

Support for remote debugging

Applies to:			No. L6-001
Platform	OS version	SDK version	Classification
LinPAC series	All version	LP-8x4x: V1.12 or later Others: All version	Linux Applications

Remote debugging is the process of debugging a program running on a different system (called target) from a different system (called host as local PC).

A debugger can pause your program and you can watch the values of the variables that you have defined. To perform remote debugging, the following utilities are needed:

- gdbserver – Run this on your target system- LinPAC controller for example.
- gdb – Execute this on your host system to connect to your target system-local PC for example.

Model Name	GDB Command
PXA270 Series: LinCon/LP-51xx/8x3x/8x4x	arm-linux-gdb.exe
AM335x Series: LP-52xx/8x2x/9x2x	arm-linux-gnueabi-gdb.exe
X86 Series: LP-8x8x/8x8x-Atom/LX-8000/9000	gdb.exe

This tutorial gives you easy-to-follow instructions works on two different interface.

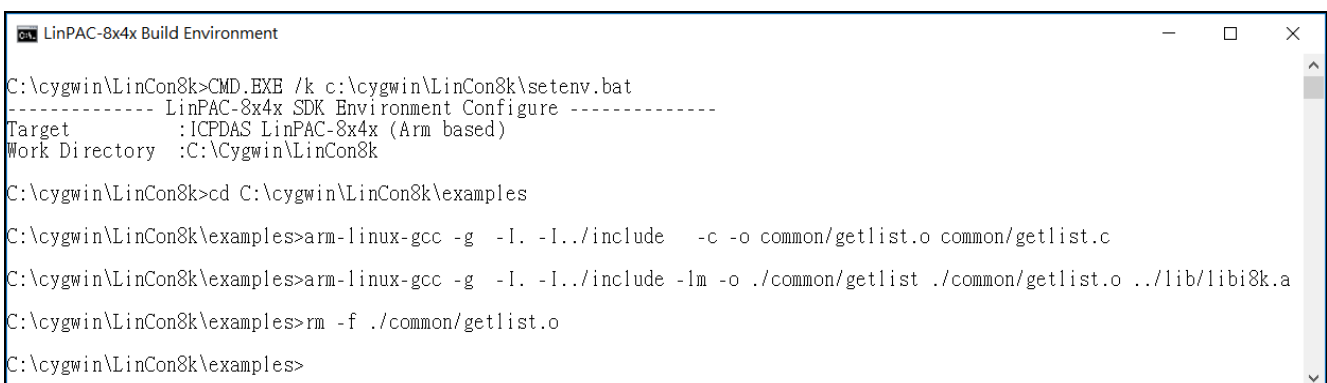
Section 1: Working with the Command line interface

The following example is based on an LP-8x4x controller.

The LP-8x4x SDK is based on the Cygwin interface and provides a Linux-like environment for Microsoft Windows systems, which can be executed within the LinPAC-8000 Build Environment in order to connect to the target system.

Step 1: Compile the `getlist.c` file to `getlist.exe`

As illustrated in the Fig. 1, the option **-g** must be added to the command line instruction in order to compile a `.c` file to an executable.



```

C:\cygwin\LinCon8k>CMD.EXE /k c:\cygwin\LinCon8k\setenv.bat
----- LinPAC-8x4x SDK Environment Configure -----
Target      : ICPDAS LinPAC-8x4x (Arm based)
Work Directory : C:\Cygwin\LinCon8k

C:\cygwin\LinCon8k>cd C:\cygwin\LinCon8k\examples
C:\cygwin\LinCon8k\examples>arm-linux-gcc -g -I. -I../include -c -o common/getlist.o common/getlist.c
C:\cygwin\LinCon8k\examples>arm-linux-gcc -g -I. -I../include -lm -o ./common/getlist ./common/getlist.o ../lib/libi8k.a
C:\cygwin\LinCon8k\examples>rm -f ./common/getlist.o
C:\cygwin\LinCon8k\examples>
  
```

Fig. 1

Step 2: Upload the “getlist.exe” file to the LP-8x4x controller

In your FTP program, FileZilla is used in this example, locate the getlist.exe file in the LP-8x4x and then right click the file icon to open the context menu.

Select the Permissions option. In the Permissions dialog box, type “777” into the Numeric value textbox, and then click the OK button, as illustrated in the Fig. 2.

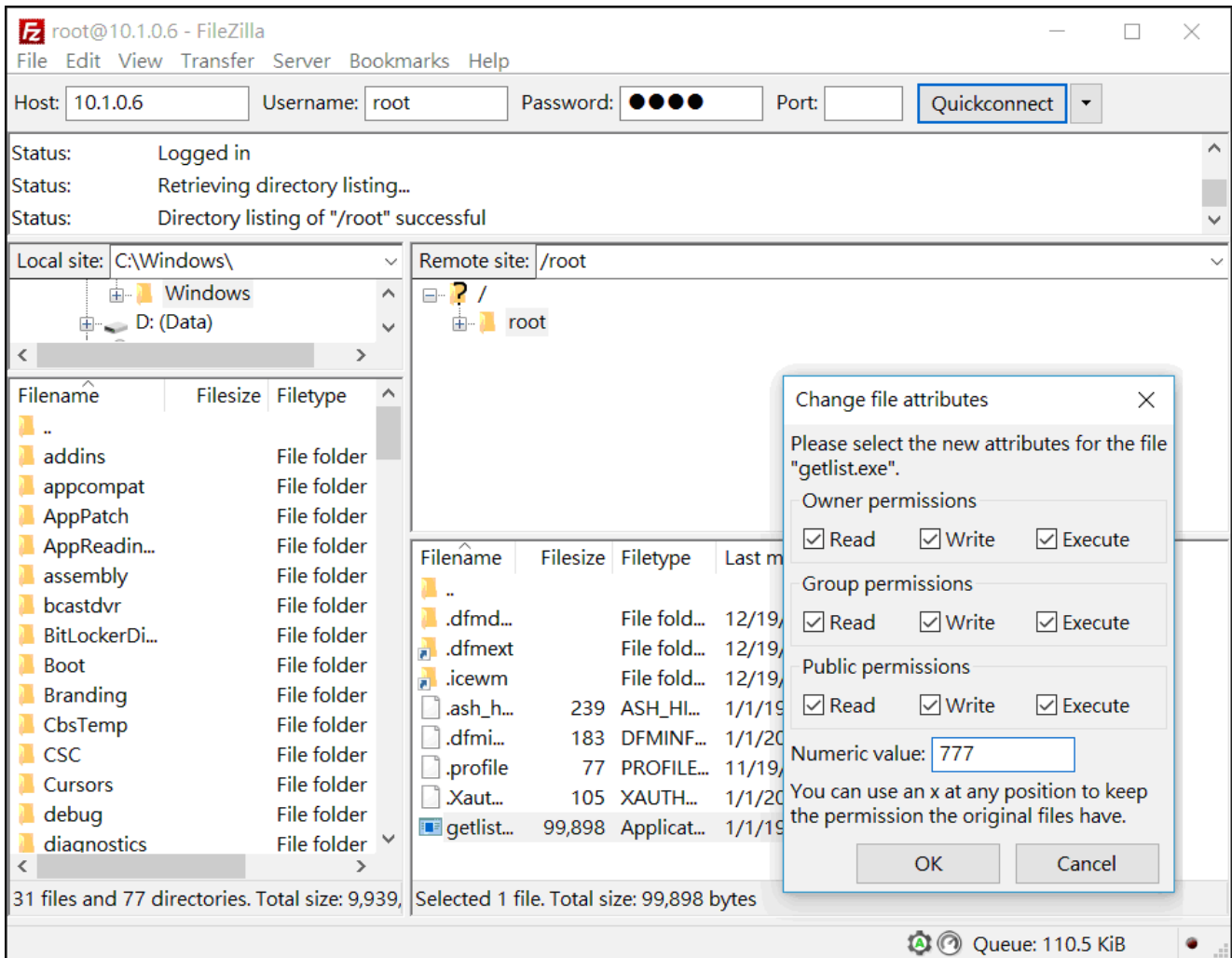


Fig. 2

Step 3: Running gdbserver on the LP-8x4x controller.

To use the server, log on to the target system, and run the gdbserver program.

When gdbserver is executed on the LP-8x4x, it passively waits for the host gdb to communicate with it, as illustrated in the Fig. 3. The usual syntax is:

Command: `gdbserver 10.1.0.61:2345 getlist.exe`

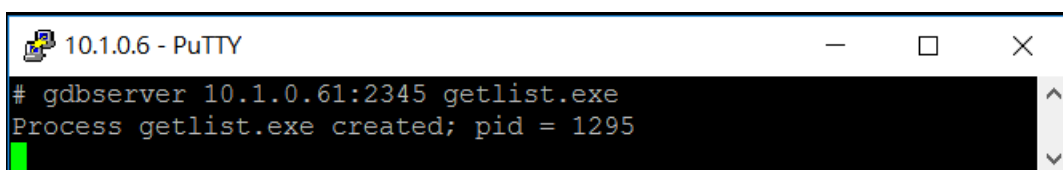


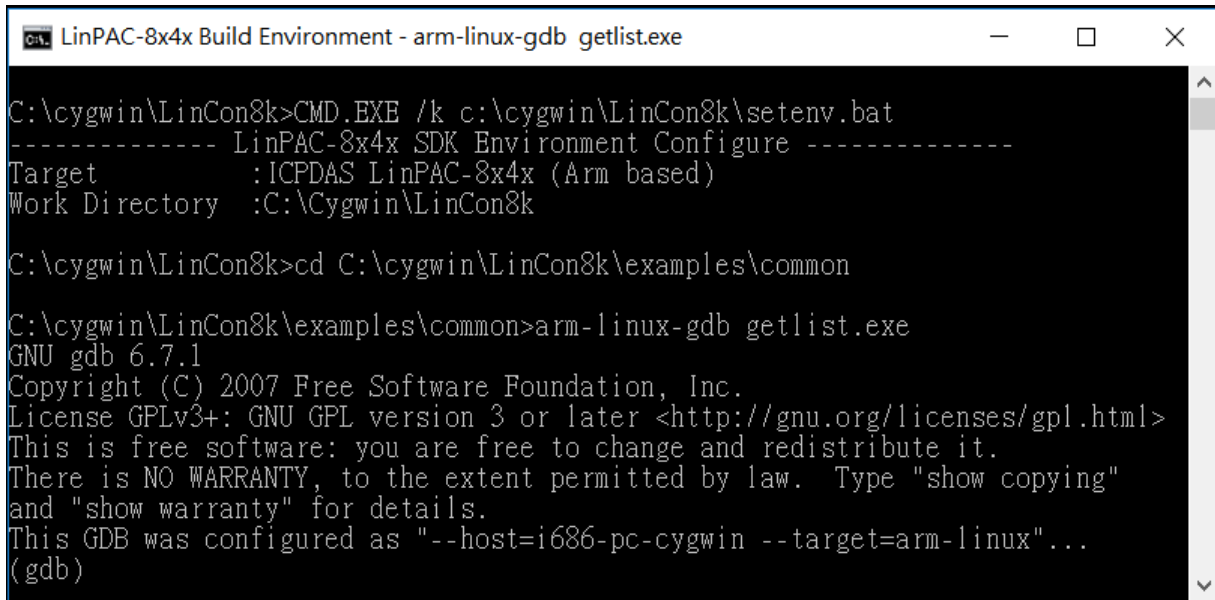
Fig. 3

Step 4: Remotely connecting to LP-8x4x Debugging using gdbserver

To perform remote debugging, start gdb and connect to the gdbserver.

The figures below illustrate the commands and processes used for debugging with gdb (Refer to Figs. 4 and 5 for more details).

Command: arm-linux-gdb getlist.exe



```

C:\cygwin\LinCon8k>CMD.EXE /k c:\cygwin\LinCon8k\setenv.bat
----- LinPAC-8x4x SDK Environment Configure -----
Target          :ICPDAS LinPAC-8x4x (Arm based)
Work Directory  :C:\Cygwin\LinCon8k

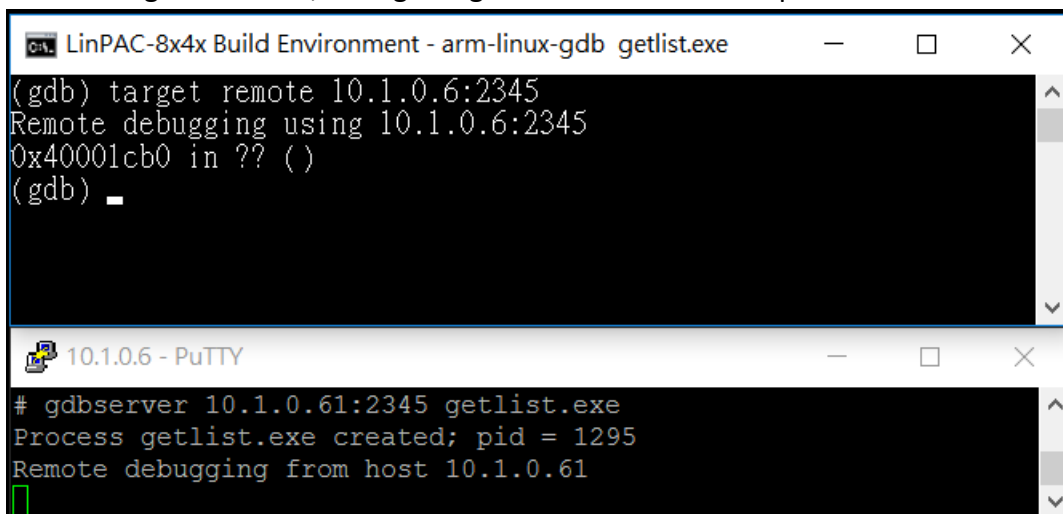
C:\cygwin\LinCon8k>cd C:\cygwin\LinCon8k\examples\common

C:\cygwin\LinCon8k\examples\common>arm-linux-gdb getlist.exe
GNU gdb 6.7.1
Copyright (C) 2007 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=arm-linux"...
(gdb)
  
```

Fig. 4

(gdb) target remote 10.1.0.6:2345

To open a remote debug connection, debug using a TCP connection to a port on the IP address.



```

(gdb) target remote 10.1.0.6:2345
Remote debugging using 10.1.0.6:2345
0x40001cb0 in ?? ()
(gdb) _

# gdbserver 10.1.0.61:2345 getlist.exe
Process getlist.exe created; pid = 1295
Remote debugging from host 10.1.0.61
  
```

Fig. 5

Breakpoints “b 30” and “b 6” have been included in the program (refer to Figs. 6 to Fig. 9), if we continue to execute the program, the output will be displayed on the target machine (refer to Fig. 10). To exit the gdb, type “q” (Quit).

(gdb) list

The “**list**” command allows you to display the demo code. By default, GDB prints ten source lines with any of these forms of the list command.



```

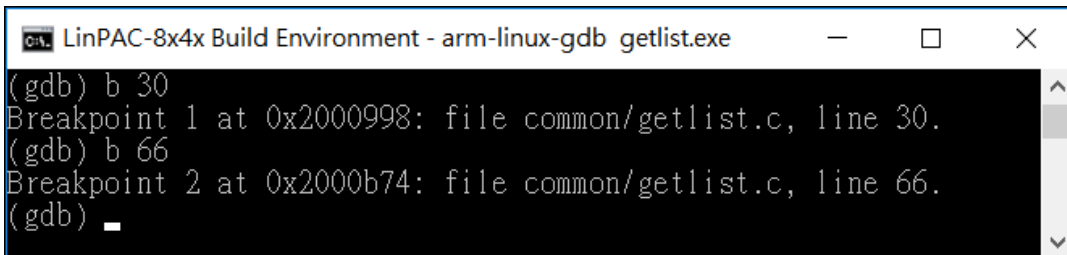
(gdb) list
21  #include "msw.h"
22
23  char szSend[80], szReceive[80];
24
25  int main()
26  {
27      int i, slot_count, wRetVal, wsRetVal;
28      //unsigned char serial_num[8];
29      WORD wT;
30      wRetVal = Open_Slot(0);
(gdb) list
31      Open_Slot(9);
32      szSend[0] = '$';
33      szSend[1] = '0';
34      szSend[2] = '0';
35      szSend[3] = 'M';
36      szSend[4] = 0;
37
38      if (wRetVal > 0) {
39          printf("open Slot failed!\n");
40          return (-1);
(gdb)

```

Fig. 6

(gdb) b 66

This command is used to set a breakpoint at line 66. To set a breakpoint on a different line, change the value “66” to the desired line number.



```

(gdb) b 30
Breakpoint 1 at 0x2000998: file common/getlist.c, line 30.
(gdb) b 66
Breakpoint 2 at 0x2000b74: file common/getlist.c, line 66.
(gdb) _

```

Fig. 7

(gdb) info break

This command is used to retrieve the status of user-defined breakpoints contained in the code and the breakpoint number.

```

LinPAC-8x4x Build Environment - arm-linux-gdb getlist.exe
(gdb) info break
Num Type           Disp Enb Address      What
1  breakpoint      keep y  0x02000998  in main at common/getlist.c:30
2  breakpoint      keep y  0x02000b74  in main at common/getlist.c:66
(gdb)

```

Fig. 8

(gdb) c

This command is used to continue debugging the program after a signal or breakpoint has been reached.

```

LinPAC-8x4x Build Environment
(gdb) c
Continuing.

Breakpoint 1, main () at common/getlist.c:30
30          wRetVal = Open_Slot(0);
(gdb) c
Continuing.

Breakpoint 2, main () at common/getlist.c:66
66          Close_Slot(0);
(gdb) c
Continuing.

Program exited normally.
(gdb) q

```

Fig. 9

Once the end of the program is reached, the output will be displayed on the target machine.

```

10.1.0.6 - PuTTY
# gdbserver 10.1.0.61:2345 getlist.exe
Process getlist.exe created; pid = 1295
Remote debugging from host 10.1.0.61
slot 1 ... not installed
slot 2 ... not installed
slot 3 ... not installed
slot 4 ... 87063
slot 5 ... not installed
slot 6 ... 8014
slot 7 ... not installed
slot 8 ... 87059

Child exited with retcode = 0

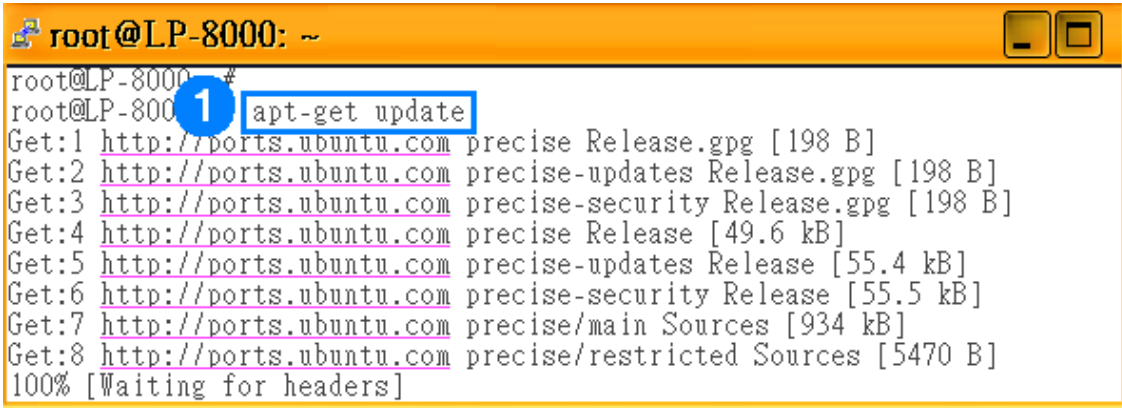
Child exited with status 28
GDBserver exiting
#

```

Fig. 10

Section 2: Working with the Code::Blocks IDE interface

At the first time, user needs to make sure the LinPAC controller have gdbserver tool. In the LP-8x2x, user can type command '**apt-get update**' and '**apt-get install gdbserver**' for install gdbserver tool, as illustrated in Fig. 11 and 12 below.

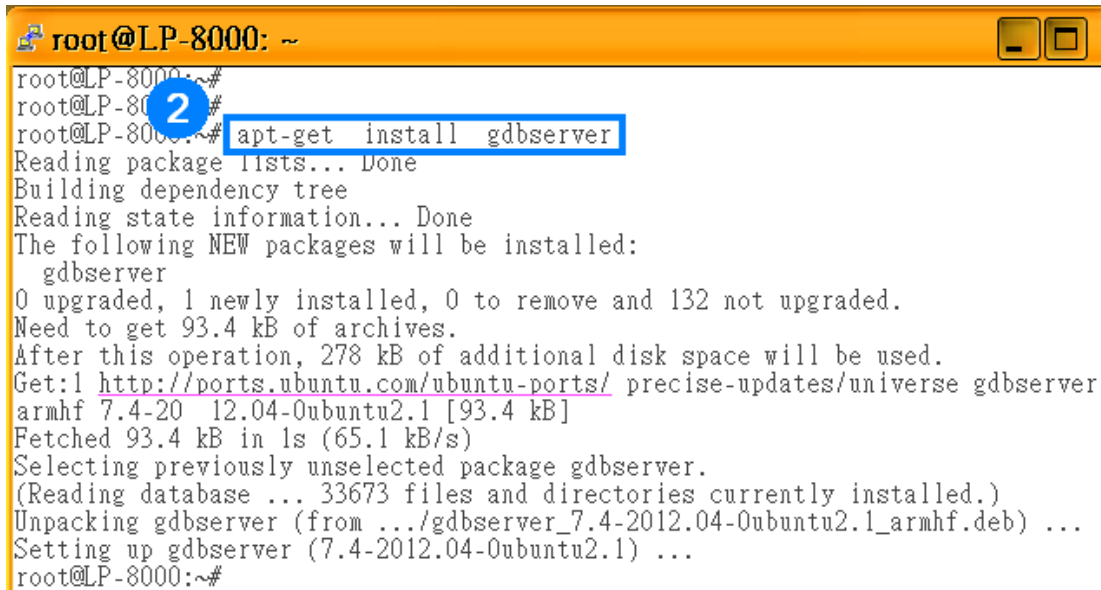


```

root@LP-8000: ~
root@LP-8000:~# apt-get update
Get:1 http://ports.ubuntu.com precise Release.gpg [198 B]
Get:2 http://ports.ubuntu.com precise-updates Release.gpg [198 B]
Get:3 http://ports.ubuntu.com precise-security Release.gpg [198 B]
Get:4 http://ports.ubuntu.com precise Release [49.6 kB]
Get:5 http://ports.ubuntu.com precise-updates Release [55.4 kB]
Get:6 http://ports.ubuntu.com precise-security Release [55.5 kB]
Get:7 http://ports.ubuntu.com precise/main Sources [934 kB]
Get:8 http://ports.ubuntu.com precise/restricted Sources [5470 B]
100% [Waiting for headers]

```

Fig. 11



```

root@LP-8000:~#
root@LP-8000:~# apt-get install gdbserver
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  gdbserver
0 upgraded, 1 newly installed, 0 to remove and 132 not upgraded.
Need to get 93.4 kB of archives.
After this operation, 278 kB of additional disk space will be used.
Get:1 http://ports.ubuntu.com/ubuntu-ports/ precise-updates/universe gdbserver
armhf 7.4-2012.04-0ubuntu2.1 [93.4 kB]
Fetched 93.4 kB in 1s (65.1 kB/s)
Selecting previously unselected package gdbserver.
(Reading database ... 33673 files and directories currently installed.)
Unpacking gdbserver (from .../gdbserver_7.4-2012.04-0ubuntu2.1_armhf.deb) ...
Setting up gdbserver (7.4-2012.04-0ubuntu2.1) ...
root@LP-8000:~#

```

Fig. 12

The following C program example will be used to demonstrate the remote debugging on the LP-8x2x with Code::Blocks IDE platform (<http://www.codeblocks.org/home>).

Step 1: To create a new project for remote debugging.

Click on the **File** pull-down menu, open **New** and then **Project**, and it will bring up the New from template window, as illustrated in Fig. 13 below.

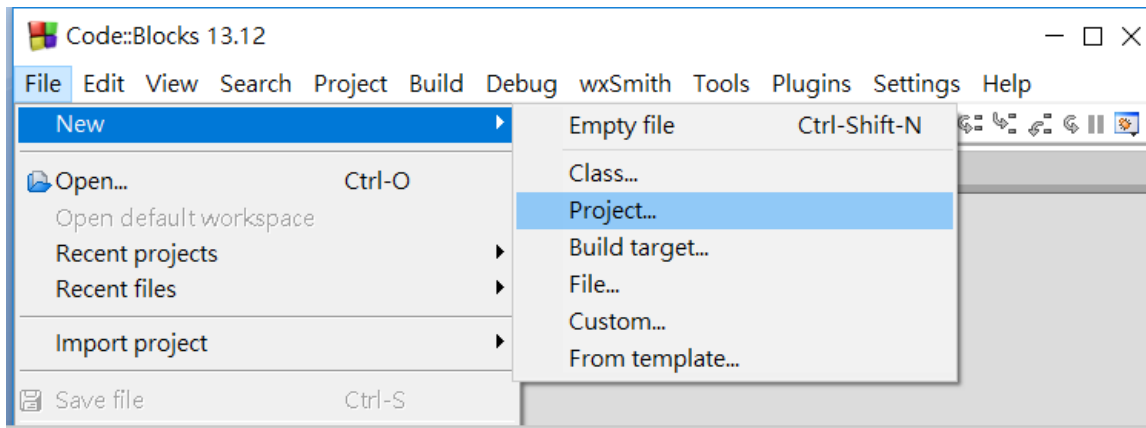


Fig. 13

Opening (clicking on) **Console Application** will then allow you to write a program on the console. The next window allows you to choose the language that you will use. Select the language as **C**, then press Finish, as illustrated in Fig. 14 below.

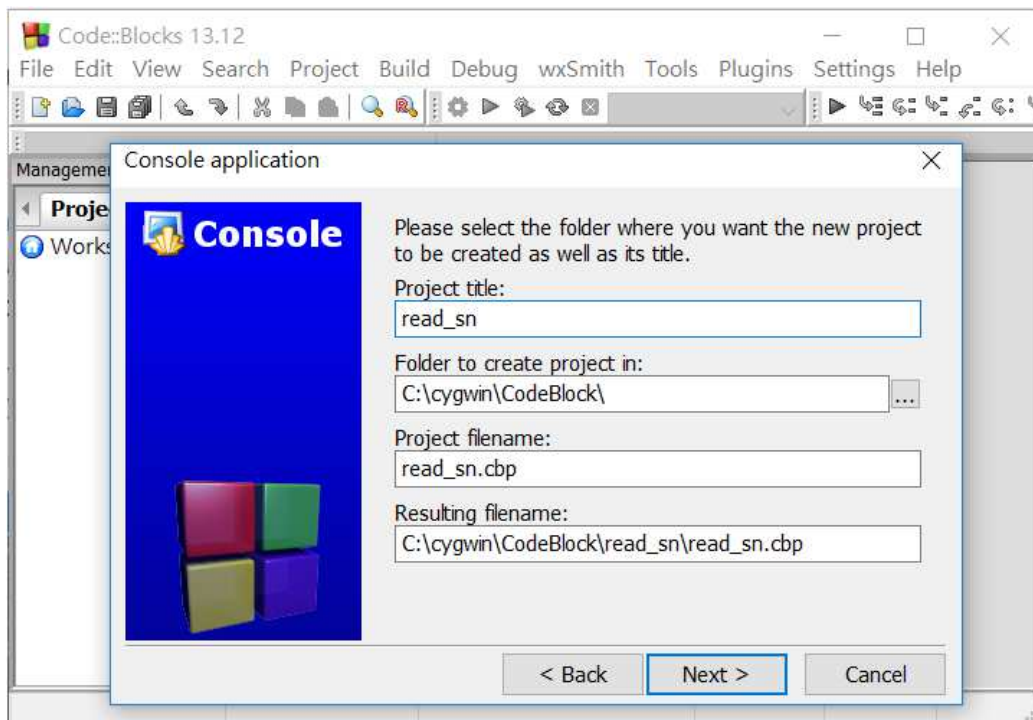


Fig. 14

Step 2: Setup a custom makefile.

To copy and modify **'Makefile'** file from C:\cygwin\LP-8x2x_SDK\ folder to debugger project (C:\cygwin\CodeBlock\read_sn) folder. Compile the C program with debugging option **-g**, as illustrated in Fig. 15 below.

```

*Makefile - Code::Blocks 13.12
File Edit View Search Project Build Debug wxSmith Tools Plugins Settings Help

*Makefile x
1 #SHELL = /bin/sh
2
3 LIBS = C:\cygwin\LP-8x2x_SDK\lib\libi8k.a
4 CFLAGS = -g -I. -I C:\cygwin\LP-8x2x_SDK\include
5 LDFLAGS = -lm
6
7 #GTKCFLAGS = -I. -I../include `PKG_CONFIG_PATH=/opt/lib/pkgconfig pkg-c
8 CROSS_COMPILE = arm-linux-gnueabihf-
9
10 AS = $(CROSS_COMPILE)as
11 LD = $(CROSS_COMPILE)ld
12 CC = $(CROSS_COMPILE)gcc
13 CPP = $(CC) -E
14 AR = $(CROSS_COMPILE)ar
15 NM = $(CROSS_COMPILE)nm
16 STRIP = $(CROSS_COMPILE)strip
17 OBJCOPY = $(CROSS_COMPILE)objcopy
18 OBJDUMP = $(CROSS_COMPILE)objdump
19
20 Debug: read_sn
21
22 read_sn: ./read_sn.o
23     $(CC) $(CFLAGS) $(LDFLAGS) -o ./bin/Debug/$@.exe ./read_sn.o $(LIBS)
24     rm -f ./read_sn.o

```

Fig. 15

Open '**Project--> Properties**' to access the main properties of the active project. And user will see a tick box for 'this is a custom makefile'.

Tick this box, make sure the name just above it is the one you want for your makefile, and change the current working directory, as illustrated in Fig. 16 below.

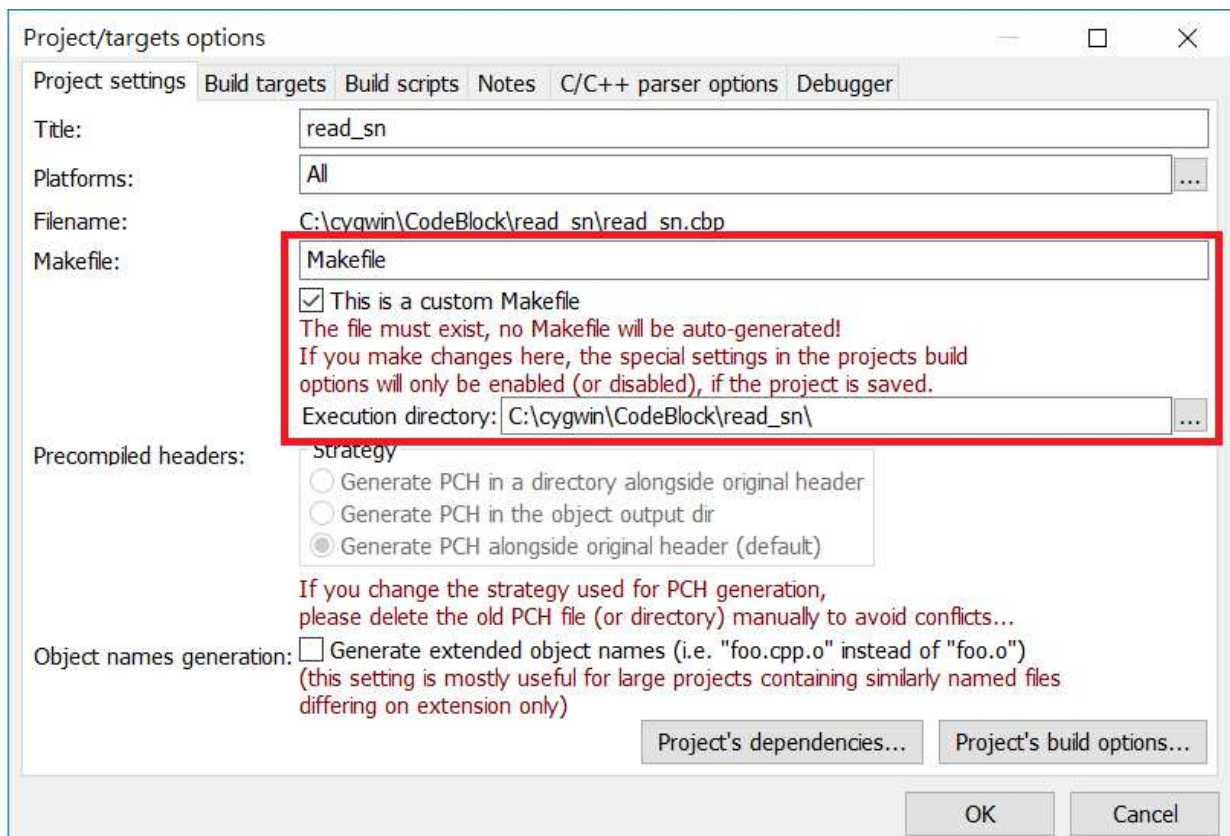


Fig. 16

In addition to customize your IDE, user also double click the default debug project file - read_sn, as illustrated in Fig. 17 below.

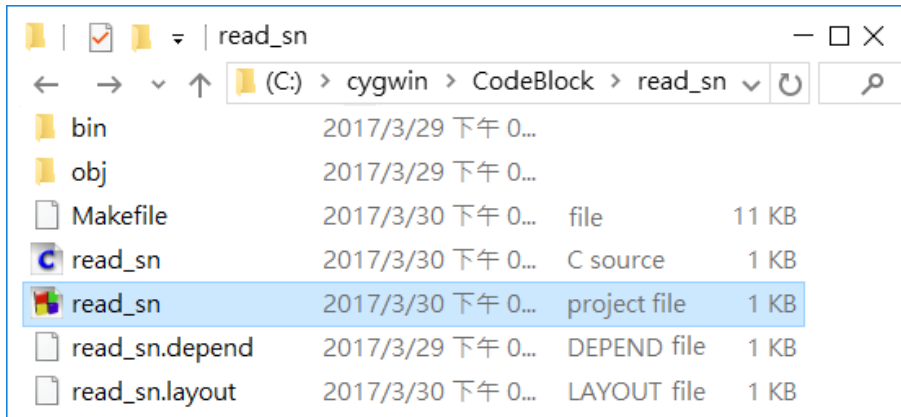


Fig. 17

Step 3: Click '**Settings->Debugger**' to make sure the executable path of default debugger is set to **arm-linux-gnueabi-gdb.exe** (as illustrated in Fig. 18).

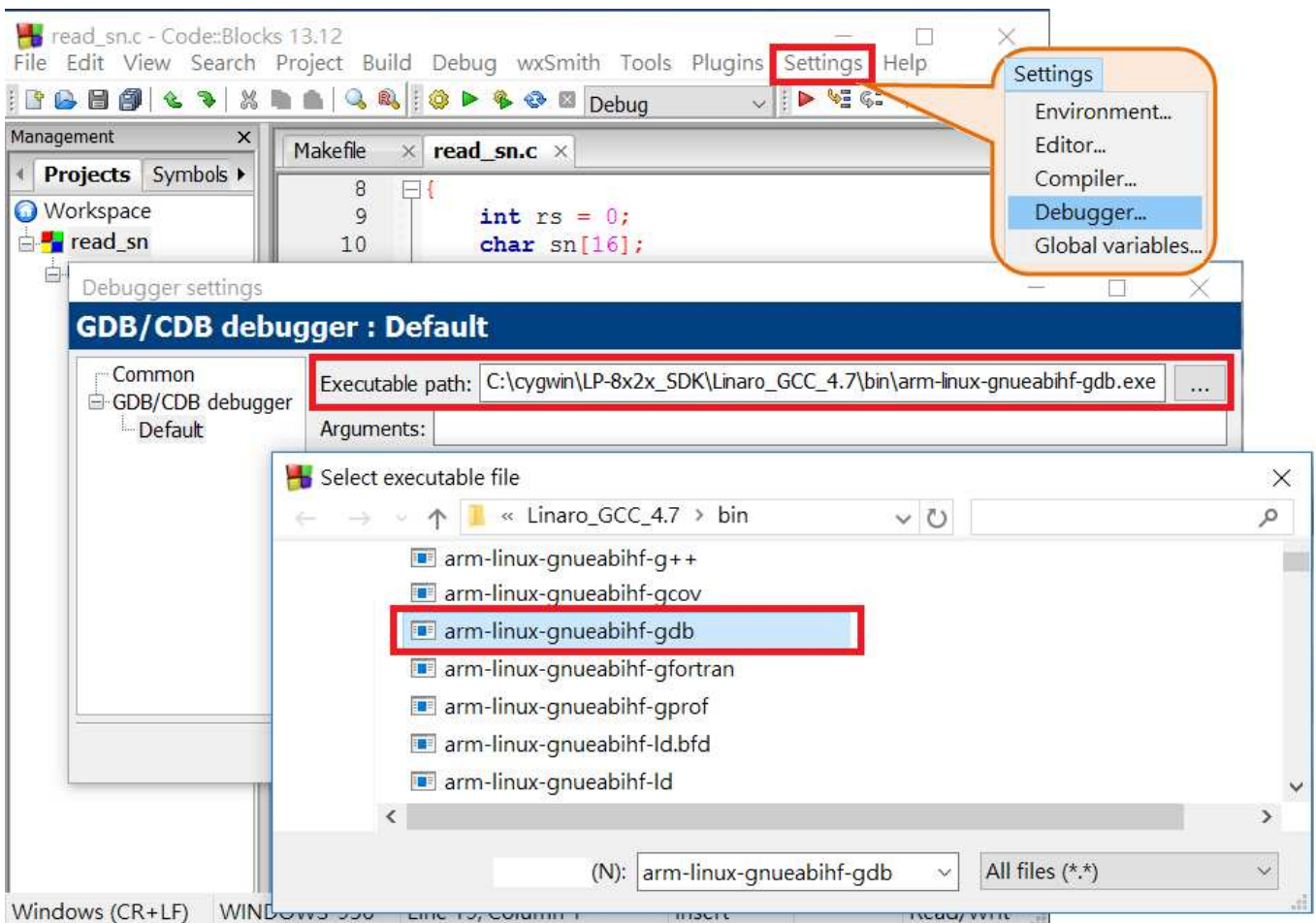


Fig. 18

Step 4: As your programs become more complicated, there will be a need to trace the program execution step by step or place break points where you wish the program to pause, as illustrated in Fig. 19 below.

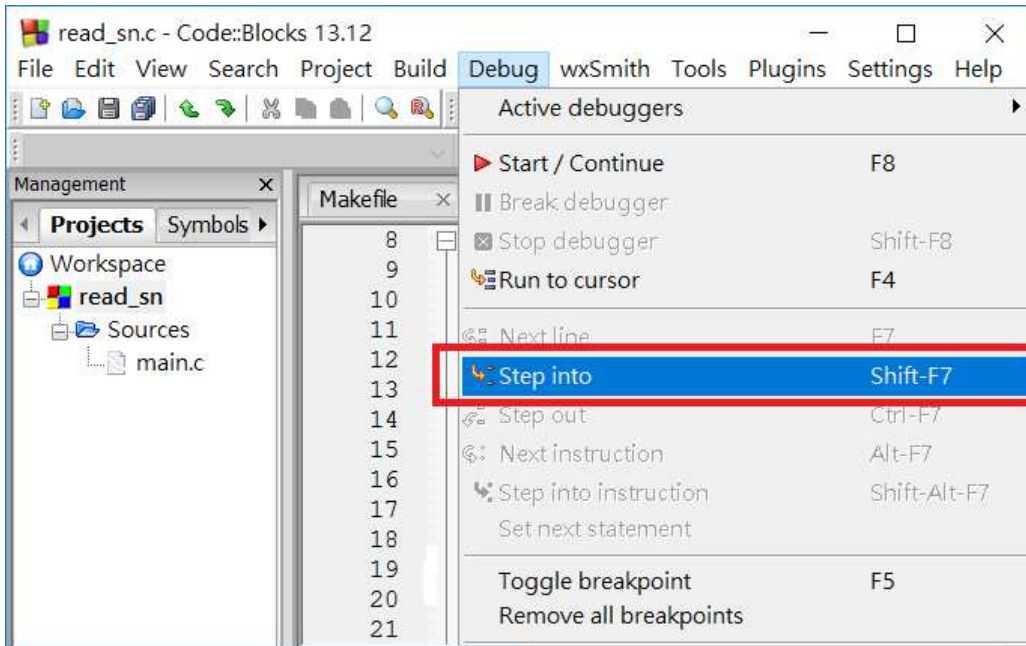


Fig. 19

Step 5: Make sure the remote debugger is running on the target machine.

To start remote debugging, a debugger running on host machine (local PC) connects to a program which is running on the target (LinPAC controller) via network (as illustrated in Fig. 20 and 21), that can be traced “line by line” while watching what happens as each line of code is executed(as illustrated in Fig. 22), type the command as follows:

- On target system (LP-8x2x): gdbserver 10.1.0.37:1234 read_sn.exe
- On host system (local PC): target remote 10.1.0.109:1234

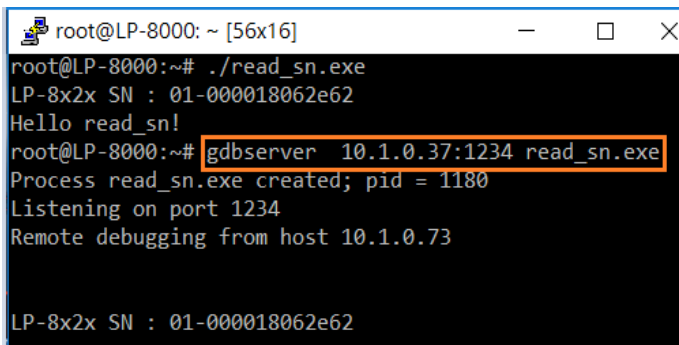


Fig. 20

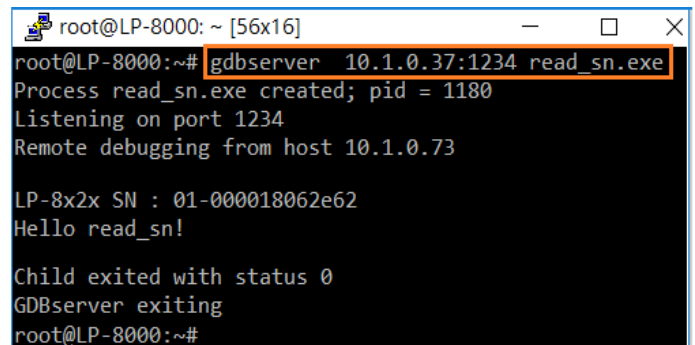


Fig. 21

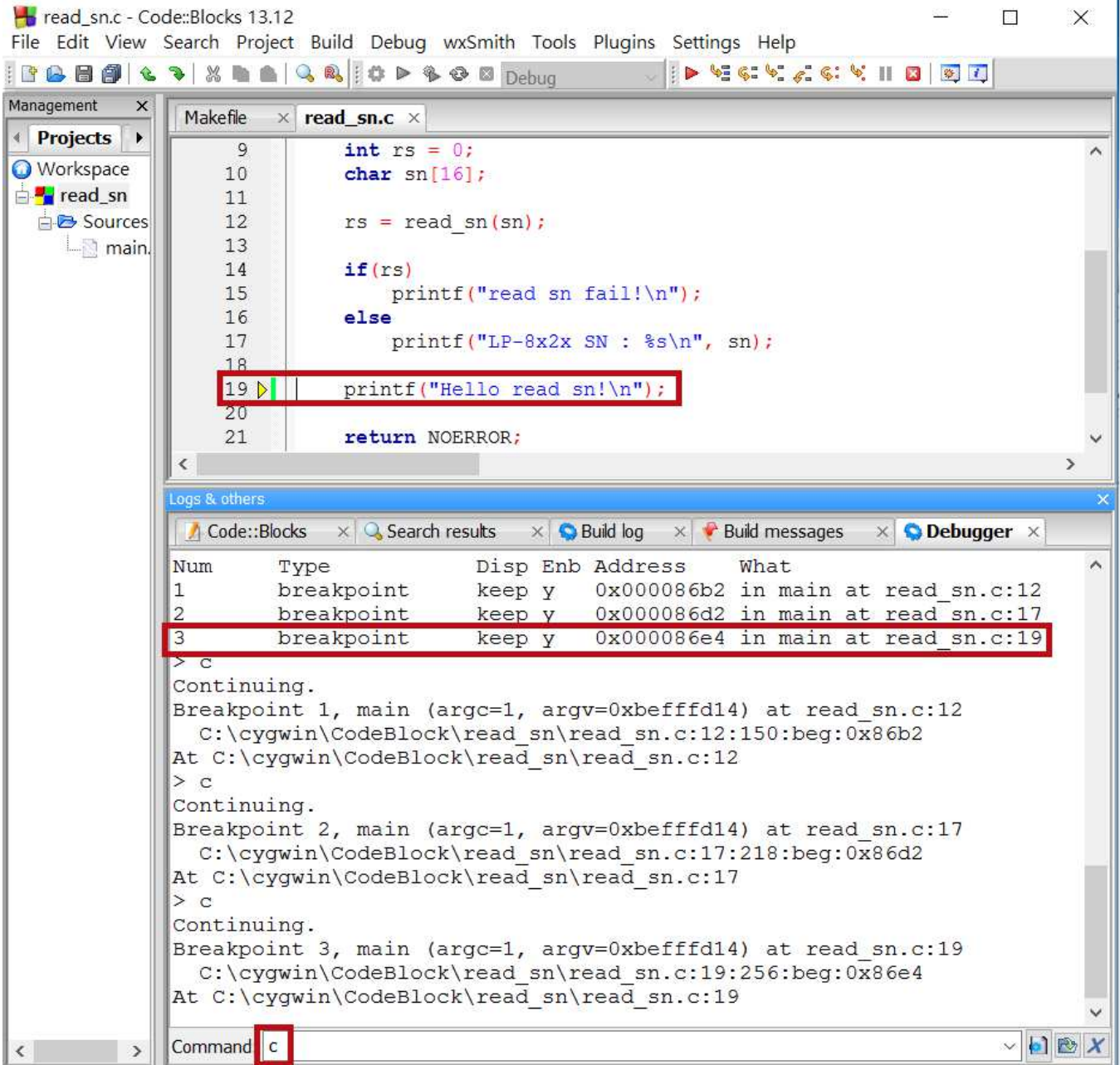


Fig. 22