# I, linux OS

Basic features: print
variables + environment
Math
File description
Array
Alias, function, IFS, debug
Useful commands: edit in file
find file
xargs
translate
checksum and verification
sort
split
rename and moving file
File: create and edit
find and delete
permissions, ownership
head and tail
printing tree
Text and driving: Text processing
Backup plan: Archiving
Compressing
Git

# II, Vi editor

vi is generally considered the de facto standard in Unix editors because –

It's usually available on all the flavors of Unix system.

Its implementations are very similar across the board.

It requires very few resources.

It is more user-friendly

Command mode

Insert mode

# III,Timer handling

## Thread:

- ➔ thành phần của tiến trình
- ➔ 1 tiến trình có thể chứa nhiều thread
- ➔ các thread sử dụng vùng nhớ của tiến trình cha và chia sẻ bộ nhớ với nhau.(global memory)
- ➔ số thread bị giới hạn phụ thuộc vào tiến trình cha.

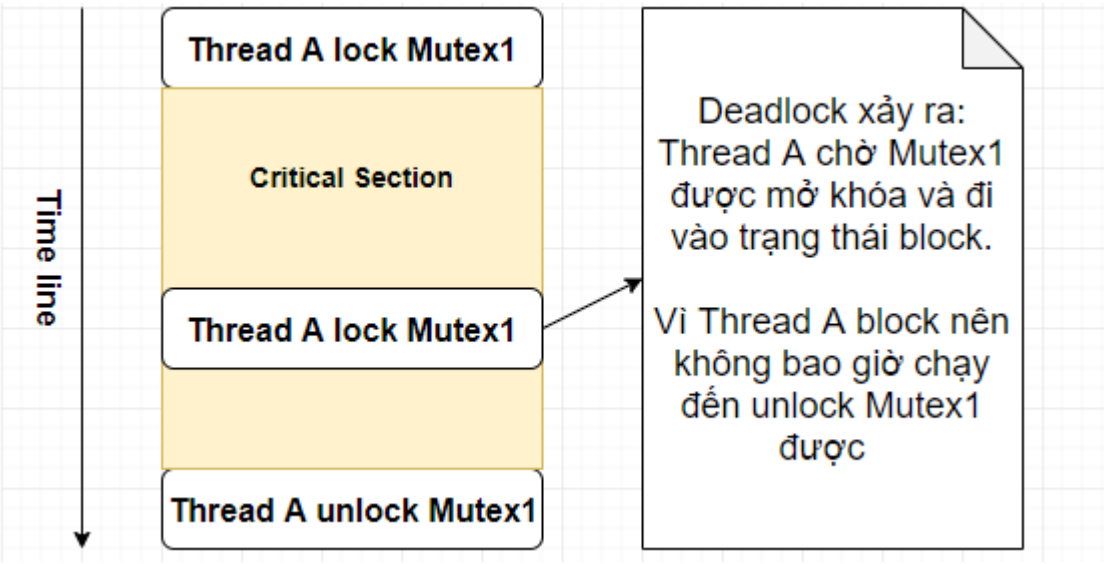
| Kiểu dữ liệu | Mô tả |
|---|---|
| pthread_t | Số định danh của thread (threadID) |
| pthread_mutex_t | Mutex |
| pthread_mutexattr_t | Thuộc tính của mutex |
| pthread_cond_t | Biến điều kiện |
| pthread_condattr_t | Thuộc tính của biến điều kiện |
| pthread_key_t | Khóa cho dữ liệu của thread |
| pthread_attr_t | Thuộc tính của thread |

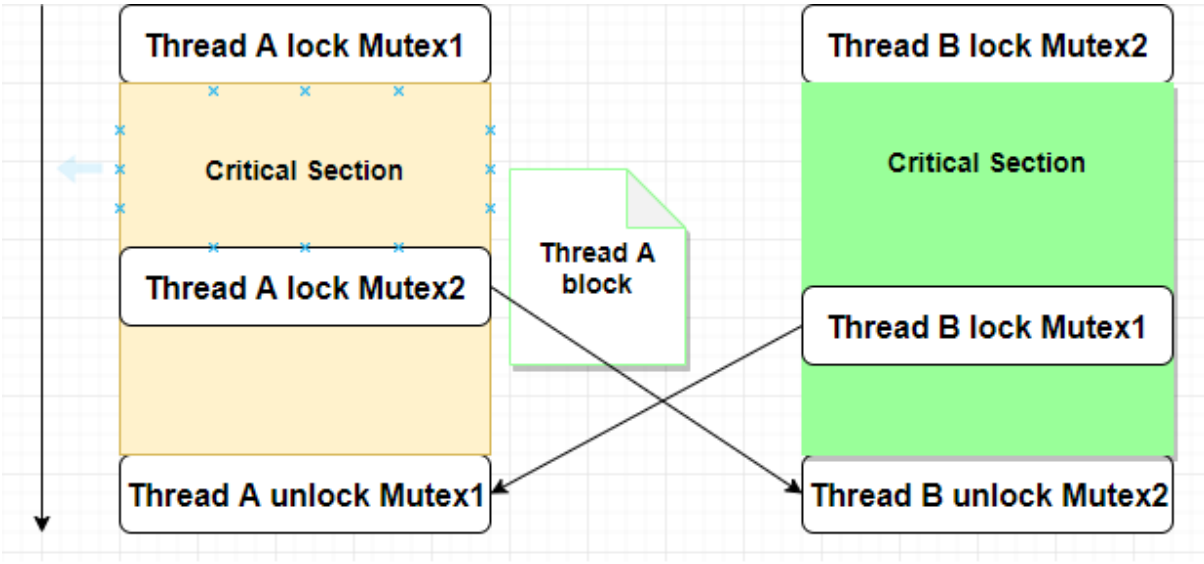
## Threads Management:

- ➔ tạo mới
- ➔ joinable thread (mặc định): khi gọi hàm join(), sẽ block thread và chờ cho thread kết thúc và trả về giá trị. nếu thread đã kết thúc trước khi gọi join() thì sẽ trả về ngay lập tức và thu hồi tài nguyên..
  - ◆ Gọi 2 lần hàm join() sẽ gây ra lỗi.
  - ◆ có thể gọi trong bất kỳ thread nào trong tiến trình
- ➔ detached thread (tự thu hồi tài nguyên khi kết thúc):
  - ◆ không thể truy vấn trạng thái kết thúc.

Mutex: dùng để bảo vệ tài nguyên chia sẻ giữa các thread. hoạt động giống như khóa cửa. Trước khi sử dụng tài nguyên thì khóa lại không cho các thread khác truy nhập vào, sau khi xong thì mở khóa.

- ➔ chỉ có thread nào lock mutex thì mới có thể unlock
- ➔ hủy mutex khi không sử dụng với hàm destroy()
- ➔ Vấn đề với Deadlock: gây treo hệ thống




- giải quyết bằng trylock()

## Semaphore:

- ➔ quản lí tài nguyên dùng chung giữa các thread
  Binary semaphore (0 hoặc 1)
  Counting semaphore (n)
- ➔ post (tăng giá trị cho semaphore thêm 1)
- ➔ wait (khi semaphore != 0 thì giảm 1 nếu =0 thì đợi cho đến khi giá trị khác 0)
- ➔ destroy (hủy semaphore khi không có thread nào đang sử dụng)

| semaphore | mutex |
|---|---|
| giá trị có thể khác 0 và 1 | chỉ có thể là 0 hoặc 1 |
| semaphore thường sử dụng làm phần tử để đồng bộ dữ liệu giữa các luồng. | mutex được dùng để tránh tình trạng nhiều luồn truy nhập vào 1 vùng tài nguyên. |
| có thể bị thay đổi giá trị từ các thread khác | chỉ bị thay đổi bởi thread đang sử dụng. |
| có thể xảy ra tình trạng đảo ngược ưu tiên | không thể |

## Thread cancellation:

- ➔ pthread_exit: kết thúc mà kernel không tự thu hồi tài nguyên.
- ➔ pthread_cancel: dọn dẹp tài nguyên trước khi kết thúc thread.

# IV, IPC

## 1, Semaphore

- ➔ quản lí tài nguyên dùng chung giữa các process
  Binary semaphore (0 hoặc 1)
  Counting semaphore (n)
- ➔ khi giá trị semaphore > 0 thực hiện tiến trình vào giảm semaphore đi 1. khi thực hiện xong sẽ đưa semaphore +1.
- ➔ nếu semaphore = 0 đưa tiến trình vào hàng đợi và trang thái ngủ. tiến trình sẽ quay lại khi có tiến trình khác trả lại semaphore.

## 2, share memory

- ➔ chia sẻ bộ nhớ giữa các tiến trình
- ➔ các tiến trình có thể truy nhập vào bộ nhớ này để trao đổi thông tin
- ➔ tồn tại độc lập với các tiến trình

tạo vùng share → gắn vùng share vào vùng bộ nhớ của tiến trình → trao đổi dữ liệu → tách vùng share ra khỏi tiến trình → hủy vùng share( chỉ hủy khi ko còn tiến trình nào sử dụng)

- ➔ nhược điểm về đồng bộ hóa.

## 3, message queue

- ➔ dữ liệu truyền đi có kiểu cụ thể
- ➔ bị giới hạn kích thước queue
- ➔ không nhất thiết FIFO.

# V, Socket

IPv4 : struct sockaddr_in { }
IPv6 : struct sockaddr_in6 { }