# Introduction

## Two fundamental protocols of programming:

---ISO (International Standard Organization)

---POSIX (Portable operation system of unix)

## Principles

Encapsulation、Polymorphism、Inheritance

## Type theory

Void

normally, keyword void is being used to describe type of result a function returned. Specially, void\* is used to annotate pointers to data of an unspecified type.

## MMU (Memory Management Unit)

### PA (Physical Address, Real Address)

Unaligned memory address

(reference:https://www.cs.umd.edu/~meesh/cmsc411/website/projects/outer/memory/align.htm、https://en.wikipedia.org/wiki/Physical\_address)

### VAS (Virtual Address Space)

## Instructions Keywords 指令关键字

Instructions Keywords will not be ignored by compiler, and access the value directly.

Instructions keywords are not data type, just 修饰符

Automatic、register、 volatile

### Automatic

Automatic variable is a local variable which is allocated and deallocated automatically when program flow enters and leaves the variable's scope. The scope generally means function or block scope. Its values are mainly stored in the stack frames

### Register

Reference： <https://baike.baidu.com/item/Register>

Variables of register type that are often used will be put into register (寄存器)

### Volatile

Variables that declared by using volatile, will be stored in memory.

For example:

XBYTE[2]=0x55;

XBYTE[2]=0x56;

XBYTE[2]=0x57;

XBYTE[2]=0x58;

The first three instructions will be removed by compiler (called compiler optimization) because compiler could distinguish the first three instructions are invalid operations. However if using volatile, four instructions will be kept.

# 基本函数

# 3. Memory内存

GNU C中一般对于进程分配的内存空间大小为3G。程序运行时， 只会分配给部分内存（小于3G）, 当进程需要的内存空间增大时，此时会调用calloc、realloc方法进行内存分配。

## 3.1 Memory allocation内存分配

### 3.1.1 common-used function in gcc

void \*malloc(int size): allocate designated size of memory

void \*calloc(obj, offsets): allocate extra size of memory for one designated object

void \*realloc(area, newSize): allocate new size of memory for one area

***Note:***

When using Malloc to allocate memory, remember to use free to gc unused memory, otherwise, it could cause memory leakage.

When using free to release memory, if the pointer has already been freed, or the pointer is not obtained by the three function above, it would case fatal problems.

### 3.1.2 Allocation Algorithms

#### 3.1.2.1 Quick-fit algorithms

Pros:

Cons: cause more memory

#### 3.1.2.2 TCMalloc

using thread-local cache to allocate memory.

Pros: avoid locking overhead

#### 3.1.2.3 Alloca

using stack frame of current function, instead of heap memory.

Pros: no need to care about release of memory, it will release memory automatically when the function returns.

### 3.1.3 Goto & branching

#### 3.1.3.1 setjmp

to record the location the function should return

#### 3.1.3.2 longjmp

to restore the status of stack

## 3.2虚拟内存/物理内存

sbrk 实现虚拟内存到物理内存的映射.

## 3.3 Memory layout

#### 3.3.1 Introduction

\* Text segment: machine instructions, which is sharable and immutable

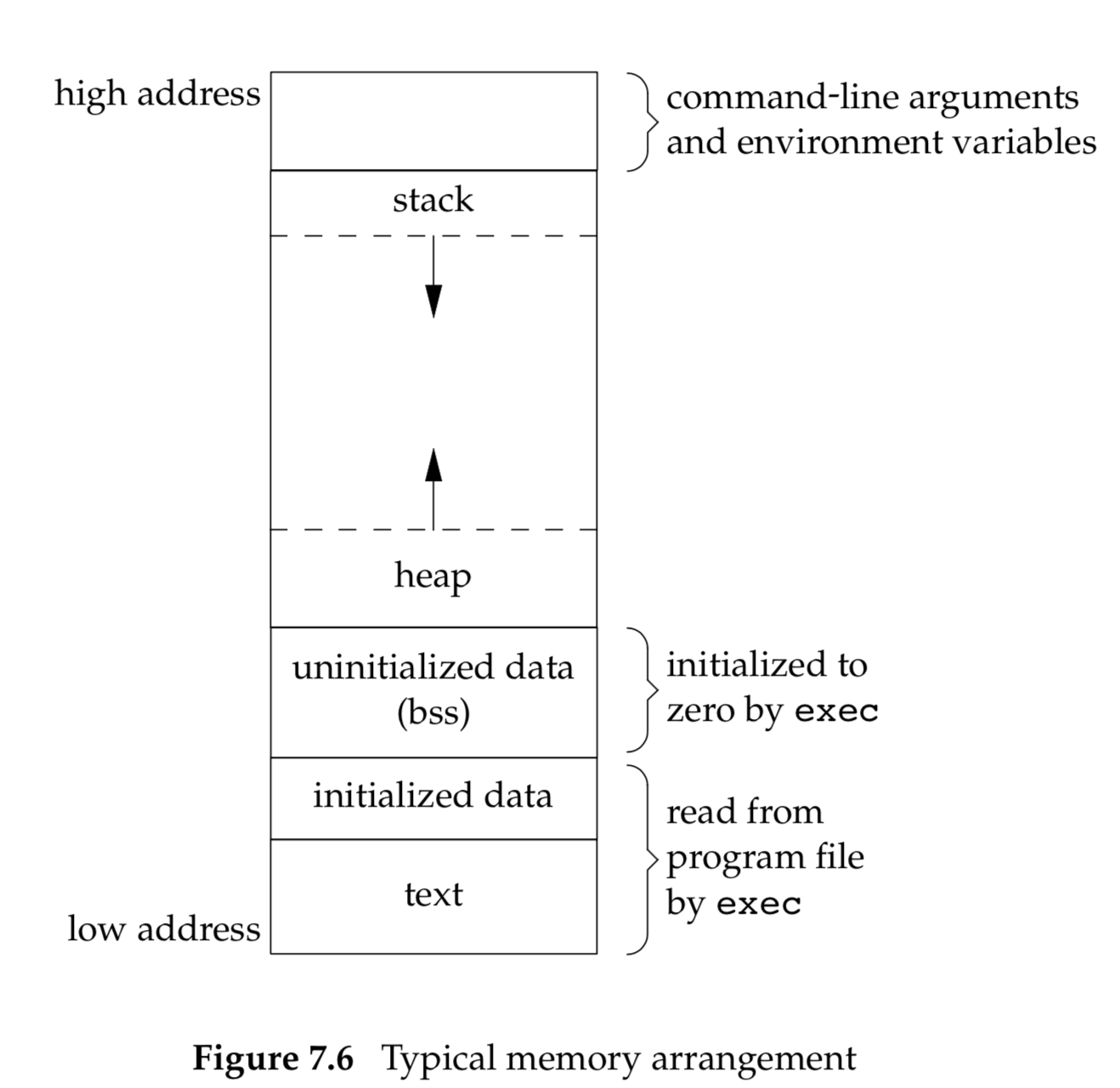
\* Data segment:

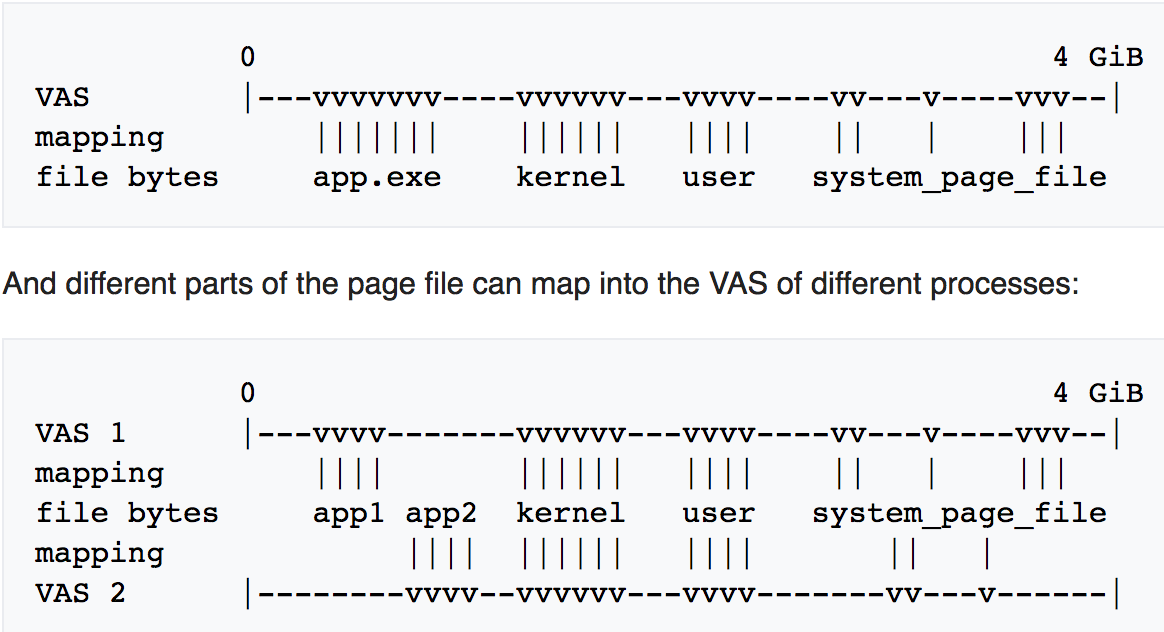
---initialized data segment: statically allocating memory

---uninitialized data segment: initialize data to null or 0 by the kernel（bss）. BSS is short for Block Started by Symbol Segment, which is used to store uninitialized data global variables. And bss belongs to Static memory allocation. For example, after compiling c programs, initialized data will be stored in .data segment, and uninitialized data will be stored in .bss segment. Note, bss data won’t be in executable file.

\* Stack: automatic variables、 caller environment、addresses where to return

\*Heap: Dynamic memory allocation





#### 3.3.2 type of variable in c (storage class)

Reference: <http://os.camden.rutgers.edu/c_resources/c_manual/C/CONCEPT/storage_class.html>

#### 3.3.2.1 auto

used to decorate local variables

#### 3.3.2.2 register

to allocate variable in register memory rather then RAM.

#### 3.3.2.3 const

read only variables. Variable of this type will be allocated in shared memory of different processes to reduce memory usages. Meanwhile, for the same string variables, it will also be stored only once in shared memory

#### 3.3.2.4 static

Functionality:

---limit the scope

---change life time.

Decorating global variable:

---extend its scope

Decorating local variable:

---change life time. Because when function returns, automatic variable will disappear with the function, returned values will be overwritten. When using static, local variable will exists.

## 3.4 System design

### 3.4.2 Shared library

isolate some common share library (called shared library) from executable file, and put it in memory that all processes reference.

Advantage: reduce the size of executable file.

### 3.4.3 Memory Circle

3.4.3.1 Definition

Reading the data out of the memory and/or writing data into memory.

\*\*Reading the data

\*\* Writing the data

\*\*Reading the data first, writing the data later

## 3.5 Physical Equipment

### 3.5.1 Memory Address Register (MAR)

### 3.5.1 Definition

寄存器, also called memory register, memory translator, memory decoder, stores mapping between physical memory addresses and logical memory addresses.

The addresses sent by CPU or I/O are logical memory address and temporarily stored in address register.

### 3.5.2 Types

#### 3.5.2.1 Instruction Register (IR、指令寄存器)

#### 3.5.2.2 Procedure Counter (PC, 程序计数器)

# 4. 进程—Process

## 4.1 introduction

### 4.1.1 typical memory layout of a process.

### 4.1.2 stack、 heap、

### 4.1.3 environmental variable

### 4.1.4 termination of a process

### 4.1.5 special process id

#### 4.1.5.1 value 1

In linux, the first process created by linux kernel is assigned 1 as it’s pid。(Which is init process)

Process ID 1 is usually the init process primarily responsible for starting and shutting down the system.

#### 4.1.5.2 value 0

swapper or sched has process ID 0 and is responsible for paging, and is actually part of the kernel rather than a normal user-mode process.

### 4.1.6 process group & session

session: a collection of process groups attached to a single terminal

job: every process group in session is called job

## 4.2 Procedure of starting a process

Start-up routine (starting address of programs):

Before executing the main program, the start-up routine was called to retrieve command-line arguments and environmental variables.

## 4.3 Environment Variable & Command-line Arguments

Command-line arguments

Int argc: the number of arguments in command-line

Char \*argv[]: the address of arguments in command-line

***Note:***

In ISO C and POXIC protocol, argv[argc] is always a null pointer.

## 4.4 Termination Mechanism

---: \_exit,

---: \_Exit,

---: Exit

Exit: an termination method of process based on \_exit、 \_Exit, that encapsulated gc mechanism

## 4.5 Parent & Child Process

### 4.5.1 fork

Duplicate an child process according to an parent process. Sharing the same user\_id, group\_id, file descriptor and so on. However, the memory space is not sharable.

Usually, use fork to create child process, then use child process to exec corresponding programs.

#### 4.5.1.1 vfork

feature: the parent and child process use the parent memory space.

### 4.5.2 Spawn

the combination of two operation: fork && exec

### 4.5.3 Init

init process’ pid = 1;

When the parent process terminates, one child process will inherit from init and become the new parent process.

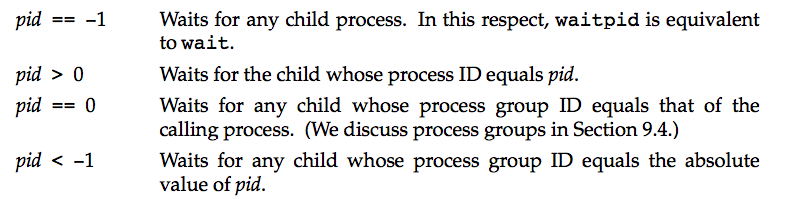
### 4.5.4 Wait & Waitpid & waiteid & wait3 & wait4

wait: caller function will block when a child process terminates

Waitpid: caller function could be configured not to block when a child process terminates and this function also provides an option to choose which process it waits for.

Waiteid: wait different type of child process.

Wait3 & wait4: wait one specific process with precise information about resources usage.



## 4.6 Behavior of process

If child processes terminate normally, parent processes could know the reason.

### 4.6.1 Abort

### 4.6.2 cancellation

### 4.6.3 execl

## 4.7 Type of process

### 4.7.1 zombie process

processes that have terminated (exit is called) still exist in Process Table, and could be called by using wait function to read its status, after read invocation, the references of zombie processes will be removed

### 4.7.2 orphan process

process whose parent process has died is still working.

## 4.8 Phenomenon

### 4.8.1 race condition(竞争条件)

## 4.9 Process vs thread

### 4.9.1 thread

#### 4.9.1.1 share among threads

text、data、heap、os resources are shared.

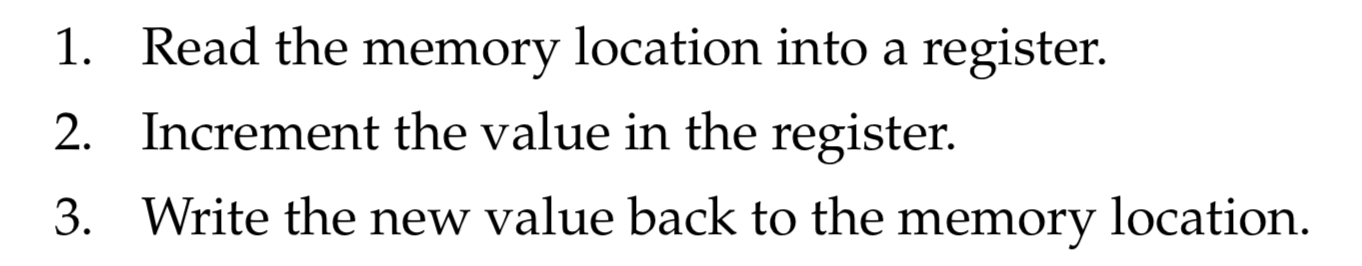
Stacks are owned by each thread.

#### 4.9.1.2 read & write operation

read operation: memory circle

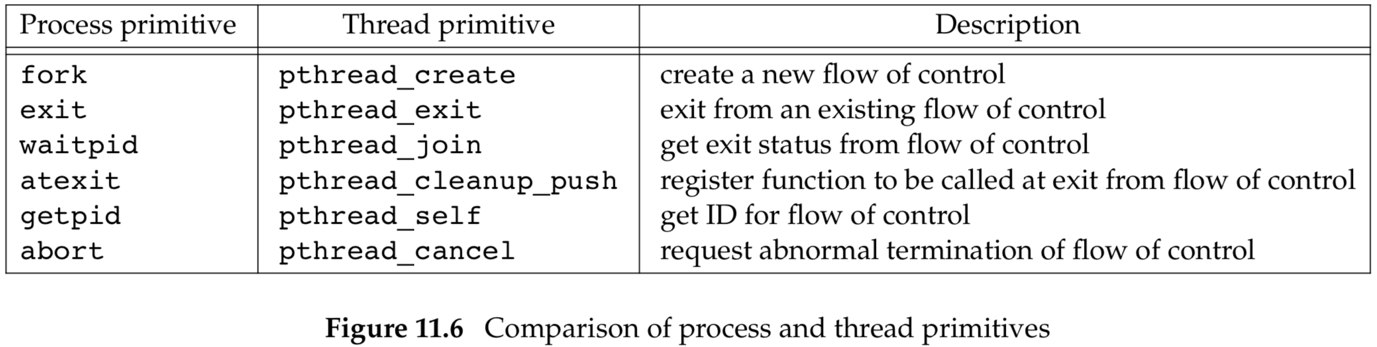
write operation: two memory circle.

For example:



### 4.9.x

#### 4.9.x.1



## 4.10 lock

### 4.10.1 situation

### 4.10.2 solutions

#### 4.10.2.1 mutex-信号量

mutex—short for mutual exclusion. 互斥锁。

Critical section: 临界区域， which means a block of code accessing the public resources.

##### 4.10.2.1.1 ordering of locking of resources

##### 4.10.2.1.2 timed lock of resources

could induce deaklock and can’t solve this problems intrinsically.

#### 4.10.2.2 reader-writer

type: read-lock、write-lock、no-lock

read-lock: shared lock

write-lock: exclusive lock

This mechanism is opt to be used in the situation in which resources tend to be read more then written.

---write operation: will block all read lock and write lock

---read operation: will block all write lock until all read locks are released.

#### 4.10.2.3 condition variables

#### 4.10.2.4 barriers

#### 4.10.2.5 spin lock

applicable in situation where locks were held for a short period of time.

Advantages: to reduce cost of being descheduled.

在检查并置位机制中，一个核心在对旗标执行读写的过程当中不会释放占用的访问总线。

## 4.11 daemon

nfsd, nfsiod, lockd, rpciod, rpc.idmapd, rpc.statd, and rpc.mountd daemons.

First four are kernel daemons, the last three are user-level daemons.

Nfsd: network file system daemon

Rpcbind: mapping of remote call procedure numbers to the network port numbers.

Inetd: listening on the network interfaces for the incoming requests

# 5. Signal

## 5.1 Signal list

### 5.1.1 terminal-generated signal

CTRL + C

### 5.1.2 Hardware exception

divide by 0、invalid memory reference

***SIGSEGV*** is generated for a process that executes an invalid memory reference. (端错误或者内存引用错误—segmentation fault)

### 5.1.3 software exception

SIGURG (urgent) : out-of-band data arrives over a network connection

SIGPIPE： write into pipe, however no readers

SIGALRM ：alarm notification

### 5.1.4 Signal

SIG\_ERR: signal\_error

SIG\_DFL: signal\_default

SIG\_IGN: signal\_

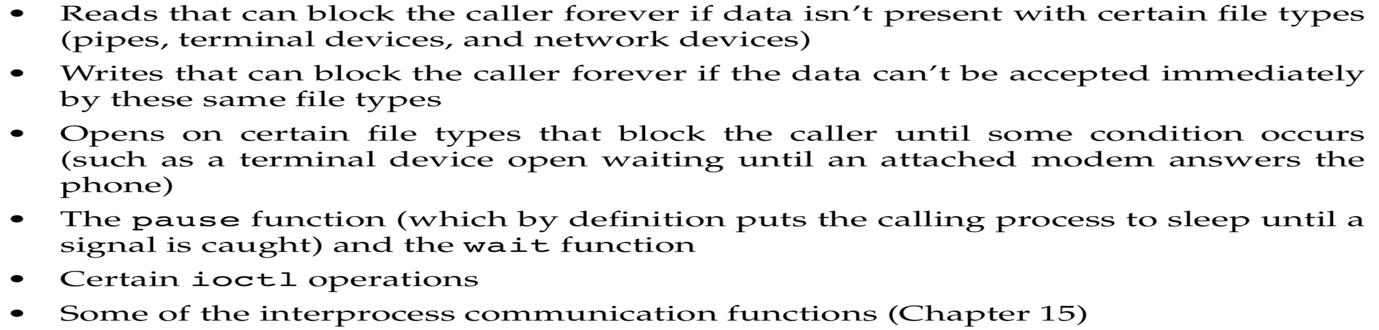
## 5.2 Signal Process

### 5.2.1 Running in Background (&)

Interrupt signal and quit signal will be ignored for the background process.

### 5.2.2 Process Block

5.2.2.1 Slow system call



# 问题

segmentation fault: user code that reaches illegal memory areas will result in segmentation fault.

Free the freed space will cause problem.

Free the pointer that are not generated by malloc family function(calloc、realloc)