class

<https://cs.nyu.edu/~mwalfish/classes/21fa/index.html>

setup & tools

<https://cs.nyu.edu/~mwalfish/classes/21fa/setup.html>

操作系统导论

<https://weread.qq.com/web/reader/db8329d071cc7f70db8a479k636320102206364d3f0ffdc>

--

wiki:

Vagrant is an open-source software product for

building and maintaining portable virtual software

development environments

It tries to simplify the software configuration

management of virtualization

Vagrant是一款用于构建及配置虚拟开发环境的软件，

基于Ruby,主要以命令行的方式运行

主要使用Oracle的开源VirtualBox虚拟化系统

--

vagrant commands must be run from the directory

where you cloned the base-image repository

list of commands that you can use with Vagrant:

-vagrant up: Start the VM, provisioning it if necessary.

-vagrant suspend: Suspend the VM, if want to

regain memory or cores not working on the class.

oct 27, midterm

**PROCESS** **进程** (OSTEP 4)

1. a program that is being executed by one or

or many threads

2. a **running program**; program itself is a lifeless

thing: just sits there on disk

instructions waiting to spring into action

OS takes these bytes and gets them running

2.1. at any time, the process can be described by

its state, contents of memory in its address space,

contents of cpu reg (including program counter

and stack pointer),

3. a typical system may be seemingly running tens or even hundreds

of processes at the same time

4. time sharing 分时系统 of CPU:

allow users to run many concurrent processes,

cost is performance

5. Processes are a more sound choice

for logically separate tasks where little

sharing of data structures in memory is needed (OSTEP 26)

**THREAD 线程** (OSTEP 26)

1. smallest sequence of programmed instructions

that can be managed indep by a scheduler

操作系统能进行运算调度的最小单位

2. multi-threaded program has more than 1

point of execution

3. state of single thread is similar to process:

program counter

each thread has own private set of registers

2 threads on single processor

switch from T1 to T2, context switch take place

4. each thread runs indep, 1 stack per thread

5. stack-allocated variables, para, ret values,

put on stack, placed on THREAD-LOCAL storage

线程局部存储 (stack of the relevant thread)

6. WHY?

a. PARALLELISM, speed up a process

considerably y using processors to

each perform a portion of work

b. AVOID BLOCKING PROGRAM

PROGRESS DUE TO SLOW I/O

due to SLOW I/O: instead of waiting,

program may wish to do sth else

(utilizing CPU to perform computation...)

**VIRTUAL ADDRESS SPACE 虚拟地址空间** (OSTEP 4)

0. wiki: VIRTUAL ADDRESS SPACE is a set of

ranges of virtual addresses that an OS makes

available to a process

0.1. benefit = process isolation assuming each

process is given a separate address space

1. memory that the process can address

(ADDRESS SPACE) is part of the process

地址空间是某个特定进程在存储器中所能够使用与控

制的地址区段

2. to run a program, OS load program code

and static data (initialized var) into memo, into

the ADDRESS SPACE of process

3. abstraction of address space enables each

program to behave as if it has its own memory

when indeed OS is secretly multiplexing address

spaces across physical memory (sometimes disk)

null char not included in char array

char name[5] = "alice"

fork 复刻 派生 分支

https://www.geeksforgeeks.org/fork-system-call/

Process vs Thread?

The primary difference is that threads within

the same process run in a shared memory space,

while processes run in separate memory spaces.

Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

<https://www.geeksforgeeks.org/thread-in-operating-system/#:~:text=Threads%20are%20not%20independent%20of,register%20set%2C%20and%20stack%20space>.

Mutex, lock, program

<https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/>

condition variables:

<https://web.stanford.edu/~ouster/cgi-bin/cs140-spring14/lecture.php?topic=locks>

**Lock**:

**acquire**(&mutex) mark the lock as owned by the current thread; if some other

thread already owns the lock, then first wait until the lock is free.

Lock typically includes a queue to keep track of multiple waiting threads

**release**(&mutex) mark the lock as free

REVIEW THE PART OF TEXTBOOK THAT IS MENTIONED IN CLASS