OpenEuler | Compiler SIG

LLVM构建openEuler技术方案兼容性问题分享

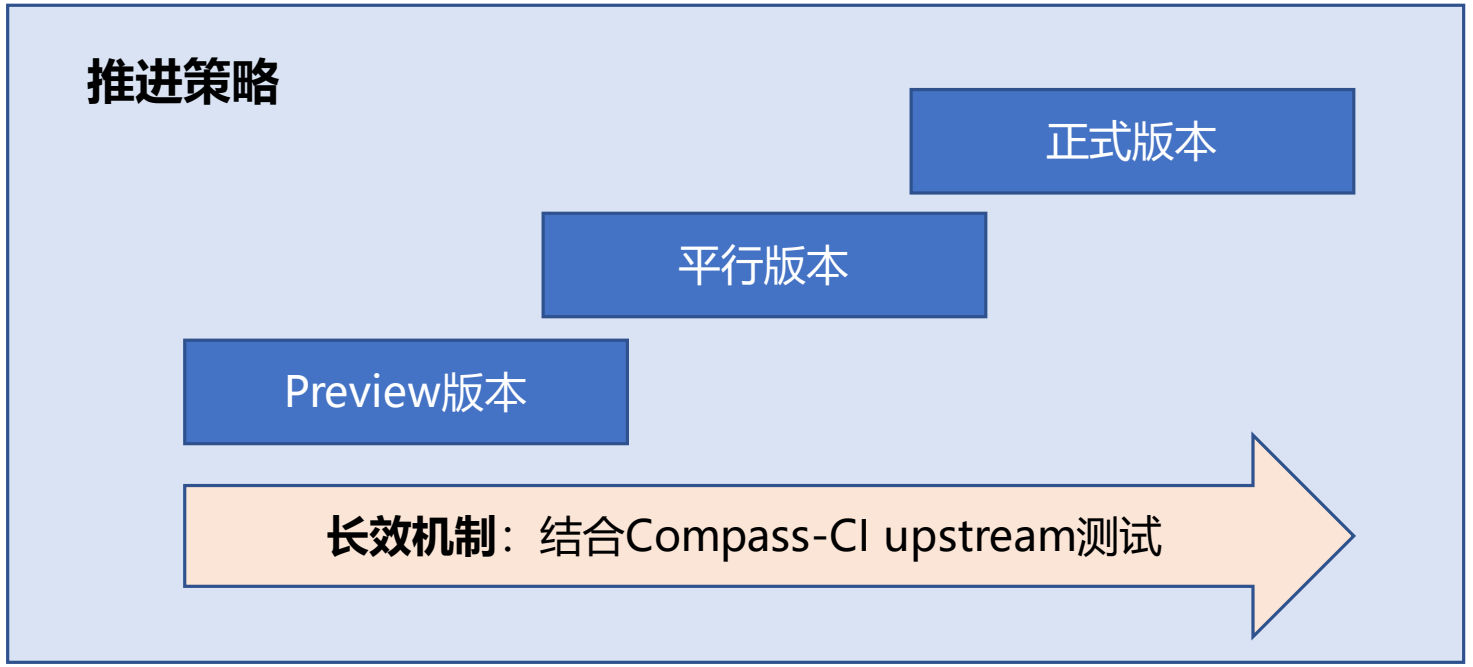
LLVM平行宇宙计划

目录

OpenEuler
Compiler SIG

LLVM平行宇宙计划：版本与长效机制结合，社区化推进

OpenEuler Compiler SIG



https://gitee.com/openeuler/compiler-docs/tree/master/LLVM%20Parallel%20Universe%20Project

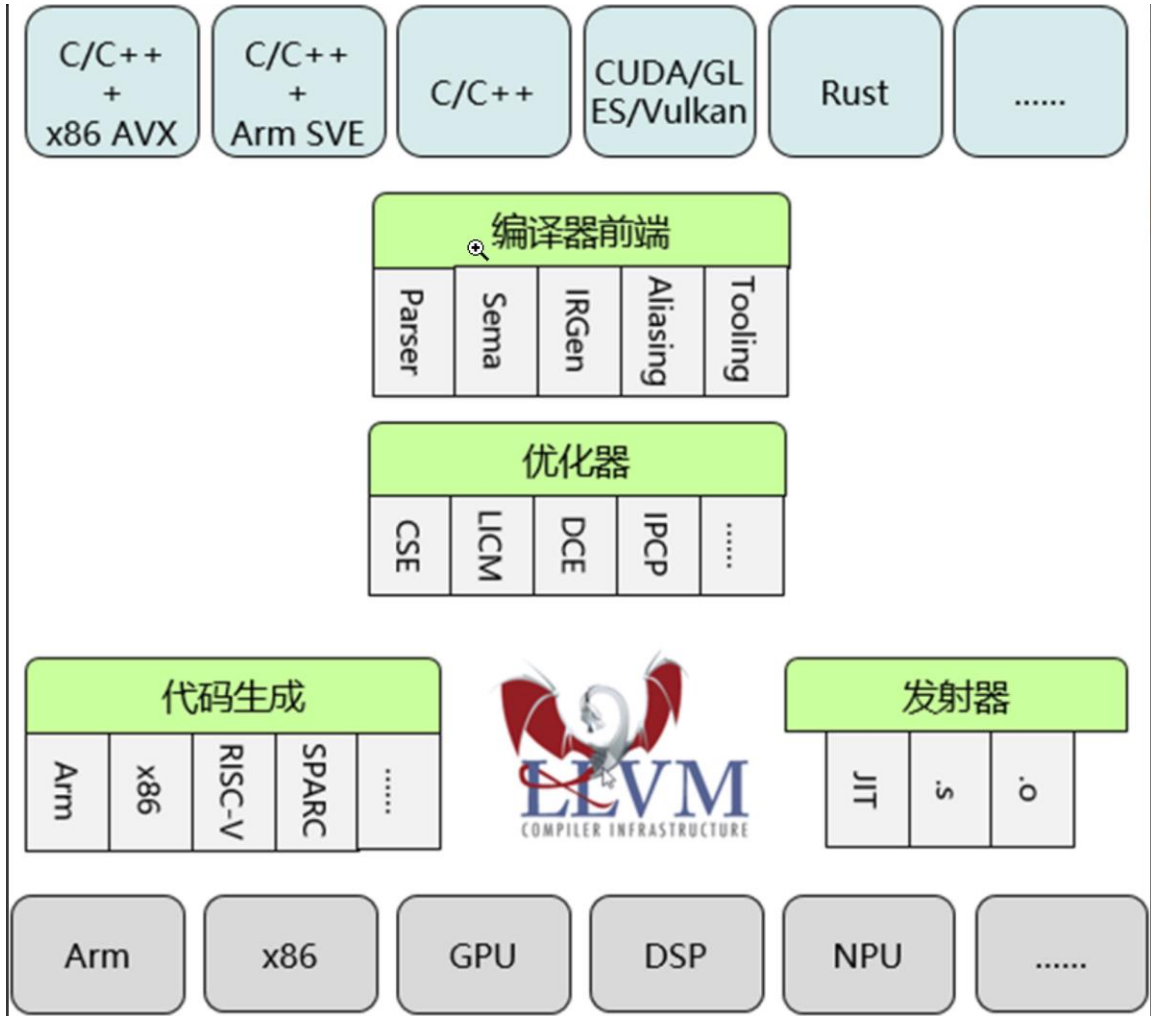
关键计划&进展：

- 中科院软件所牵头发布RISCV平台上openEuler 24.03 llvm preview版本。（530）
- 华为毕昇编译器团队与嵌入式SIG合作发布openEuler 24.03 Embedded llvm创新平行版本（530）
- 云场景全栈极致优化，全优化对象（应用、库、内核），全优化链路（LTO、PGO、BOLT）（930）

LLVM架构描述

OpenEuler
Compiler SIG

LLVM采用了**模块化架构设计**，将编译过程分为多个独立阶段，如前端、优化和后端。这种设计使得LLVM更加灵活和可扩展，有助于各阶段模块分别演进创新，而通过**统一的IR表示**又将不同的模块有机的结合起来。目前LLVM项目包含多个子项目，如clang、flang、llvm、mlir、lld等。LLVM 9.0版本之后采取**Apache License。**



LLVM子项目简介

OpenEuler
Compiler SIG

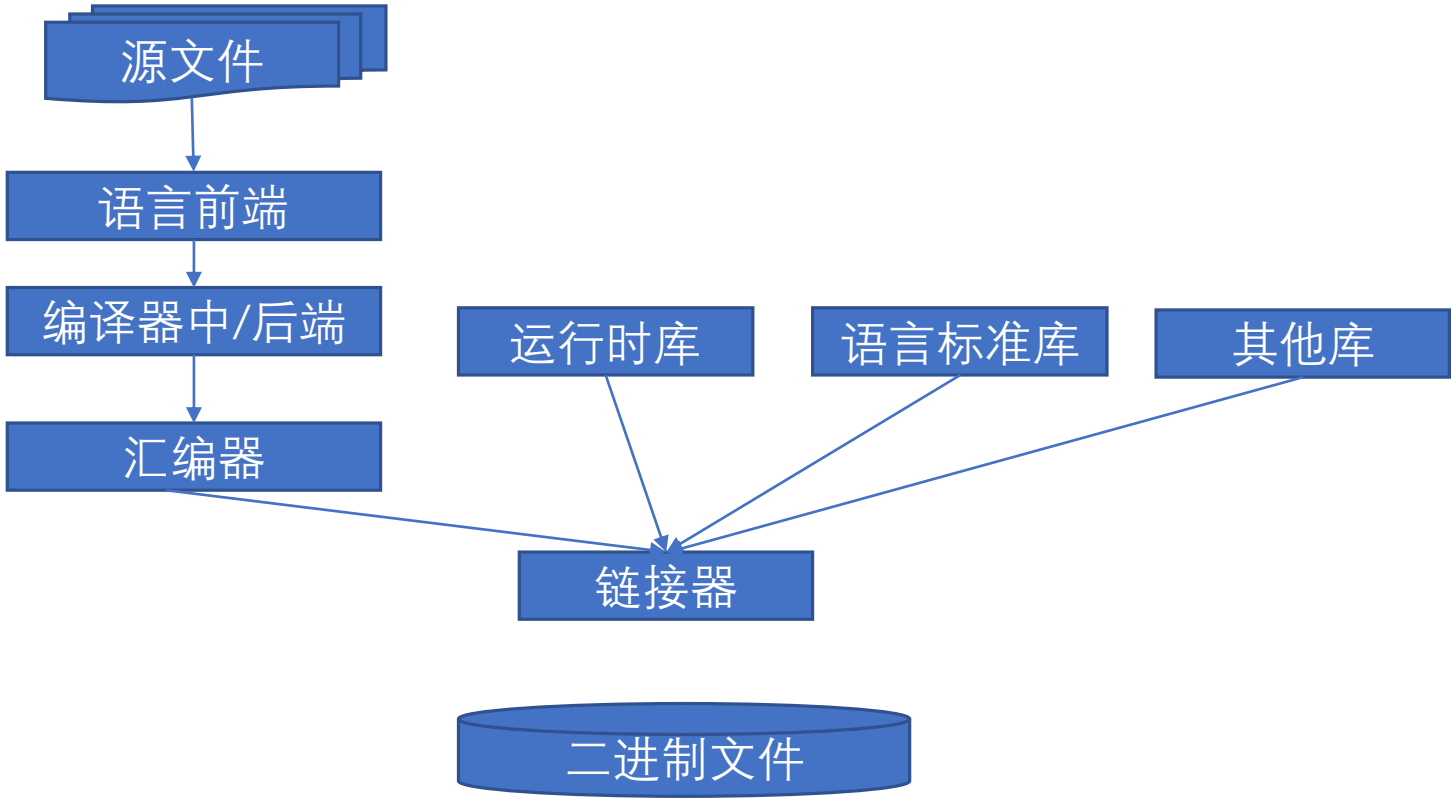
LLVM是一个伞形项目，包含模块化和可重复使用的编译器和工具链技术的集合。 LLVM的主要子项目有：

- **LLVM Core**：LLVM核心库提供了一个现代的独立于源码和目标的优化器，以及对许多主流CPU的代码生成支持。
- **Clang**：LLVM原生的C-family语言编译器。
- **LLDB**：构建在LLVM和Clang提供的库之上，以提供一个出色的原生调试器。
- **libc++和libc++ ABI**：提供一个符合标准且高性能的C++标准库实现。
- **compiler-rt**：编译器运行时库，包含一些低级别的代码生成的支持函数，也为动态测试工具提供运行时库。
- **MLIR**：一种构建可重用和可扩展编译器基础设施的新方法。
- **OpenMP**：提供了一个OpenMP运行时实现。
- **polly**：使用多面体模型实现了一系列Cache局部性优化以及自动并行和矢量化优化。
- **libclc**：OpenCL标准库的一个实现。
- **klee**：符号执行虚拟机的一个实现。
- **LLD**：LLVM原生链接器。
- **BOLT**：后链接优化器，通过优化应用程序的代码布局来达成优化效果。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

- Clang只是C-family编程语言完整工具链中的一个组件。为了组装一个完整的工具链，需要额外的工具和运行时库。
- Clang被设计为与用于其目标平台的现有工具和库进行交互操作。
- 并且LLVM项目为许多这些组件**提供了替代方案**。



基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

Clang前端

Clang前端(clang -cc1)用于编译C-family语言。Clang前端的命令行接口被视为实现细节，故没有外部文档，并且可以在提示的情况下进行更改。

汇编器

Clang既可以使用LLVM项目的集成汇编器，也可以使用外部特定于系统的汇编器。

如果想使用特定系统的汇编器，请使用-fno-integrated-as选项。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

链接器

Clang可以配置为使用几个不同的链接器其中一个：

GNU ld

GNU gold

LLVM lld

MSVC link.exe

可以通过-fuse-ld=<linker name>标志来切换。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

运行时库

C-family程序需要许多不同的运行时库提供不同的支持。Clang将隐式地链接每个运行时库的合适实现。

编译器运行时

compiler-rt (LLVM): LLVM项目的编译器运行时库提供了一组完整的运行时库函数。

libgcc_s (GNU)：GCC编译器的运行时库可以用来代替compiler-rt。但是，它缺少几个LLVM可能调用的函数，特别是在使用Clang的内置函数家族的__builtin_*_overflow时。

可以通过--rtlib=compiler-rt或--rtlib=libgcc来切换编译器运行时库

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

原子库

- compiler-rt (LLVM)：LLVM项目的原子库的实现包含在compiler-rt中。
- libatomic (GNU)：libgcc_s不提供原子库的实现，事实上，GCC的libatomic library在使用libgcc_s时被提供。

注意： 当Clang使用libgcc_s时，目前**不会自动链接到libatomic**。在使用非compiler-rt提供的原子操作时（如果您看到引用了__atomic_*函数的链接错误），可能需要手动添加-latomic来支持这种配置。。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

Unwind库

Unwind库提供了一系列_Unwind_*函数。

- libunwind (LLVM)：LLVM项目的Unwind库一种实现。
- libgcc_s (GNU)：libgcc_s有一个集成的Unwinder，不需要提供外部的Unwind库。
- libunwind (nongnu.org)：这是Unwind规范的另一个实现。请参阅libunwind(nongnu.org)。
- libunwind (PathScale): 这是Unwind规范的另一个实现。请参阅libunwind (pathscale)。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

C标准库

- libc（LLVM）
- glibc（GNU）

C++标准库

- libc++ (LLVM)：libc++是LLVM项目的C++标准库实现，旨在从C++ 11开始成为全面的C++标准实现。
- libstdc++ (GNU)：libstdc++是GCC 的C++标准库实现，Clang支持各种版本的libstdc++。

可以通过-stdlib=libc++或-stdlib=libstdc++选项来切换C++标准库。

基于Clang组装一个完整工具链

OpenEuler
Compiler SIG

C++ ABI库

C++ ABI库提供了Itanium C++ ABI库部分的实现。

- libc++ abi (LLVM)：libc++abi是LLVM项目对该规范的实现。
- libsupc++ (GNU)：libsupc++是GCC对该规范的实现。但是，只有在静态链接libstdc++时才使用这个库。libstdc++的动态库版本包含libsupc++的一个副本。
- libcxxrt (PathScale)：这是Itanium C++ ABI规范的另一个实现。

注意 虽然同一个程序可能同时使用libstdc++和libc++（只要您不试图将C++标准库对象传递到边界之外），但是在一个程序中通常不可能有一个以上的C++ABI库。

LLVM构建openEuler技术方案（中间态）

OpenEuler
Compiler SIG

openEuler软件包形态

服务器版本

用户态

3.6W+ 软件包

内核态

kernel | module | driver

嵌入式版本

用户态

百级软件包

内核态

kernel | module | driver

构建技术方案

内核态全部用LLVM构建，用户态混编

用户态：LLVM 构建 | GCC 构建

内核态：LLVM 构建（kernel | module | driver）

编译器组件切换方案

clang | llvm | as | lld

libstdc++ | ligcc_s | libatomic | GNU binutils

LLVM工具链全栈切换构建

目录

OpenEuler
Compiler SIG

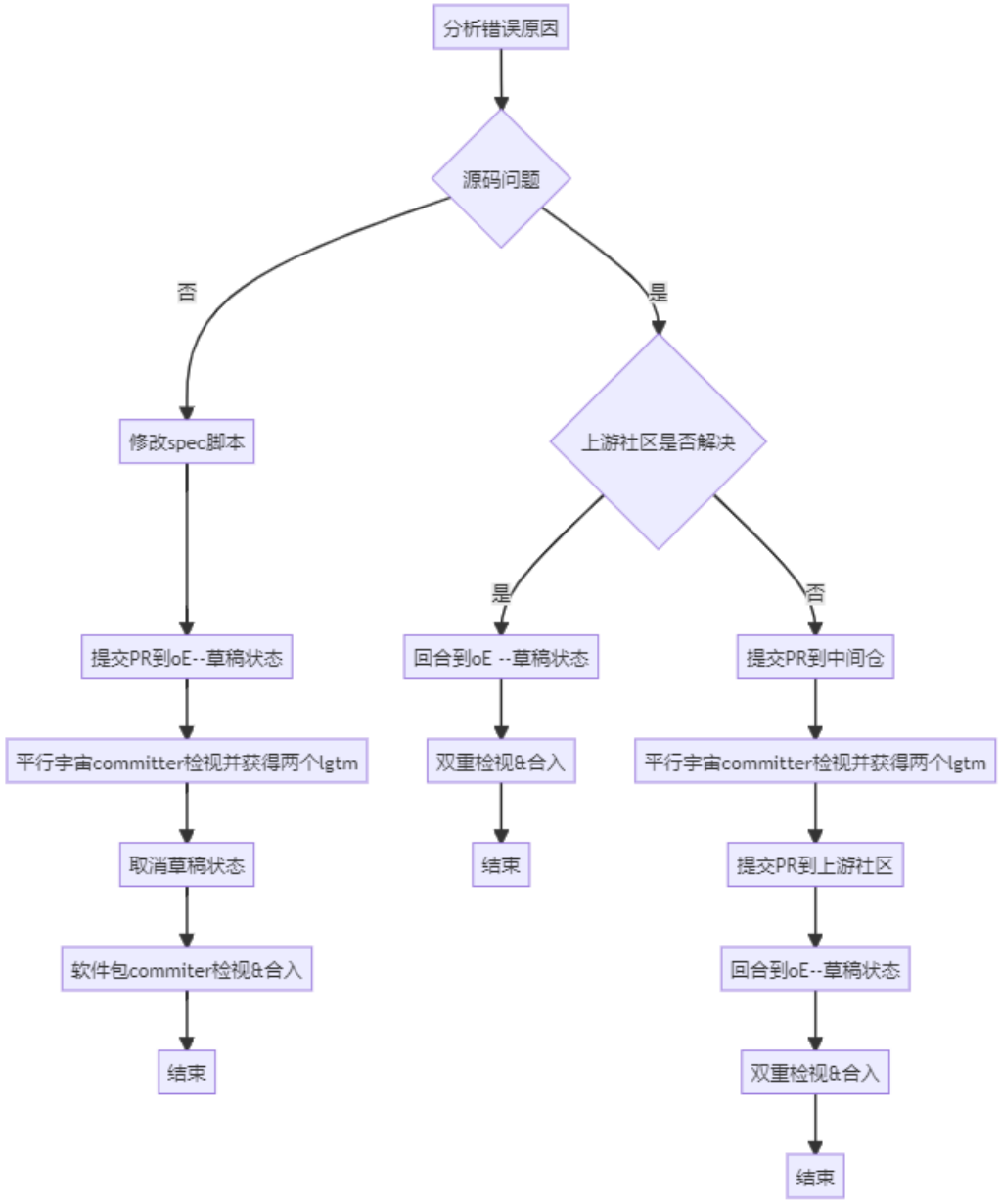
软件包构建及问题处理原则与流程

OpenEuler

r SIG

总体上，问题修复秉持**Upstream first**原则，主要原因如下：

- openEuler作为软件包的下游社区，测试能力有限，很难完成对软件包源码修改的全面测试。
- 在上游社区修改问题是在源头解决问题，可以避免在openEuler的后续版本多次修改。
- 在上游社区修改问题可以帮助提升开发者的能力和影响力。



软件包构建问题分类

- 写死GCC编译器。包括spec文件、makefile、其他语言与C互调用编译器。
- LLVM相较GCC更多warning。如更严格的标准遵从、更详细的检查、不支持选项等。
- LLVM不支持某些GNU扩展
- 编译器特性差异
- 未定义行为导致的构建/运行错误
- 重定位类型兼容问题
- ……

软件包构建问题—写死GCC编译器

OpenEuler
Compiler SIG

问题分类	详细说明	措施
构建脚本写死GCC	构建脚本Spec、Makefile、configure、CMakeLists.txt写死编译器	修改构建脚本Spec、Makefile、configure、CMakeLists.txt，使用CC或__cc。

```
265 265   export CFLAGS="$RPM_OPT_FLAGS"
266 266   export LDFLAGS="$RPM_LD_FLAGS"
267       - export CC="gcc"
268       - export CC_FOR_TARGET="gcc"
    267   + export CC="%{__cc}"
    268   + export CC_FOR_TARGET="%{__cc}"
269 269   export GOOS=linux
```

```
289       - ./bootstrap.sh
    289   + ./bootstrap.sh --with-toolset=%{__cc}
290 290   %define opt_build -d+2 -q %{?_smp_mflags} --nc
291 291   %define opt_feature release debug-symbols=on p
292 292   %define opt_libs --without-mpi --without-graph
```

```
    52  + %if "%toolchain" == "clang"
    53  +     %global make_opts HOSTCC=clang CC=clang CXX=clang++
    54  + %endif
52  55
53      - %make_build CPU="generic" TARGET="linux-glibc" USE_OPENSSL=1 USE_PCRE2=1 USE_SLZ=1 \
    56  + %make_build %{?make_opts} CPU="generic" TARGET="linux-glibc" USE_OPENSSL=1 USE_PCRE2=1
54  57      USE_LUA=1 USE_CRYPT_H=1 USE_SYSTEMD=1 USE_LINUX_TPROXY=1 USE_GETADDRINFO=1 USE_PROM
55  58      ADDINC="%{build_cflags}" ADDLIB="%{build_ldflags}"
56  59
57      - %make_build admin/halog/halog ADDINC="%{build_cflags}" ADDLIB="%{build_ldflags}"
    60  + %make_build %{?make_opts} admin/halog/halog ADDINC="%{build_cflags}" ADDLIB="%{build_ld
58  61
59  62  pushd admin/iprange
60      - %make_build OPTIMIZE="%{build_cflags}" LDFLAGS="%{build_ldflags}"
    63  + %make_build %{?make_opts} OPTIMIZE="%{build_cflags}" LDFLAGS="%{build_ldflags}"
```

```
178       - make CFLAGS="%{optflags}" %{?_smp_mflags}
    178   + make CC=%{__cc} CFLAGS="%{optflags}" %{?_smp_mflags}
179 179
```

软件包构建问题—写死GCC编译器

OpenEuler
Compiler SIG

问题分类	详细说明	措施
其他语言与C/C++互调用写死GCC	构建perl-xxx外围包，依赖perl包生成makefile，这时会沿用perl构建时的编译器; 同理，构建python-xxx外围包也会沿用构建python3时的编译器	先切换perl、python3的编译器为LLVM，然后构建perl-xxx、python-xxx外围包

- **包名称:** perl-Compress-Raw-Bzip2
- **报错现象:**

  gcc构建不识别llvm的选项
- **根本原因:**

  perl构建时配置的编译器被记录
- **措施:**

  先用llvm构建perl编译器

日志报错:
gcc_old: error: unrecognized command-line option '--config'; did you mean '-mpconfig'?

依赖perl生成Makefile

```
# perl-Compress-Raw-Bzip2.spec
perl Makefile.PL INSTALLDIRS=vendor OPTIMIZE="%{optflags}" NO_PACKLIST=1
```

 # 使用clang构建的perl

```
perl -V | grep -B3 -w cc
    usemymalloc=n
    default_inc_excludes_dot=define
  Compiler:
    cc='clang'
```

 # 使用gcc构建的perl

```
perl -V | grep -B3 -w cc
    usemymalloc=n
    default_inc_excludes_dot=define
  Compiler:
    cc='gcc'
```

clang构建的perl生成的Makefile

```
SUFFIX=>q[gz], TARFLAGS=>q[-chvf] }

# --- MakeMaker post_initialize section:


# --- MakeMaker const_config section:

# These definitions are from config.sh (via /usr/lib64/perl5/Config.pm).
# They may have been overridden via Makefile.PL or on the command line.
AR = ar
CC = clang
CCCDLFLAGS = -fPIC
CCDLFLAGS = -Wl,--enable-new-dtags -Wl,-z,relro -Wl,-z,now
```

gcc构建的perl生成的Makefile

```
SUFFIX=>q[gz], TARFLAGS=>q[-chvf] }

# --- MakeMaker post_initialize section:


# --- MakeMaker const_config section:

# These definitions are from config.sh (via /usr/lib64/perl5/Config.pm).
# They may have been overridden via Makefile.PL or on the command line.
AR = ar
CC = gcc
CCCDLFLAGS = -fPIC
CCDLFLAGS = -Wl,--enable-new-dtags -Wl,-z,relro -Wl,-z,now
```

软件包构建问题—LLVM相较GCC更多warning

OpenEuler Compiler SIG

https://clang.llvm.org/docs/DiagnosticsReference.html

选项/告警	类型	软件包
Wunknown-warning-option Wunused-parameter Wunused-function Wunused-but-set-parameter Wunused-but-set-variable Wunused-command-line-argument Wignored-optimization-argument	clang warning	软件包较多
Wimplicit-function-declaration	clang error / gcc warning（clang17新增）	multipath-tools、openssh、sblim-sfcc、tcp_wrappers、tftp、trace-cmd、unixODBC、xinetd、xorg-x11-drv-nouveau、netperf、mcpp、lvm2、libxslt、libevent、libesmp、ipmitool、gnome-vfs2、cvs、createrepo_c、accountsservice
Wincompatible-function-pointer-types	clang error / gcc warning（clang17新增）	pkcs11-helper、perl-XML-LibXML、setroubleshoot、telepathy-glib、vinagre、libsecret、libgee、gnome-font-viewer、freetype、freerdp、alsa-tools
Wregister	clang error / gcc warning（clang17新增）	tog-pegasus、lshw、djvulibre
Wint-conversion	clang error / gcc warning	zziplib、libdb、cups
Wdeprecated-non-prototype	clang特有告警	opensc、libreswan
Wimplicit-int	clang error / gcc warning	rarian、xmlto
Wunknown-warning-option	兼容性相关告警	pesign
Wcast-align	warning（clang更严格）	vdo
Wenum-constexpr-conversion	clang error / gcc warning（clang17新增）	webkit2gtk3
Wunsafe-buffer-usage	clang特有告警	libipt
Wunused-command-line-argument	兼容性相关告警	acpica-tools
Wenum-conversion	warning（clang更严格）	ltrace
Wstring-plus-int	clang特有告警	libstoragemgmt
Wignored-attributes	兼容性相关告警	lcr
Wunused-result	warning	edk2
Wreturn-type	clang error / gcc warning（clang17新增）	dmraid

软件包构建问题—LLVM不支持某些GNU扩展

OpenEuler
Compiler SIG

问题分类	详细说明	措施
其他问题	2、clang不知道GNU扩展的__va_arg_pack等导致的问题	具体情况具体分析

- **包名称:**

dhcp、crash、cyrus-sasl

- **报错现象:**

error: expected parameter declarator

extern int asprintf(char **strp, const char *fmt, ...);

- **问题根因:**

llvm不支持GNU扩展__va_arg_pack

- **措施:**

修改源码包，屏蔽软件包内部的实现

usr/include/bits/stdio2.h文件有如下实现：

#define __fortify_function __extern_always_inline __attribute_artificial__

```
182 #  ifdef __va_arg_pack
183 __fortify_function int
184 __NTH (asprintf (char **__restrict __ptr, const char *__restrict __fmt, ...))
185 {
186   return __asprintf_chk (__ptr, __USE_FORTIFY_LEVEL - 1, __fmt,
187                          __va_arg_pack ());
188 }
189
190 __fortify_function int
191 __NTH (__asprintf (char **__restrict __ptr, const char *__restrict __fmt,
192                    ...))
193 {
194   return __asprintf_chk (__ptr, __USE_FORTIFY_LEVEL - 1, __fmt,
195                          __va_arg_pack ());
196 }
197
198 __fortify_function int
199 __NTH (obstack_printf (struct obstack *__restrict __obstack,
200                        const char *__restrict __fmt, ...))
201 {
202   return __obstack_printf_chk (__obstack, __USE_FORTIFY_LEVEL - 1, __fmt,
203                                __va_arg_pack ());
204 }
205 #  elif !defined __cplusplus
206 #   define asprintf(ptr, ...) \
207   __asprintf_chk (ptr, __USE_FORTIFY_LEVEL - 1, __VA_ARGS__)
208 #   define __asprintf(ptr, ...) \
209   __asprintf_chk (ptr, __USE_FORTIFY_LEVEL - 1, __VA_ARGS__)
210 #   define obstack_printf(obstack, ...) \
211   __obstack_printf_chk (obstack, __USE_FORTIFY_LEVEL - 1, __VA_ARGS__)
212 #  endif
213
214 __fortify_function int
215 __NTH (vasprintf (char **__restrict __ptr, const char *__restrict __fmt,
216                   __gnuc_va_list __ap))
217 {
```

软件包内有asprintf的函数实现：

```
[z00509839@A191240619 libiberty]$ grep asprintf * -R
asprintf.c:@deftypefn Extension int asprintf (char **@var{resptr}, const char *@var{format}, ...)
asprintf.c:asprintf (char **buf, const char *fmt, ...)
asprintf.c:  status = vasprintf (buf, fmt, ap);
ChangeLog:      strtol.c, vasprintf.c, vprintf.c, vsnprintf.c, xmemdup.c:
ChangeLog:      COPYING.LIB, Makefile.in, _doprnt.c, argv.c, asprintf.c,
ChangeLog:      strtod.c, ternary.c, unlink-if-ordinary.c, vasprintf.c,
ChangeLog:      * asprintf.c: Include config.h.
ChangeLog:      * configure.ac:  Do check declarations for basename, ffs, asprintf
ChangeLog:      and vasprintf for real
```

修改源码包，如果系统有定义asprintf的宏，则不自定义次函数：

```
18  + +--- gdb-7.6/include/libiberty.h.orig
19  + ++++ gdb-7.6/include/libiberty.h
20  + +@@ -612,9 +612,10 @@ extern int pwait (int, int *, int);
21  + + #if !HAVE_DECL_ASPRINTF
22  + + /* Like sprintf but provides a pointer to malloc'd storage, which must
23  + +    be freed by the caller.  */
24  + +-
25  + ++#ifndef asprintf
26  + + extern int asprintf (char **, const char *, ...) ATTRIBUTE_PRINTF_2;
27  + + #endif
28  + ++#endif
```

软件包构建问题—编译器特性差异

OpenEuler
Compiler SIG

问题分类	详细说明	措施
单元测试运行错误	软件包自带单元测试用例运行失败	具体情况具体分析

- **外围包名称：** satyr
- **报错现象:** 测试用例core dump
- **问题根因:**

clang与gcc处理attribute属性不一致

- **措施:**

（1）修改源码，处理编译器是clang的情况；

（2）clang兼容optimize((0))

```
[   37s] make[3]: Entering directory '/home/abuild/rpmbuild/BUILD/satyr-0.42/tests'
[   37s] PASS: abrt
[   37s] PASS: cluster
[   37s] PASS: core_frame
[   39s] ../test-driver: line 112: 1266360 Aborted                 (core dumped) "$@" >> "$log_file" 2>&1
[   39s] FAIL: core_stacktrace
[   39s] PASS: core_thread
```

4 tests/dump_core.c

```
@@ -15,7 +15,11 @@
15  15
16  16   static char const *prefix = "/tmp/satyr.core";
17  17
        18 + #if __clang__
        19 + __attribute__((optnone))
        20 + #else
18      21   __attribute__((optimize((0))))
        22 + #endif
```

https://github.com/abrt/satyr/pull/340

问题分析示例—未定义行为引起的运行问题

OpenEuler Compiler SIG

test.c

```
#define MAX_DIRS   1

__attribute__((noinline))
void setV(char **dirs) {
    for (int i = 0; i < MAX_DIRS; i++) {
        dirs[i] = "ok";
    }
}

__attribute__((noinline))
void printV(char **dirs) {
    for (int i = 0; dirs[i]; i++) {
        printf("dirs[%d] %s\n", i, dirs[i]);
    }
}

int main(void) {
    char *new_dirs[MAX_DIRS];
    setV(new_dirs);
    printV(new_dirs);
    return 0;
}
```

- **包名:** dbus
- **现象:** 运行crash
- **根因:** 软件包越界访问Bug
- **措施:** 循环增加是否越界判断

构建命令	打印
gcc -O2 test.c	dirs[0] ok
clang -O2 test.c	dirs[0] ok dirs[1] dirs[2] \

数组越界访问导致输出随机

上游社区已合入:

https://gitlab.freedesktop.org/dbus/dbus/-/merge_requests/453/diffs

bus/dir-watch-inotify.c

```
@@ -131,7 +131,7 @@ _set_watched_dirs_internal (BusContext
131 131     /* Look for directories in both the old and new sets, if
132 132      * we find one, move its data into the new set.
133 133      */
134     -   for (i = 0; new_dirs[i]; i++)
    134 +   for (i = 0; i < MAX_DIRS_TO_WATCH && new_dirs[i]; i++)
135 135       {
136 136         for (j = 0; j < num_wds; j++)
137 137           {
@@ -160,7 +160,7 @@ _set_watched_dirs_internal (BusContext
160 160           }
161 161       }
162 162
163     -   for (i = 0; new_dirs[i]; i++)
    163 +   for (i = 0; i < MAX_DIRS_TO_WATCH && new_dirs[i]; i++)
164 164       {
165 165         if (new_wds[i] == -1)
166 166           {
```

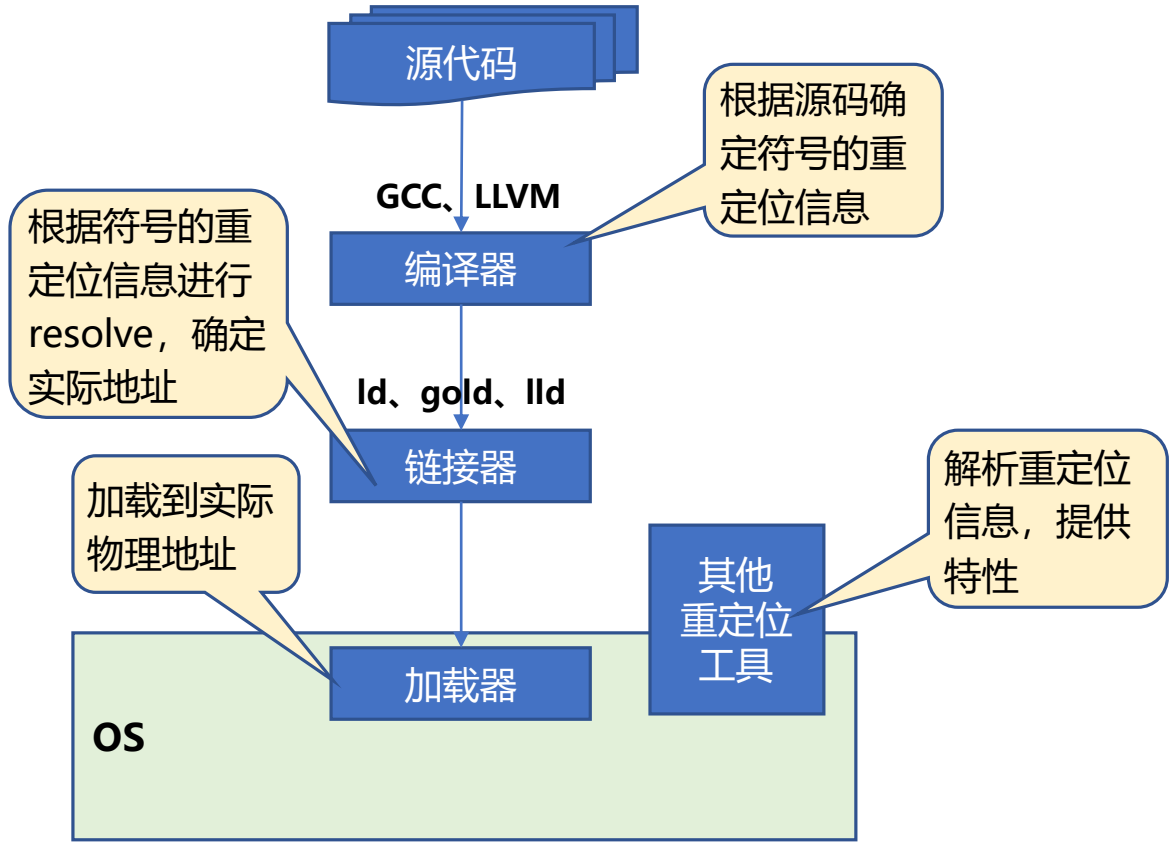
openEuler社区已Backport:

https://gitee.com/src-openeuler/dbus/pulls/79/files

软件包构建问题—重定位兼容性问题

OpenEuler
Compiler SIG

重定位就是把程序的逻辑地址空间变换成内存中的实际物理地址空间的过程。它是实现多道程序在内存中同时运行的基础。重定位有两种，分别是动态重定位与静态重定位。 --百度百科



重定位类型：

- 不同的处理器架构有不同的重定位类型。
  如：X86架构有R_X86_64_32、R_X86_64_PC32等；
  Aarch64架构有：R_Aarch64_ABS64等
- 不同架构的重定位机制&类型是**定义良好**的。

兼容性问题：

- 由于编译器、链接器、工具实现地程度不同，现实中存在兼容性问题。

软件包构建问题—重定位兼容性问题

OpenEuler
Compiler SIG

- **包名:** kexec-tools （其中包含kdump工具）
- **现象:** 程序报错， “ERROR Unknow type: 264”
- **根因:** 264为重定位类型R_AArch64_MOVW_UABS_G0_NC，LLVM生成该定位类型，但kdump不识别该类型。
- **措施:** 方案一、kdump支持对R_AArch64_MOVW_UABS_G0_NC重定位类型；方案二、llvm生成kdump可识别的重定位类型。

```
[root@localhost ~]# systemctl status kdump
× kdump.service - Crash recovery kernel arming
     Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset: enabled)
     Active: failed (Result: exit-code) since Fri 2023-12-29 17:23:22 CST; 1min 17s ago
    Process: 1577 ExecStart=/usr/bin/kdumpctl start (code=exited, status=1/FAILURE)
   Main PID: 1577 (code=exited, status=1/FAILURE)

Dec 29 17:23:21 localhost.localdomain systemd[1]: Starting Crash recovery kernel arming...
Dec 29 17:23:22 localhost.localdomain kdumpctl[1999]: machine_apply_elf_rel: ERROR Unknown type: 264
Dec 29 17:23:22 localhost.localdomain kdumpctl[1580]: kexec: failed to load kdump kernel
Dec 29 17:23:22 localhost.localdomain kdumpctl[1580]: Starting kdump: [FAILED]
Dec 29 17:23:22 localhost.localdomain systemd[1]: kdump.service: Main process exited, code=exited, status=1/FAILURE
Dec 29 17:23:22 localhost.localdomain systemd[1]: kdump.service: Failed with result 'exit-code'.
Dec 29 17:23:22 localhost.localdomain systemd[1]: Failed to start Crash recovery kernel arming.
```

```
1147 void machine_apply_elf_rel(struct mem_ehdr *ehdr, struct mem_sym *UNUSED(sym),
1148     unsigned long r_type, void *ptr, unsigned long address,
1149     unsigned long value)
1150 {
1151 #if !defined(R_AARCH64_ABS64)
1152 # define R_AARCH64_ABS64 257
1153 #endif
```

Kexec-tool 2.0.23

```
1246     case R_AARCH64_LDST64_ABS_LO12_NC:
1247         if (value & 7)
1248             die("%s: ERROR Unaligned value: %lx\n", __func__,
1249                 value);
1250         type = "LDST64_ABS_LO12_NC";
1251         loc32 = ptr;
1252         *loc32 = cpu_to_le32(le32_to_cpu(*loc32)
1253             + ((value & 0xff8) << (10 - 3)));
1254         break;
1255     default:
1256         die("%s: ERROR Unknown type: %lu\n", __func__, r_type);
1257         break;
```

openEuler 24.03 LTS将Kexec-tool升级到2.0.26，**已经支持该重定位类型。**

https://git.sr.ht/~ft/kexec-tools/commit/74c68b80b85b7a1e9cca3ada0952d8f7b9bb3858

目录

OpenEuler
Compiler SIG

LLVM对openEuler核心包的构建情况

OpenEuler
Compiler SIG

openEuler 24.03-LTS 给出了[最小集核心包列表]

目前构建情况：
共226个包，成功构建216个。

libseccomp
util-linux
tzdata
parted
nfs-utils
mpfr
multipath-tools
tpm2-tss
grub2
systemd

核心包名	当前构建情况	备注
kernel	构建成功	
glibc	Aarch64架构构建成功	Glibc代码有较多GNU扩展，需要支持
openjdk	jdk17、21、latest构建成功	Openjdk8代码较老，源码修改后成功
python3	LLVM构建成功	
openssl	LLVM构建成功	

详情请见：https://docs.qq.com/sheet/DUXhYTGxQVVZnV2lq

OpenEuler | Compiler SIG