注意：黄色区域为考试考到的题目

1、（P54）

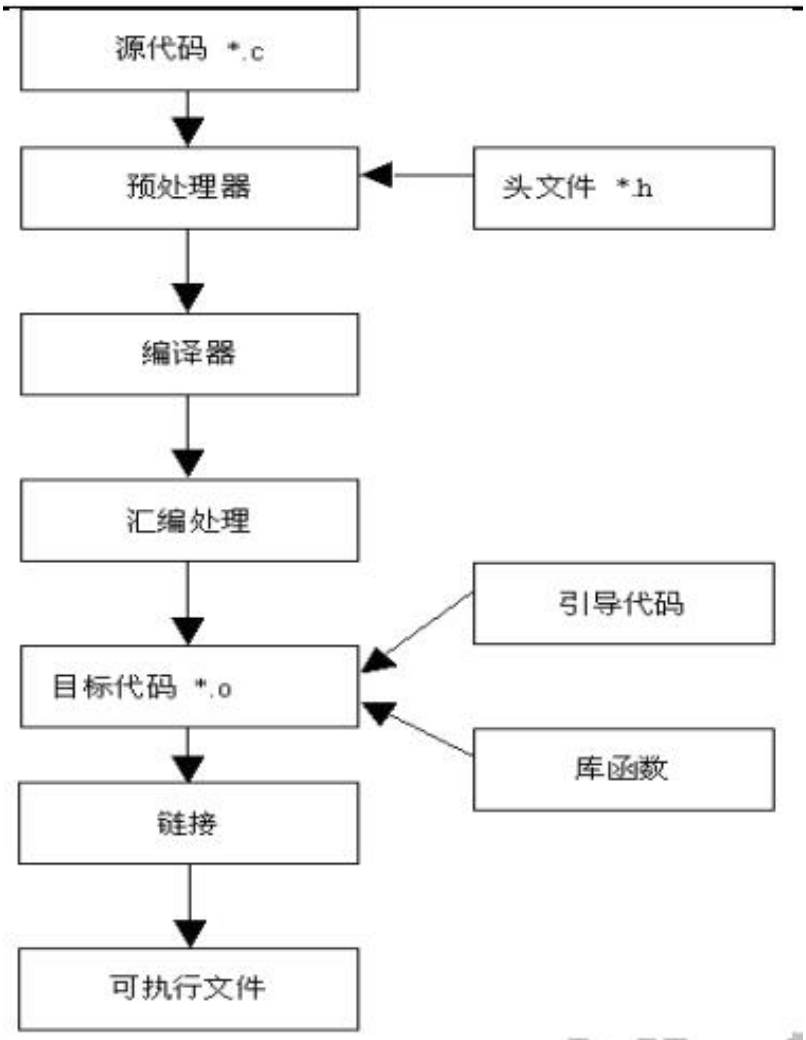


图 3.1 编译过程

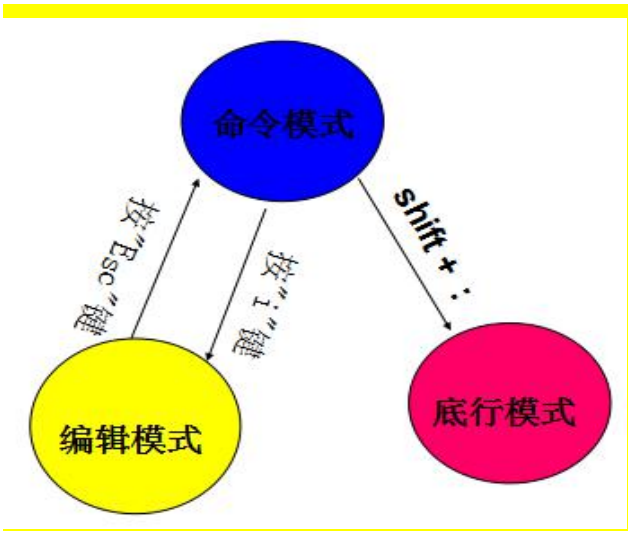
2、（P55）

vi 编译器，三个模式

命令模式：编辑字符，光标的移动

插入模式：输入字符

底行模式：模式匹配，查找指定字符串，保存文件



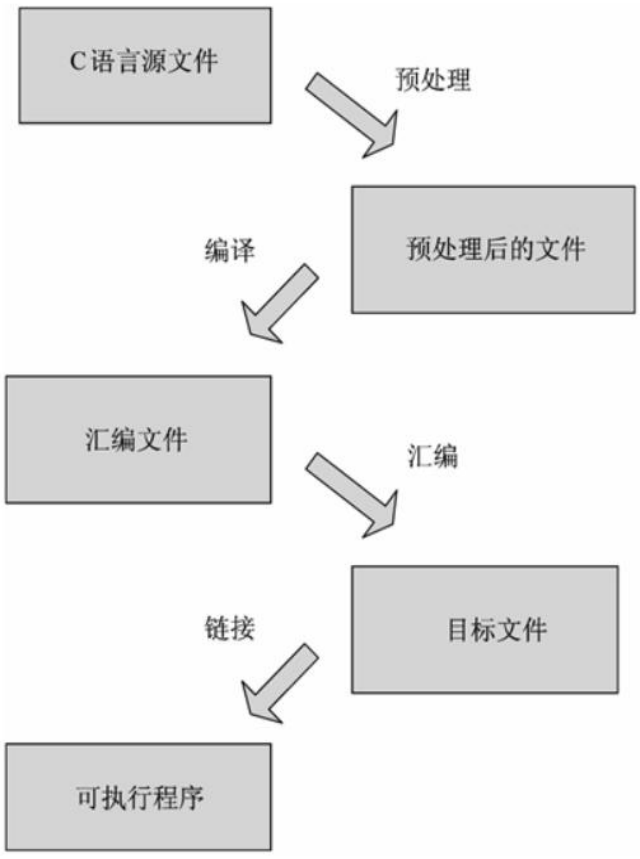
3、（P60）

编译器 GCC 的编译流程分为了 4 个步骤：预处理、编译、汇编、链接。

GCC 编译器在编译一个C语言程序时需要经过以下 4 步:

1. 将C语言源程序预处理，生成 .i 文件。
2. 预处理后的.i文件编译成为汇编语言，生成 .s 文件。
3. 将汇编语言文件经过汇编，生成目标文件 .o 文件。
4. 将各个模块的 .o 文件链接起来生成一个可执行程序文件。

GCC 编译流程如下图所示:



4、（P64）

函数库，静态库，动态库（概念）

静态库：静态链接库，程序编译时复制到目标程序中，编译完成后即无用，目标程序没有外部依赖，可直接运行

动态库：动态链接库，程序编译时不链接到目标代码，运行时才会被载入

5、（P67）【重点掌握】

gdb 调试，

考了大题，读代码，解释每一步做了啥，主要是 p 和 s

在保存退出后首先使用 gcc 对 test.c 进行编译，注意一定要加上选项“-g”，这样编译出的可执行代码中才包含调试信息，否则之后 gdb 无法载入该可执行文件。

```
[root@localhost gdb]# gcc -g test.c -o test
```

```
[root@localhost gdb]# gdb test
```

（1）查看文件。

在 gdb 中键入“l”（list）就可以查看所载入的文件，如下所示。

```
(gdb) l
```

（2）设置断点。

在 gdb 中设置断点非常简单，只需在“b”后加入对应的行号即可（这是最常用的方式，另外还有其他方式设置断点），如下所示：

```
(gdb) b 6
Breakpoint 1 at 0x804846d: file test.c, line 6.
```

（3）查看断点情况。

在设置完断点之后，用户可以键入“info b”来查看设置断点情况，在 gdb 中可以设置多个断点。

```
(gdb) info b
Num Type          Disp Enb Address    What
1   breakpoint     keep y   0x0804846d in main at test.c:6
```

（4）运行代码。

接下来就可运行代码了，gdb 默认从首行开始运行代码，键入“r”（run）即可（若想从程序中指定行开始运行，可在 r 后面加上行号）。

```
(gdb) r
Starting program: /root/workplace/gdb/test
Reading symbols from shared object read from target memory...done.
Loaded system supplied DSO at 0x5fb000

Breakpoint 1, main () at test.c:6
6               sum(50);
```

（5）查看变量值。

在程序停止运行之后，程序员所要做的工作是查看断点处的相关变量值。在 gdb 中键入“p”＋变量值即可，如下所示：

```
(gdb) p n
$1 = 0
(gdb) p i
$2 = 134518440
```

（6）单步运行。

单步运行可以使用命令“n”（next）或“s”（step），它们之间的区别在于：若有函数调用的时候，“s”会进入该函数而“n”不会进入该函数。因此，“s”就类似于 Uisual 等工具中的“step in”，“n”类似与 Uisual 等工具中的“step over”。它们的使用如下所示：

```
(gdb) n
The sum of 1-m is 1275
7           for (i = 1; i <= 50; i++)
(gdb) s
sum (m=50) at test.c:16
16              int i, n = 0;
```

可见，使用“n”后，程序显示函数 sum()的运行结果并向下执行，而使用“s”后则进入 sum()函数之中单步运行。

（7）恢复程序运行

在查看完所需变量及堆栈情况后，就可以使用命令“c”（continue）恢复程序的正常运行了。这时，它会把剩余还未执行的程序执行完，并显示剩余程序中的执行结果。以下是之前使用“n”命令恢复后的执行结果：

```
(gdb) c
Continuing.
The sum of 1-50 is :1275
Program exited with code 031.
```

可以看出，程序在运行完后退出，之后程序处于“停止状态”。

6、（P75）

考了 makefile 组成，作用

make 工程管理器，用来管理较多的文件

make 工程管理器就是个自动编译管理器，能够根据文件时间戳自动发现更新过的文件而减少编译的工作量，同时它通过读入 Makefile 文件的内容来执行大量的编译工作。

用户只需一次编写简单的编译语句即可。它大大提高了实际项目的工作效率。

---

makefile 是 make 读入的惟一配置文件，因此本节的内容实际就是讲述 makefile 的编写规则。在一个 makefile 中通常包含如下内容：

- 需要由 make 工具创建的目标体（target），通常是目标文件或可执行文件；
- 要创建的目标体所依赖的文件（dependency_file）；
- 创建每个目标体时需要运行的命令（command），这一行必须以制表符（tab 键）开头。

它的格式为：

```
target: dependency_files
    command /* 该行必须以 tab 键开头*/
```

例如，有两个文件分别为 hello.c 和 hello.h，创建的目标体为 hello.o，执行的命令为 gcc 编译指令：gcc –c hello.c，那么，对应的 makefile 就可以写为：

```
#The simplest example
hello.o: hello.c hello.h
      gcc -c hello.c -o hello.o
```

接着就可以使用 make 了。使用 make 的格式为：make target，这样 make 就会自动读入 makefile（也可以是首字母大写的 Makefile）并执行对应 target 的 command 语句，并会找到相应的依赖文件。如下所示：

```
[root@localhost makefile]# make hello.o
gcc -c hello.c -o hello.o
[root@localhost makefile]# ls
hello.c  hello.h  hello.o  makefile
```

可以看到，makefile 执行了“hello.o”对应的命令语句，并生成了“hello.o”目标体。

---

makefile 变量，自定义变量，预定义变量，自动变量，环境变量

makefile 规则，隐式规则，模式规则

autotools 功能：用来自动制作 makefile

7、（P153）

名词解释

文件 I/O 编程

系统调用：操作系统提供给用户程序的一组接口，用户程序可以获得系统内核提供的服务

用户编程接口（API）

系统命令：是一个可执行程序，内部引用用户编程接口（API）来实现相应的功能

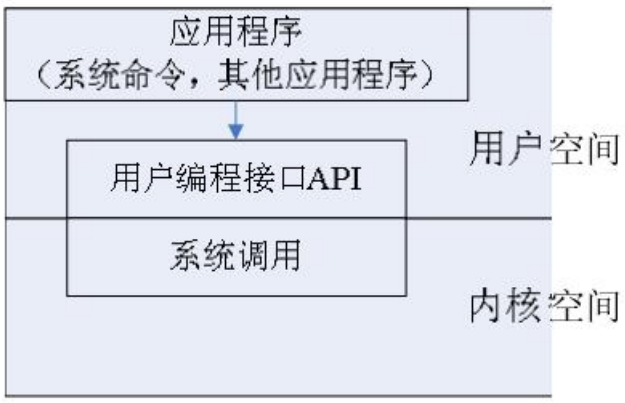


图 6.1 系统调用、AI
系统命令之间的关

文件 I/O 操作（P157）copy_file.c 不直接考，结合考

open() 打开创建文件
close() 关闭文件
read() 从指定文件描述符读数据放到缓存区，返回实际读入的字节数
write() 向打开的文件写数据
lseek() 在指定文件描述符中将文件指针定位到相应的位置

文件锁：多个进程同时访问同一个文件，实现并发互斥访问[lockf() / fcntl()]

读锁，写锁
同时上读锁可以

当前锁状态	读锁请求	写锁请求
无锁	可以	可以
读锁	可以	阻塞
写锁	阻塞	阻塞

I/O 多路复用（P163、166）【重点掌握】
multiplex_select，抽一段代码，解释参数

考了代码执行过程和改进措施

文件读写方式实现生产者和消费者（P192）【要考，节选代码，说明含义，参数意思】

8、（P204）

进程的定义：一个程序执行一次的过程，同时也是资源分配的最小单位（基本调度和管理资源的单位），是动态的，而程序是静态的，执行一个程序的时候它将启动一个进程

进程控制块：包含进程的描述信息、控制信息及资源信息，是进程的一个静态描述

进程标识：进程号（PID），父进程号（PPID），PID 唯一地标识一个进程，PID 与 PPID 都是非零的正整数

进程运行的三个状态：

执行态：正在运行，即正在占用 CPU

就绪态：具备执行的一切条件，等待分配 CPU 地处理时间片

等待态：不能使用 CPU，若等待事件发生（等待的资源分配到）则可将其唤醒

图 7.1 进程 3 种状态的转化关系

fork()函数（P206）

用于从已经存在的进程中创建一个新进程，几乎是父进程地复制品，独有的只有进程号、资源使用、计时器等，父进程返回的是子进程的进程号，子进程返回的是 0

函数返回值	0：子进程
	子进程 ID（大于 0 的整数）：父进程
	-1：出错

exec 函数族（P208）

提供了一个在进程中启动另一个程序执行的方法

wait()，waitpid() （P216）

核心功能：阻塞父进程，等待子进程的退出，子进程不退出，父进程不能退出

第三个参数：要么是 0，要么是 WNOHANG。0 代表阻塞父进程期间，父进程不能做任

何事；若是 WNOHANG，父进程不挂起，等待期间还可以做一些事情

如果使用了 WNOHANG(wait no hung)参数调用 waitpid，即使没有子进程退出，它也会立即返回，不会像 wait 那样永远等下去。

1. 当正常返回的时候，waitpid 返回收集到的子进程的进程 ID；
2. 如果设置了选项 WNOHANG，而调用中 waitpid 发现没有已退出的子进程可收集，则返回 0；
3. 如果调用中出错，则返回-1，这时 errno 会被设置成相应的值以指示错误所在；

考了作用和步骤

守护进程（P217）

含义：系统后台运行的进程，他是脱离控制终端的（不能用 gdb 调试，从日志中看）

步骤：

1. 创建子进程，父进程退出
2. 在子进程中创建新会话
3. 改变当前目录为根目录
4. 重设文件权限掩码
5. 关闭文件描述符

9、（）

进程间通信：管道、信号、消息队列、共享内存、信号量、套接字（Socket）

三种管道（特征/区别）：

无名管道，在有亲缘关系的进程间通讯，只存在于内核空间，无实体文件

标准流管道，固定读写端，写端固定于键盘，读端固定于显示器，创建管道的一系列动作标准化到 popen()函数中，大大减少代码的编写量

[第二个参数，“r”读，”w”写]

有名管道，创建了一个管道文件，任意两个进程都可以通讯

信号（P243）

是进程通信的一种最古老的方式，是在软件层面对中断机制的一种模拟，可以直接进行用户空间进程和内核进程之间交互

SIGINT，用户键入 INTR 字符（ Ctrl + C ）时发出，终端驱动发出此信号并送到前台进程中的每一个进程，默认终止

SIGQUIT，和 SIGINT 类似，由 QUIT 字符（ Ctrl + \ ）来控制，默认终止

信号处理函数，signal()和 sigaction()

信号集函数组：创建信号集合、注册信号处理函数、检测信号

创建信号集合：

sigemptyset()	将信号集合初始化为空
sigfillset()	将信号集合初始化为包含所有已定义的信号的集合
sigaddset()	将指定信号加入到信号集合中
sigdelset()	将指定信号从信号集合中删除
sigismember()	查询指定信号是否在信号集合中

注册信号处理函数：

sigprocmask()	检测并更改信号屏蔽字
sigaction()	定义进程接收到特定信号之后的行为

检测信号：

sigpending()	允许进程检测“未处理”信号，并决定如何处理

（P252）代码

（P254）

信号量：两个原子操作

P 操作：如果有可用的资源（信号量值>0），则占用一个资源（给信号量值减去一，进入临界区代码）;如果没有可用的资源（信号量值等于 0），则被阻塞到，直到系统将资源分配给该进程（进入等待队列，一直等到资源轮到该进程）。

V 操作：如果在该信号量的等待队列中有进程在等待资源，则唤醒一个阻塞进程。如果没有进程等待它，则释放一个资源（给信号量值加一）。

p 操作和 v 操作是不可中断的程序段，称为原语。P,V 原语中 P 是荷兰语的 Passeren，相当于英文的 pass,V 是荷兰语的 Verhoog,相当于英文中的 incremnet。

P 原语操作的动作是：

（1） sem 减 1；

（2） 若 sem 减 1 后仍大于或等于零，则进程继续执行；

（3）若 sem 减 1 后小于零，则该进程被阻塞后进入与该信号相对应的队列中，然后转进程调度。

V 原语操作的动作是：

（1） sem 加 1；

（2） 若相加结果大于零，则进程继续执行；

（3）若相加结果小于或等于零，则从该信号的等待队列中唤醒一等待进程，然后再返回原进程继续执行或转进程调度。

需要提醒大家一点就是 P,V 操作对于每一个进程来说，都只能进行一次。而且必须成对使用。且在 P,V 愿语执行期间不允许有中断的发生。

（P258）【要考】

sem_com.c/fork.c，不是二维信号量，信号量为 0 时不执行 P 操作，对任何一个都可以做 V 操作（语句次序颠倒一下，按要求实现要求，调整为正确顺序）

（P260）

共享内存，是一种最为高效的进程间通信方式，进程可以直接读写内存，不需要任何数据的复制

本质原理：内核中留出一块内存区，映射到两个进程的存储空间，使得两个进程读写自己的存储空间就是读写同一块共享内存

（P262）没用信号量，用密钥字符串做的，父进程必须通过密钥才能读取，否则不能读取

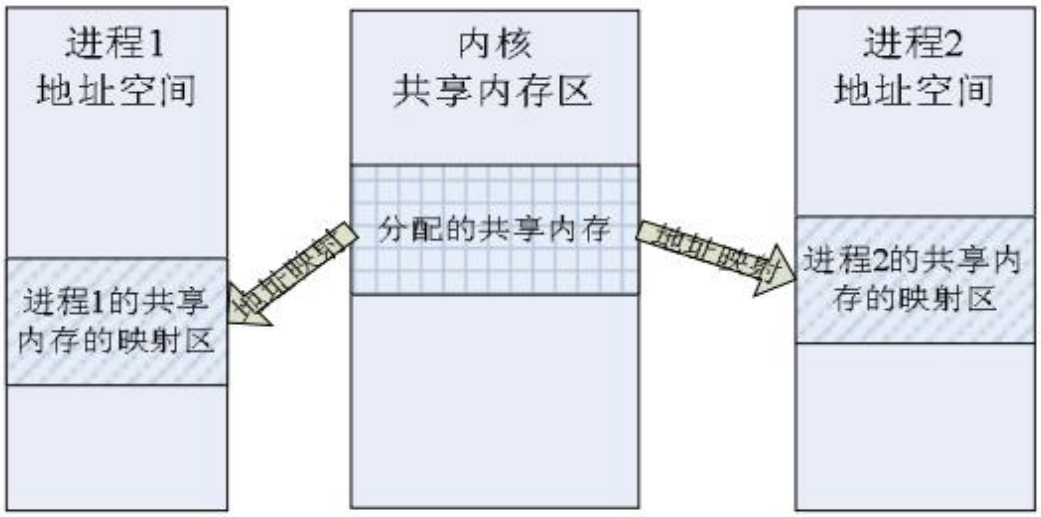


图 8.8　共享内存原理示意图

（P266）消息队列

一个进程进程给消息队列挂消息，一个进程从消息队列取消息，FIFO（first in first out），操作永远针对消息队列末尾

10、

编译带有多线程的程序必须带有 -pthread

（eg：g++ -pthread -o test test.o init.o ethernet.o）

（P292）【要考，两种考法】

把 201 线程执行次序对应的代码应该做什么修正，或者自己写的代码说一下按什么顺序执行

（P298）【要考，意义价值功能】

生产者消费者，多线程和信号量综合解决方案

三个进程，producer，customer，存放资源的区域

读代码，mutex 放在第一行，初值为 1，消费者先抢到 cpu，会出现什么情况

使用三个信号量，

avail，有界缓冲区中的空单元数，初值为 3，约束消费者，意思是三个单元中资源全为空

full，非空单元数，初始值为 0，限制生产者，意思三个单元满了

mutex，互斥信号量，初始值为 1，

如果 mutex 初值为 0，想实现先生产后消费，应该把 mutex 信号量放在消费者线程

第一行，用 P 操作（-1，但是初值为 0，所以被阻塞，生产者先执行，然后信号量执行执行 V 操作后才可执行）

11、

给下面那张图，写出工作流程

网络编程（基于 TCP 协议，不考代码，考工作流程，P313）

socket 基础编程

客户端主动向服务端发起 socket 连接，连接成功后，客户端发送消息，服务端接收消息，一次通信完成

局限：服务端只能接受一个客户端的一次消息传递，接收完后就结束

高级编程

服务端可以同时接收多个客户端的多个消息发送，select()函数

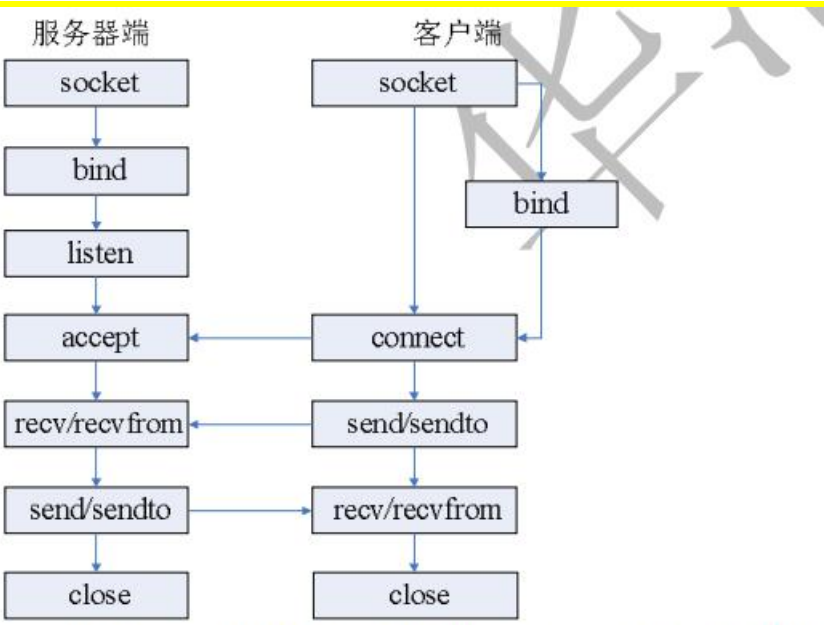


图 10.6　使用 TCP 协议 socket 编程流程图

程图