## process

### general:

process & files

bash as a process

linux kernel

functions & commands

(API)

system calls

wrapper

kernel

kernel activities

user/groups process/threads files/directories

### terminology:

pid

every process

unique at any point of time

ppid

parent

init

first process to run

(or systemd)

mostly pid = 1

### files:

modes:-

r : read

w : write

x : execute

training.xlsx

r everyone

w training team

x prem

rwx rw- r--

prem training everyone

user group others

111 110 100

7 6 4

110 100 110

6 4 6

umask

max: rw- rw- rw-

6 6 6

2 2 2

4 4 4

umask 222

7 7 7

2 2 2

5 5 5

r-x r-x r-x

7 5 5

type:

- regular files

d directory (folder)

p pipes (fifos)

b block driver

c character driver

s sockets

l links

inode number:

unique identifier

uniqueness within file system (different partitions)



file desc

(temporary number for every file opened in the memory = RAM)

lowest available number

process opens:

stdin 0

stdout 1

stderr 2

### system calls:

getpid()

getppid()

open()

close()

write() (printf uses this)

read() (scanf, fread etc use this)

create()

pipe (popen uses this)

msgget()

## states:

R running

S sleep

wait

T suspended

stop

I (kernel thread, process, service)

D uninterruptible

Z zombie

W paging

X dead (won't be seen by ps)

+ foreground to display

l multi threaded

probably CLONE\_THREAD

s session leader

< high priority

L memory lock

jobs:

process in background of shell:

jobs [1] [2]

fg

man pages:

1 commands (executables)

2 system calls

3 functions

4 special files (/dev)

5

6 games, entertainment

7 signals, protocols, miscellaneous

8 system based commands (mount etc)

## signals:

kill : used to send signals

-9 (force)

signal:

handle in 3 ways:

SIG\_IGN ignore

SIG\_DFL default behaviour

handler custom handler

can not be handled:

SIGKILL

SIGSTOP

keyboard signals:

ctrl + C SIGINT

ctrl + Z SIGSTOP

ctrl + \ SIGQUIT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| signal | terminate ? | keyboard? | handle? | core dump? |
| SIGTERM | yes | no | yes | no |
| SIGINT | yes | ctrl + C | yes | no |
| SIGKILL | yes | no | no | no |
| SIGQUIT | yes | ctrl + \ | yes | yes |
| SIGSTOP | no (stops) | ctrl + Z | no | no |
| SIGSEGV | yes | no | yes | yes |
|  |  |  |  |  |

user defined signals:

SIGUSR1

## process Memory:

1. code
2. data
   1. stack
   2. heap
   3. static
3. code (text)
4. static
5. stack
6. heap

## fork:

return value:

0 to child

child’s pid to parent

## IPCs:

### primitive

pipes

fifos

### pipes:

unidirectional

pipe(pd)

pd[0] read end

pd[1] write end

data once read is deleted

lseek wont work

only related process can use it for communication

data is in stream of characters (no boundaries)

### fifo:

named pipe

creation:

mkfifo()

mknod()

mkfifo

data once read is deleted

lseek wont work

unrelated process can use it for communication

data is in stream of characters (no boundaries)

multiple process can get attached

bsd (berkley)

### sys V

message queues

shared memory

semaphores

### POSIX:

message queues

shared memory

semaphores

mutex

### Message Queues :



general sysV IPCs:

key = ftok(“filepath”, num)

id = xxxget(key, , )

msgget

semget

shmget

msgsnd() / msgrcv()

msgctl()

waiting cases:

trying to read from empty queue

trying to read a particular mtype (which is not present)

trying to write in a queue which is full

IPC\_NOWAIT

ipcs

-q msg queue

-s semaphores

-m shared memory

-l limits

ipcrm

with id:

-q msg queue

-s semaphores

-m shared memory

with key:

-Q msg queue

-S semaphores

-M shared memory

### POSIX

### shared memory

broadcasting

### POSIX:

### semaphores:

## kernel

kthreadd

### daemon:

kswapd:

flush:

init:

cron:

atd:

sshd:

features:

most daemons will have root privileges

no controlling terminal

most of the user based dameons are only process in their session

most kernel daemons launched without terminal

all user daemons are disconnected to terminal after launching

daemons can be launched based on conditions

steps:

create child

kill (or exit) parent

create new session by setsid

change current working directory

close all open file descriptors

open system log

write into it

close it

## threads:

parallel execution

lwp (light weight process)

pthreads

joinable

joined

detached

pthread\_t

phtread\_create

pthread\_join

pthread\_self

pthread\_detach

pthread\_exit

pthread\_kill

pthread\_mutex\_wait

pthread\_mutex\_trywait

pthread\_mutex\_timedwait

### History:

GE PDP7

AT&T (BELL)

Ken Thompson

Dennis Ritchie

Brian Kerningham

PDP 11

MULTICS

UNICS

UNIX C

SUN solaris

HP hp-ux

IBM aix

Linus Torvalds

linux (kernel)

Berkley BSD

sys V

IEEE: POSIX

1a OS

1b RTOS

1c threads

## Linux architecture:

open source

free

variations = distributions

Kernel

set of imp features

special space

special mode



### Linux Kernel Modules (lkm):

adding custom codes:

1. add code to source & recompile the kernel
2. add code like a module (lkm)

need:

1. device driver
2. system calls
3. special interrupt (ISR)
4. file system

information (module):

1. License

GPL, BSD, etc..

un understood license can taint the kernel

1. author

firm’s name

your name

1. module description
2. version

tainting:

-- signature verification was required

-- non GPL compatible

-- staging drivers

part of kernel source code

not yet completely tested

-- out-of-tree

-- machine check exceptions

-- multiproc kernel on a uniproc machine

passing arguments:

Module parameters macros

module\_param()

module\_param\_array()



### Major & minor numbers:

allocation :

static allocation

register\_chrdev\_region()

dynamic allocation

alloc\_chrdev\_region()

unregister (de alloc)

unregister\_chrdev\_region()

### Device Files:

forceful (manual)

mknod /dev/bcs\_device c 224 0

mknod -m 644 /dev/bcs\_device c 224 0 (permissions, filename, type, major, minor)

auto

/sys/class/

### udev:

device manager

cdev

file\_operations

## 

## file systems

ext4

iso9660 (cd rom)

procfs

VFS virtual file system

/proc

1.3 water 2 4 3 13

0.4 oil 1 1 1 4

0.8 milk 1 2 2 8

1L 2.5 4 (wastage = 1.5)

0.4L 2.5 7 (wastage = 0.3)

0.5L 6 (wastage = 0.5)

0.1L 25 (wastage = 0)

inventory1:

water bottle1, bottle2

oil bottle3

milk bottle4

inventory2:

bottle1 water

bottle2 water

bottle3 oil

bottle4 milk

journaling

block size 4kb (4096)

1gb, how many blocks = ?

262144 blocks

file\_abc 654200

inode data struct (generally 1 block)

metadata,

block addresses (2067, 2068, 2069)

### links:

hard link:

## general:

cow

copy-on-write

tasks (assignments):

1. kill () to send a signal to another process
2. kill yourself
3. explore alarm (kill yourself)
   1. handle the signal
4. on response from a signal, terminate your process in 10 seconds
5. three children from 1 parent (no grand children)
6. check if parent waits for any child
7. use wait in parent & make it wait
   1. does wait() wait for one child or all children
   2. wait() waits for first child to get created or first child to exit
8. create
9. file copy using system calls
10. are signals & signal handlers shared between parent & child
11. are open files & file desc shared between parent & child
12. are the internal cursors also shared ?
13. chat application (fifos)
    1. fork

or

* 1. threads

1. IPC\_NOWAIT
2. sem\_trywait

kernel:

1. create your own module
2. create a module with dynamic allocated major & minor number

#include<linux/kdev\_t.h>

parked for future discussion :

what is the difference between ext4 or VFS etc and Root FS?

hidden chars in text

select()

nohup & SIGHUP

gdb

stack of threads

## Embedded Linux

uc-linux

emb-linux

x86

ARM

IBM/motorolla power PC

MIPS

Hitachi

open source projects;

buildroot

busybox

openembedded

yocto

build

custom build

hardware porting

debug

driver development

bootloader

## Build steps :

### prerequisite :

80 GB of free hard disk space

any supported linux distribution (Fedora, openSUSE, CentOS, Debian, Ubuntu)

The following packages need to be pre-installed:

git 1.8.3.1 or greater

tar 1.28 or greater

python 3.5.0 or greater.

gcc 5.0 or greater.

Note: you can check the version by

$ git --version

$ tar --version

### Build host packages:

Let us update our current installation packages by:

$ sudo apt-get update

We need a bunch of packages because the build steps will have multiple dependencies.

The following command will install the dependency packages:

$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib build-essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev pylint3 xterm python3-subunit mesa-common-dev

### cloning the poky repository:

$ git clone git://git.yoctoproject.org/poky

You can move to poky directory and check out the tags

$ cd poky

$ git fetch --tags

$ git tag

### building the image:

The build process is required to create an entire Linux distribution. This will include all the required toolchains too. Move into the poky directory

$ cd ~/poky

run the environment set-up script to define Yocto project’s build environment

$ source oe-init-build-env

along with other folders, a build directory will be created

after running of the previous command (source oe-init-build-env), the script will automatically change the directory to build directory

### build an OS image for the target:

We are using core-image-sato in this example.

$ bitbake core-image-sato

We can test the build image in a simulator. Let us use qemu for the same.

this has to be done from the build directory.

$ runqemu qemux86-64

The above will open up a new window with a new qemu (emulator).

It boots up Linux that we have just built in the emulator.

the local.conf inside conf directory has options for various machines.

we have used

MACHINE ??= “qemux86-64”

(its by default)

### Build an Image for Raspberry Pi:

most of the initial steps would be similar.

after we have build a simple sample image, we need to clone the meta raspberry pi layer.

$ cd ~/poky

$ git clone git://git.yoctoproject.org/meta-raspberrypi

Then Initialize the build environment  
$ source oe-init-build-env

open the bblayers.conf folder and add the meta-raspberrypi to BBLAYERS

The poky address in my system is /home/nigam/poky, but will vary in your case

BBLAYERS ?= " \

/home/nigam/poky/meta \

/home/nigam/poky/meta-yocto \

/home/nigam/poky/meta-yocto-bsp \

/home/nigam/poky/meta-raspberrypi \

"

Open conf/local.conf and change MACHINE to whatever hardware you have chosen. In this specific case it would be raspberrypi3

MACHINE ??= "raspberrypi3"