1. 课外练习（不需要提交）

1）阅读程序，并在Linux下运行，确保完全读懂。

#include <stdio.h>

#include <unistd.h>

int main()

{

int p1, p2;

while ((p1 = fork()) == -1)

;

if (p1 == 0)

putchar('b');

else

{

while ((p2 = fork()) == -1)

;

if (p2 == 0)

putchar('c');

else

putchar('a');

}

return 0;

}

2）对上述程序略加修改，如下，分析其运行结果，并在Linux下运行验证，确保完全弄懂。

#include <stdio.h>

#include <unistd.h>

int main()

{

int p1, p2;

while ((p1 = fork()) == -1)

;

if (p1 == 0)

{

putchar('b');

fflush(stdout);

sleep(2);

putchar('b');

}

else

{

while ((p2 = fork()) == -1)

;

if (p2 == 0)

{

putchar('c');

fflush(stdout);

sleep(1);

putchar('c');

}

else

{

putchar('a');

fflush(stdout);

sleep(3);

putchar('a');

}

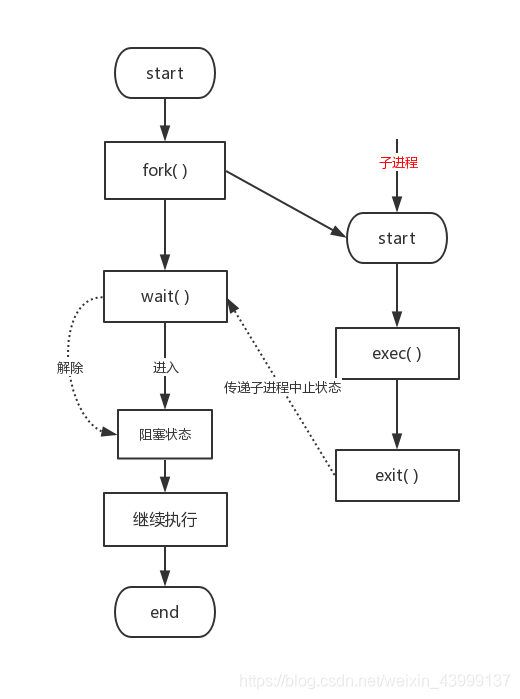
}

return 0;

}

3）上述程序中，三个分支都调用了fflush()函数，把这三处调用都注释掉，分析运行结果，并在Linux下运行验证，确保完全弄懂。

4）对照流程图，阅读程序，分析运行结果，并在Linux下运行验证，确保完全读懂。



#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

int main()

{

int pid;

pid = fork();

switch (pid)

{

case -1:

printf("fork fail!\n");

exit(1);

case 0:

sleep(3);

execl("/bin/ls", "ls", "-1", "-color", NULL);

sleep(30);

printf("exec fail!\n");

exit(1);

default:

wait(NULL);

printf("ls completed !\n");

exit(0);

}

return 0;

}

2. 程序分析

1）分析以下程序，LINE A这一行的输出是什么？

#include <sys/types.h>

#include <sys/wait.h>

#include <stdio.h>

#include <unistd.h>

int value = 5;

int main()

{

pid\_t pid;

pid = fork();

if (pid == 0)

{ /\* child process \*/

value += 15;

return 0;

}

else if (pid > 0)

{ /\* parent process \*/

wait(NULL);

printf("PARENT: value = %d", value); /\* LINE A \*/

return 0;

}

}

2）分析以下程序总共产生了多少个进程。

#include <stdio.h>

#include <unistd.h>

int main()

{

/\* fork a child process \*/

fork();

/\* fork another child process \*/

fork();

/\* and fork another \*/

fork();

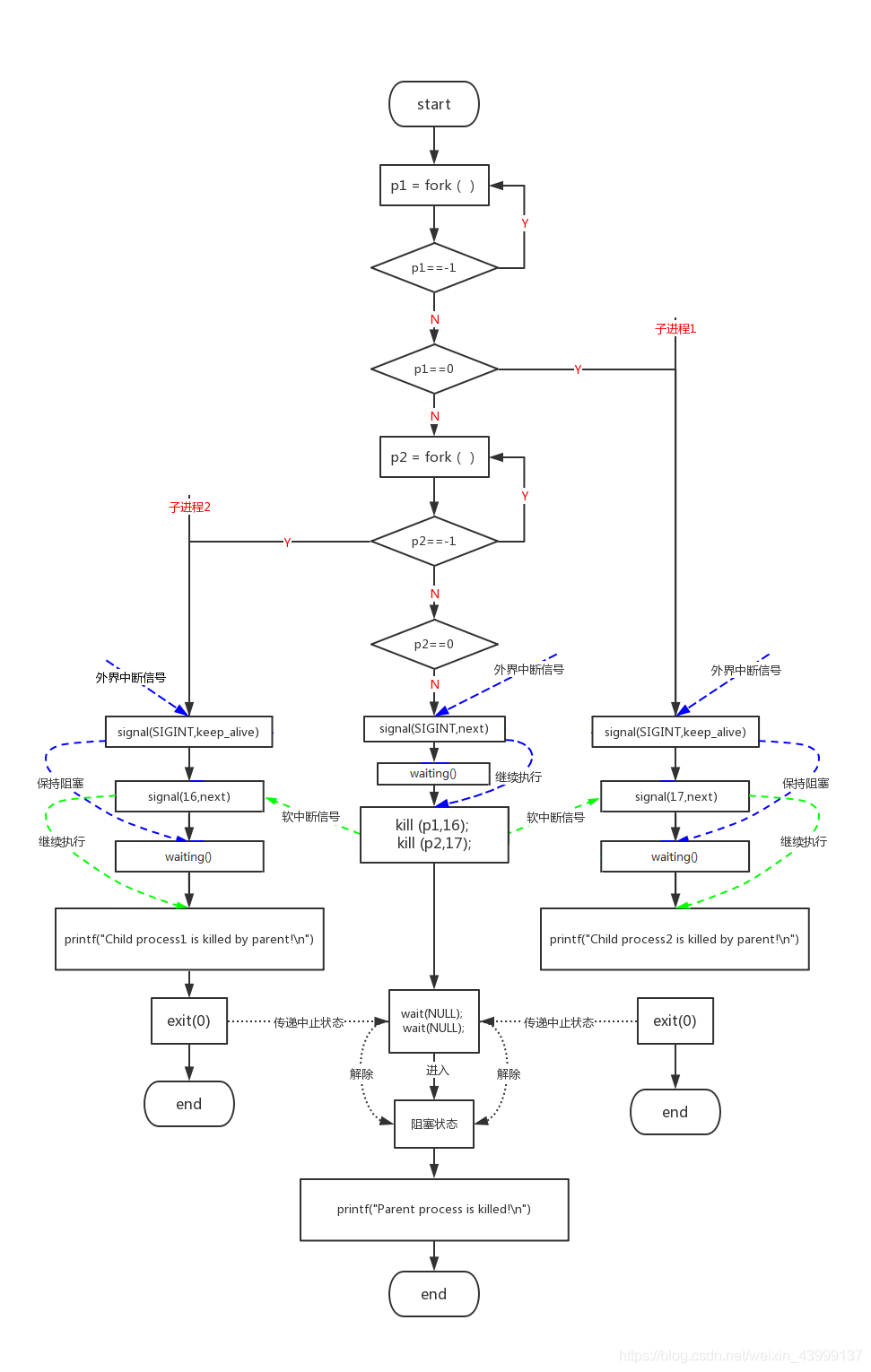
return 0;

}

3. Linux信号机制课外练习（不需要提交）

自行搜索和学习：信号机制，SIGINT和进程组，signal()和kill()函数。

注意：信号机制（signal）与后面要讲的信号量（semaphore）完全无关。



进一步的学习建议：进程组、会话、控制终端之间的关系，前台进程、后台进程和守护进程之间的差异，signal()和sigaction()之间的差异。

以下程序使用fork()创建2个子进程，然后使用signal()为父进程捕获键盘（Ctrl+C）触发的信号。捕获信号后，使用kill()作为父进程向2个子进程发送信号。2个子进程在捕获信号后输出一些提示消息并退出，父进程在两个子进程退出后输出一些提示消息。如图所示。

注意图中提到“外界中断信号”，是指SIGINT信号，与中断机制无关。图中还提到“保持阻塞”，阅读程序会发现，并没有阻塞，只是保持原地循环等待。

阅读程序，并在Linux下运行，确保完全读懂。注意：此程序不适合以调试方式运行，因为其运行要求用户输入Ctrl+C，而在调试方式下，输入Ctrl+C不会被我们的程序收到，却会把GDB调试器干掉（你也可以尝试一下，然后在终端输入top命令，看看出现什么情况）。正确做法是直接编译生成（VSCode菜单：终端->运行生成任务）可执行文件，然后在终端中敲命令运行。

#include <stdio.h>

#include <signal.h>

#include <unistd.h>

#include <sys/wait.h>

#include <stdlib.h>

void waiting(), next(int), keep\_alive(int);

int wait\_mark;

int main()

{

int p1, p2, stdout;

while ((p1 = fork()) == -1)

;

if (p1 > 0)

{

while ((p2 = fork()) == -1)

;

if (p2 > 0)

{

wait\_mark = 1;

signal(SIGINT, next);

waiting();

kill(p1, 16);

kill(p2, 17);

wait(NULL);

wait(NULL);

printf("Parent process is killed!\n");

exit(0);

}

else

{

wait\_mark = 1;

signal(SIGINT, keep\_alive);

signal(17, next);

waiting();

printf("Child process2 is killed by parent!\n");

exit(0);

}

}

else

{

wait\_mark = 1;

signal(SIGINT, keep\_alive);

signal(16, next);

waiting();

printf("Child process1 is killed by parent!\n");

exit(0);

}

return 0;

}

void waiting()

{

while (wait\_mark != 0)

;

}

void next(int signum)

{

wait\_mark = 0;

}

void keep\_alive(int signum)

{

}

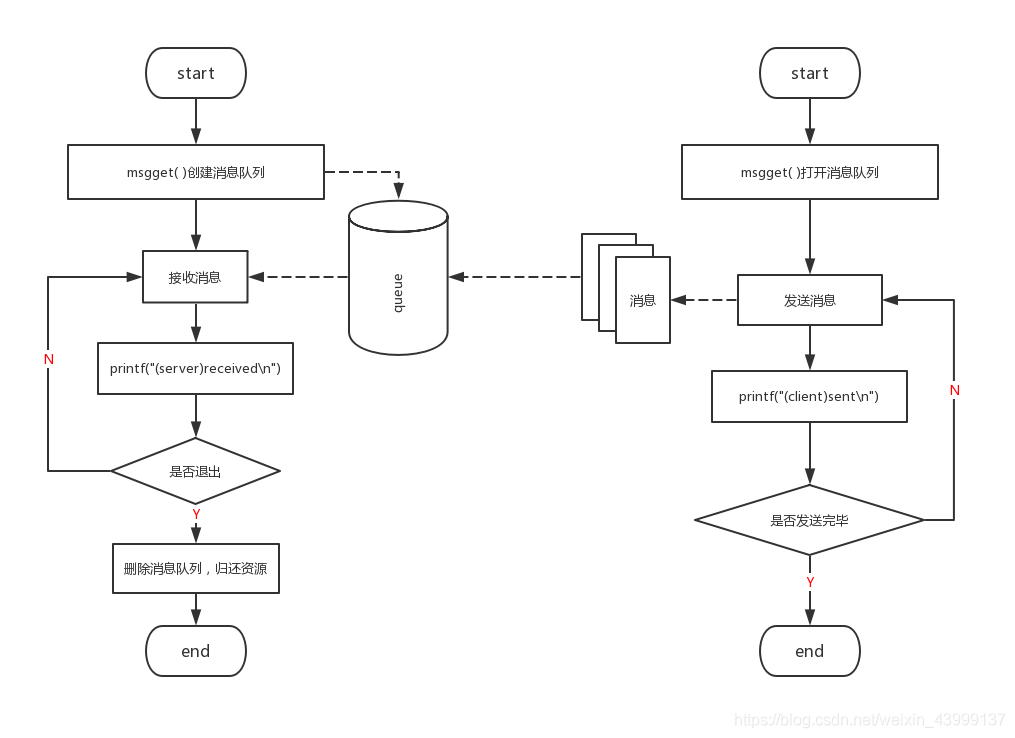
4. 消息机制程序填空

自行搜索和学习：消息队列机制，msgget()，msgsnd()，msgrev()和msgctl()，msgsnd()和msgrev()的msgflg参数与阻塞设定。

服务端一个进程，使用msgget()创建一个Key为75的消息队列，然后反复使用msgrev()从队列中获取消息，识别消息类型，并向屏幕输出一些东西，如果消息的类型值为1，表示QUIT消息，则程序退出消息循环并使用msgctl()删除消息队列，最后退出。

客户端另一个进程，它将在服务端开始运行后启动运行，使用msgget()获取Key为75（由服务端创建）的消息队列，然后构造10条消息类型从10到1的消息并使用msgsnd()逐个发送，之后程序退出。

如下图所示。



程序如下，填写其中空缺的部分，并提交关键代码和运行结果的截图（要求输出中增加学号姓名）。

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/msg.h>

#include <sys/ipc.h>

#include <sys/wait.h>

#define MSGKEY 75

struct msgform

{

long mtype;

char mtext[1000];

} msg;

int msgqid;

void run\_server()

{

// 请填写

}

void run\_client()

{

// 请填写

for (…)

{

// 请填写

sleep(1);

}

}

int main()

{

int pid;

while ((pid = fork()) == -1)

;

if (pid == 0)

{

sleep(1);

run\_client();

exit(0);

}

else

{

run\_server();

}

wait(NULL);

return 0;

}