files, like:

```
% crontab -l > /tmp/crontab.txt
% grep MAX_VALUE *.c > /tmp/max_value.txt
```

COMP-232 Programming Languages

/usr Directory

/usr contains:

```
john@oho:/usr$ ls –F /usr
bin/    include/  lib32/  libexec/  local/  share/
games/  lib/      lib64/  libx32/   sbin/   src/
```

/usr/include is where the actual include files are located, e.g.

```
#include <stdio.h>
#include <stdlib.h>
```

/usr/local allows you to add local binary, include, lib, and src files:

```
john@oho:/usr/local$ ls –F /usr/local
bin/  etc/  games/  include/  lib/  man@  sbin/  share/  src/
```

/usr/local/bin contains programs local to your system and that others in your group can run.

COMP 232 Programming Languages

/etc

/etc contains 186 files and directories. We’ll cover some of these.

/etc contains system configuration files, such as:

hosts
passwd
services

and directories, such as:

X11
fonts
init.d

COMP-232 Programming Languages

/etc/passwd

- **Defines all users on Unix system.**

Format:

```
login_name:password:user_id:group_id:user_name:home_directory:shell
```

where:

Login Name used to log into system.

Password is not used. Passwords are found in /etc/shadow file today.

User Id holds unique numeric value for user.

Group Id holds numeric value for user’s primary group.

User Name is a comment field to store first/last name or application name.

Home directory is user’s home and location after logging in.

Shell defines the default shell user uses after logging in.

COMP-232 Programming Languages

/etc/passwd

```
john@oho:~$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
sshd:x:109:65534::/run/sshd:/usr/sbin/nologin
john:x:1000:1000:John Dempsey:/home/john:/bin/bash
```

**** The above list is a partial list of all passwd entries.***

**← User id 0/Group id 0 is known as the super user/root user.
root user has full access to everything on the system.**

← Login ids do not need to be associated with an actual person.

← nologin says you cannot log into the system as user sys.

← Says mail program has /var/mail as its home directory.

**← Start of regular user accounts. User id 1000, group id 1000,
User John Dempsey, home dir is /home/john, uses /bin/bash**

COMP 232 Programming Languages

/etc/shadow

- File not readable to most users.

Format:

login_name:password:date_of_last_password_change:min_password_age:maximum_password_age:password_warning_period:password_inactivity_period:account_expiration_date:reserved_field

Where:

Login name is user login as found in the /etc/passwd file.

Password is the encrypted password for user.

Date of last password change expressed as the number of days since January 1, 1970.

Minimum password age is the number of days the user must wait before being able to change their password.

Maximum password age is the number of days before user must change their password.

Password warning period is the number of days before user is notified they are reaching the maximum age. 0 no warning.

Password inactivity period is the maximum number of days past the maximum age where user can still login and must then change their password. Empty is no expiration period.

Account expiration date is when user can no longer log into system. Empty or 0 means no expiration.

/etc/group

- **Defines all user groups on the Unix system.**
- **When you log into a Unix system, the passwd file sets your default group.**
- **A user is assigned one “primary group”, but may belong to multiple groups.**
- **Each file and directory is owned by one group on the system.**

Fields:

```
group_name:password:GID:user_list
```

where:

group_name is the name of the group.
password is the encrypted group password or x if not used.
GID is group id number.
user_list contains list of usernames that are members of this group.

```
john@oho:/etc$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
…
admin:x:116:
netdev:x:117:john
john:x:1000:
staff:x:1001:amy,betty,john,miguel,wendy
```

/etc/hosts

The hosts file contains Internet Protocol (IP) addresses. It’s format is:

IP Address   hostname   aliases …

```
john@oho:~$ cat /etc/hosts
127.0.0.1           localhost
127.0.1.1           oho.localdomain oho
192.60.50.10        charlie      prodhost
192.60.50.11        lucy         devhost
192.60.50.15        linus        testhost
143.198.238.179     comp232      comp232.com
142.93.89.28        openhouseon.com
45.55.2.35          plus1se.com  ← If DNS not available, can enter domain names.
```

COMP 232 Programming Languages

/etc/services

- To access a service on a system, you need two things:
  1. **The IP address of the system.**
  2. **The port number of the service enabled on the system.**

- /etc/services defines the port number and the transport protocol (TCP and/or UDP) supported for each known service.

- Officially assigned port number are defined by Internet Assigned Numbers Authority (IANA) at https://www.iana.org.

- Companies can add unassigned port numbers to support local applications.

COMP-232 Programming Languages

/etc/services

There are 413 lines in the /etc/services file. Here are some of the more important services.

```
% cat /etc/services
tcpmux          1/tcp                           # TCP port service multiplexer
echo            7/tcp
echo            7/udp
netstat         15/tcp
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
whois           43/tcp          nicname
tftp            69/udp
finger          79/tcp
http            80/tcp          www             # WorldWideWeb HTTP
kerberos        88/tcp          kerberos5 krb5 kerberos-sec     # Kerberos v5
kerberos        88/udp          kerberos5 krb5 kerberos-sec     # Kerberos v5
pop3            110/tcp         pop-3           # POP version 3
ntp             123/udp                         # Network Time Protocol
imap2           143/tcp         imap            # Interim Mail Access P 2 and 4
snmp            161/tcp                         # Simple Net Mgmt Protocol
snmp            161/udp
```

/etc/services Continued

```
snmp-trap       162/tcp         snmptrap        # Traps for SNMP
snmp-trap       162/udp         snmptrap
mailq           174/tcp                         # Mailer transport queue for Zmailer
xdmcp           177/udp                         # X Display Manager Control Protocol
bgp             179/tcp                         # Border Gateway Protocol
smux            199/tcp                         # SNMP Unix Multiplexer
qmtp            209/tcp                         # Quick Mail Transfer Protocol
z3950           210/tcp         wais            # NISO Z39.50 database
ipx             213/udp                         # IPX [RFC1234]
#
# UNIX specific services
#
exec            512/tcp
biff            512/udp         comsat
login           513/tcp
who             513/udp         whod
talk            517/udp
ntalk           518/udp
route           520/udp         router routed   # RIP
rsync           873/tcp
ftps-data       989/tcp                         # FTP over SSL (data)
ftps            990/tcp
telnets         992/tcp                         # Telnet over SSL
imaps           993/tcp                         # IMAP over SSL
pop3s           995/tcp                         # POP-3 over SSL
```

COMP-232 Programming Languages

/etc/hostname

/etc/hostname contains the name of the host.

```
% cat /etc/hostname
oho

% hostname
oho

% grep oho /etc/hosts
127.0.1.1       oho.localdomain oho
```

COMP-232 Programming Languages

/etc/profile

/etc/profile runs each time you and others login and helps set up your environment.

```
john@oho:/etc$ more /etc/profile
if [ "${PS1-}" ]; then
  if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi
if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
```

/etc/os-release

- os-release provides details on the Unix version you’re using.

```
john@oho:/etc$ more os-release
NAME="Ubuntu"
VERSION="20.04.2 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04.2 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
```

COMP 232 Programming Languages

/etc/timezone

- Displays what time zone system is using.

```
john@oho:/etc$ more /etc/timezone
America/Los_Angeles
```

COMP 232 Programming Languages

/etc/resolv.com

Lists the IP addresses for Domain Name Servers (DNS) to resolve domain names, like https://plus1se.com

```
john@oho:/etc$ cat resolv.conf
nameserver      192.168.1.1              ← IP Version 4 Format
nameserver      2001:1998:f00:1::1       ← IP Version 6 Format
nameserver      2001:1998:f00:2::1
search          lan     example.com
```

COMP-232 Programming Languages

crontab

```
john@oho:~/LAB4/STOCKS$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# m h  dom mon dow   command
30 17 1 * *     /home/john/LAB4/STOCKS/run_report.bash 2024
```

To edit crontab using vi, type:

```
% export EDITOR=vi
% crontab –e
```

To view crontab, type:

```
% crontab -l
```

COMP-232 Programming Languages

shmflg = 0644 | IPC_CREAT

To create a shared memory id, we can use:

```
if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1)
```

But wait, IPC_CREATE is defined as:

```
#define IPC_CREATE  0x200
```

So there is no change?

We want:

```
if ((shmid = shmget(key, SHM_SIZE, 01644)) == -1)
```

Answer

```
if ((shmid = shmget(key, SHM_SIZE, 0644 | IPC_CREAT)) == -1)
```

0644 is in octal.
IPC_CREAT is 0x200, but 0x200 is in hex.

0644 = 000 110 100 100
0x200 = 0010 0000 0000

0644	0	0	0	1	1	0	1	0	0	1	0	0
0x200	0	0	1	0	0	0	0	0	0	0	0	0
01644	0	0	1	1	1	0	1	0	0	1	0	0

So there is a change and

01644 = 0644 | IPC_CREAT

Root Directories

/bin	Binaries.	Contains 1,103 binary commands most of which you can run.
/dev	Devices.	Contains device definitions.
/etc	Ecetera.	Contains system configuration files, like password and hosts file.
/home	Home.	Contains user home directories.
/lib	Libraries.	A link to /usr/lib
/mnt	Mount.	Device mount points, e.g. C drive mount point.
/opt	Optional.	Can contain optional files and directories. Third party software.
/proc	Processes.	Contains process information.
/sbin	System Binaries.	Contains system binaries some which you can run.
/tmp	Temp.	Contains temporary files which are removed on system reboot.
/usr	User.	User System Resources (USR) directory.
/var	Variable.	Contains variable length files.

COMP 232 Programming Languages

/etc/sudoers

- **Lists users who can use sudo to run commands a super user (su).**

```
john@oho:/etc$ sudo cat sudoers
[sudo] password for john:
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/bin"

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

#includedir /etc/sudoers.d
```

```
john@oho:/etc$ grep admin /etc/group
admin:x:116:
john@oho:/etc$ grep sudo /etc/group
sudo:x:27:john
```

COMP 232 Programming Languages