## 使用gdb调试：

gcc -g server.c -o server.o

gdb ./server.o

start

next 简写 n 单步执行

step s step into

list l 显示附近代码

list 6,21 显示行号范围内的代码

list file.c:函数名 指定函数显示

print expr p 打印expr值

display

quit q 退出gdb

run 提示是否需要重新运行程序，输入y重新开始运行程序

start 也可以重新运行程序

break file.c:行号 b file.c:18 如果是单文件则可以直接 b 行号

break file.c:行号 if expr 条件断点

info b 显示当前断点信息，尤其是第一个num 可以用来删除断点

delete breakpoints num 删除断点

continue c 继续执行到下一个断点

多进程调试

gdb chengxu.o

catch fork 让调试在fork()处停止，相当于在fork()处打断点

show follow-fork-mode 显示当前哪个进程被阻塞用于调试

show detach-on-fork on代表follow-fork-mode没有指定的进程不被调试器控制，自己运行，off代表两个进程都被调试器控制，进入阻塞等待切换调试

set detach-on-fork off 让调试器控制两个进程的调试

start

info inferiors 显示当前运行的多个进程信息

inferior num 切换到某进程调试

detach inferiors num 让某个进程脱离调试器自己运行

kill inferiors num 杀死进程num

linux gdb 调试多进程

fork.c

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

int add(int x,int y){

return x+y;

}

int test(){

int num=0;

pid\_t pid=fork();

if(pid==0){

while(1){

num=add(1,2);

printf("child pid=%d , num=%d,\n",getpid(),num);

sleep(1);

}

exit(0);

}

if(pid>0){

while(1){

num=add(10,20);

printf("pid=%d,num=%d,\n",getpid(),num);

sleep(1);

}

}

return 0;

}

int main(){

test();

return 0;

}

linux gdb调试多线程

编译gcc -g thread.c -lpthread(链接pthread线程库)

gdb调试：gdb ./a.out

在线程中两个调用add函数的位置分别设置断点

start启动程序，c继续执行程序直到断点处

info threads显示当前程序中的线程

c继续执行程序，到达第二个程序再执行info threads命令，发现当前程序发生了线程切换，由线程2变为线程1

thread 2手动切换线程到线程2，再执行info threads查看当前线程

#include<stdio.h>

#include<unistd.h>

#include<pthread.h>

int add(int x,int y){

return x+y;

}

void\* thread(void \* args){

int num=0;

while(1){

num+=add(1,2);

sleep(1);

}

}

int main(){

pthread\_t tid;

pthread\_create(tid,NULL,thread,NULL);

int num=0;

while(1){

num+=add(10,20);

sleep(1);

}

return 0;

}