1. 共享内存（不需要提交）

自行搜索并学习Linux的共享内存机制：shmget()、shmat()、shmdt()、shmctl()，以及ipcs -m命令。

使用fork()创建2个子进程，二者分别担任服务端和客户端，并使用共享内存。然后对父进程调用system()以执行ipcs -m命令，并使用wait()等待两个子进程完成其工作。

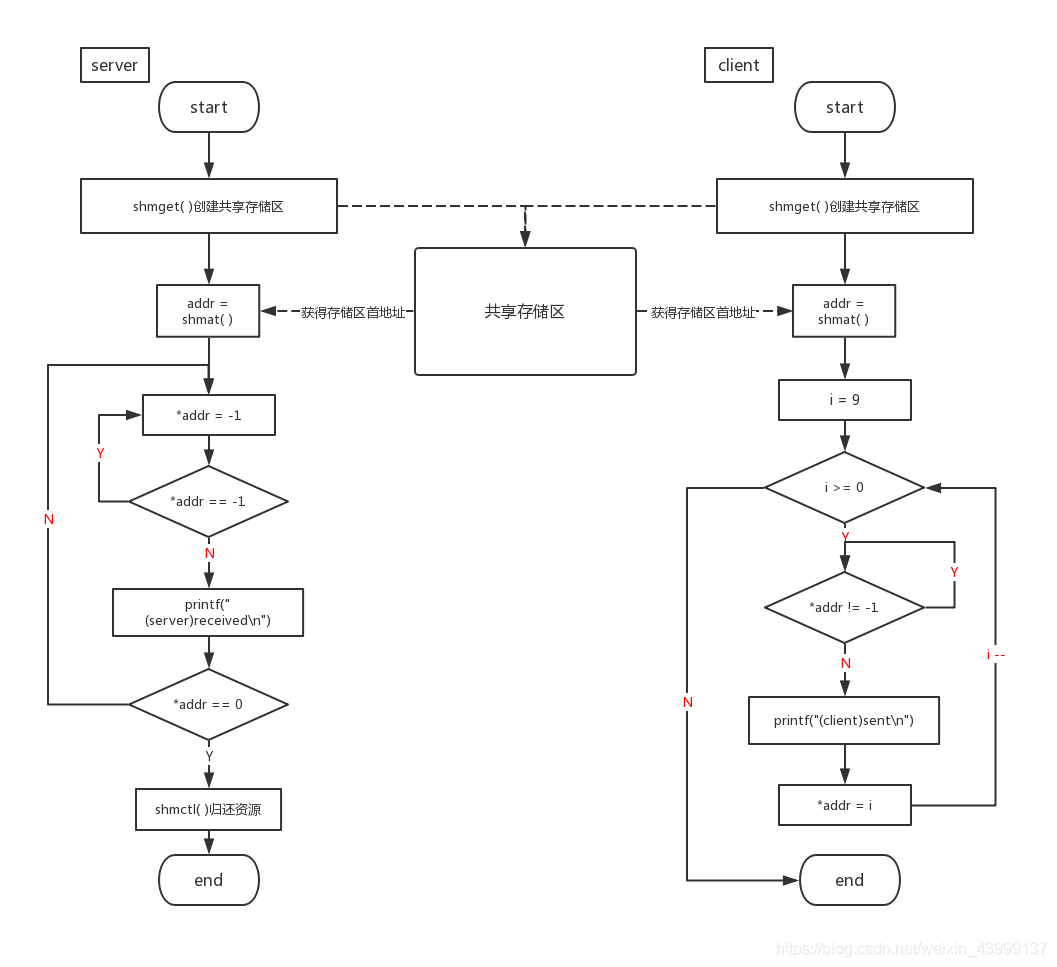
对于两个子进程，使用 shmget()创建或获取共享内存（事先约定的键值为75，长度为1024字节），并使用shmat()将共享内存映射到逻辑地址空间。

然后，两个子进程都重复检查共享内存起始位置的int变量。

服务器端进程从共享内存读取，一旦int变量不是-1，则表示客户端已经写入了某些内容，因此读取它并输出到屏幕上，然后将int变量设置回-1；如果int变量为0，则表示客户端请求退出，因此使用shmctl()删除共享内存并退出。

客户端进程写入共享内存，一旦int变量变为-1，则意味着该值已被服务器端获取，因此可以把下一个值写入int变量。这样，客户端进程将10个值（从9到0）逐个写入int变量，然后退出。

工作方式如下图所示。



#include <sys/shm.h>

#include <sys/ipc.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#define SHMKEY 75

int shmid, pid;

int \*addr;

void client()

{

int i;

shmid = shmget(SHMKEY, 1024, 0777 | IPC\_CREAT);

addr = (int \*)shmat(shmid, 0, 0);

for (i = 9; i >= 0; i--)

{

while (\*addr != -1)

;

printf("(client)sent: %d\n", i);

fflush(stdout);

\*addr = i;

usleep(10);

}

exit(0);

}

void server()

{

shmid = shmget(SHMKEY, 1024, 0777 | IPC\_CREAT);

addr = (int \*)shmat(shmid, 0, 0);

do

{

\*addr = -1;

while (\*addr == -1)

;

printf("(server)received: %d\n", \*addr);

fflush(stdout);

} while (\*addr);

shmctl(shmid, IPC\_RMID, 0);

exit(0);

}

int main()

{

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

;

if (!pid)

server();

system("ipcs -m");

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

;

if (!pid)

client();

wait(0);

wait(0);

return 0;

}

2 信号量

自行上网搜索并学习Linux的信号量机制：semget()、semop()、shmctl()。

对前述共享内存案例的2个子进程使用信号量，执行相同的工作，但避免重复检查int变量。

请注意，父进程应在创建子进程之前创建信号量并设置其初始值，调用system()以执行ipcs -s命令，并在子进程完成后删除信号量。创建信号量的键可以是6666、7777、8888等。

semop()使用的sembuf结构在sem.h中定义如下：

struct sembuf{

short sem\_num;

short sem\_op;

short sem\_flg;

};

semctl()使用的semun类型必须由我们自己定义如下：

union semun {

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

注意：

1）此题本质上是生产者消费者问题的简化版，由于只有一个生产者，一个消费者，所以不需要担心共享内存的互斥访问问题，不需要mutex信号量。

2）此题同时用到共享内存和信号量，不要把自己绕晕了。

3）此题的目的不是锻炼同步的逻辑思维能力，只是体验真实环境下的相关编程。程序的主体结构老师已经提供，只需要在空缺处填写即可。

4）提交填写的程序的截图即可，附上学号姓名。输出结果与前一题基本相同，不用截图。

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#define SHMKEY 75

#define SEMFULL 6666

#define SEMEMPTY 7777

/\* defined in sem.h

struct sembuf{

short sem\_num;

short sem\_op;

short sem\_flg;

};

\*/

union semun

{

int val;

struct semid\_ds \*buf;

unsigned short \*arry;

};

int shmid, pid, semid\_full, semid\_empty;

struct sembuf sem\_b;

int \*addr;

void client()

{

int i;

shmid = shmget(SHMKEY, 1024, 0777 | IPC\_CREAT);

addr = (int \*)shmat(shmid, 0, 0);

for (i = 9; i >= 0; i--)

{

// 请填写此处的程序

}

exit(0);

}

void server()

{

shmid = shmget(SHMKEY, 1024, 0777 | IPC\_CREAT);

addr = (int \*)shmat(shmid, 0, 0);

do

{

// 请填写此处的程序

} while (\*addr);

shmctl(shmid, IPC\_RMID, 0);

exit(0);

}

int main()

{

union semun sem\_union;

semid\_empty = semget((key\_t)SEMEMPTY, 1, 0777 | IPC\_CREAT);

sem\_union.val = 1;

semctl(semid\_empty, 0, SETVAL, sem\_union);

semid\_full = semget((key\_t)SEMFULL, 1, 0777 | IPC\_CREAT);

sem\_union.val = 0;

semctl(semid\_full, 0, SETVAL, sem\_union);

sem\_b.sem\_num = 0;

sem\_b.sem\_flg = SEM\_UNDO;

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

;

if (!pid)

server();

system("ipcs -s");

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

;

if (!pid)

client();

wait(0);

wait(0);

semctl(semid\_empty, 0, IPC\_RMID, sem\_union);

semctl(semid\_full, 0, IPC\_RMID, sem\_union);

return 0;

}