

Bringing klp-build to LoongArch

Toolchain lessons for the next architecture

What broke when functions moved — and three problems we want your advice on

George Guo · Kylin OS · guodongtai@kylinos.cn

Linux Plumbers Conference 2026 · Live Patching MC

History: kpatch first, then klp-build

When	What
2024-12-05 → 2026-01-12	kpatch LoongArch support: posted, several review rounds with Joe Lawrence, merged
2026-07-24	klp-build LoongArch v4 (14 patches) posted

- kpatch: <https://github.com/dynup/kpatch/pull/1427>
- klp-build v4: <https://lore.kernel.org/all/20260724114128.31451-1-dongtai.guo@linux.dev/>
- The kpatch port already hit two of today's problems: local labels and compiler flags.

The function moves; some dependencies stay

Build: `source patch → original / patched objects → objtool klp diff → .ko`



- `klp diff` selects changed functions and follows their dependencies.
- References to existing unexported symbols need **KLP relocations**.
- Metadata also points **into** code: exception tables, static keys, alternatives.

Three questions for each reference

Question	Failure found on LoongArch	Response
What does it refer to?	Local labels hide the owning function or data	Normalize the target
What value does it express?	Two relocations encode one difference	Fold a supported pair
Can the new code reach the target?	Data stays beyond the PC-relative range	Use GOT indirection

Target identity → expression semantics → address reachability

1 · Local labels hide the owner

Case	What went wrong	Fix
GCC: <code>__ex_table</code> entry → <code>.L1^B4 + 0</code>	label is not the function → entry dropped	text label → containing function + offset
GCC: <code>ANNOTATE_DATA_SPECIAL</code> → <code>.Lentry + 0</code>	offset read from addend only → whole <code>__bug_table</code> cloned	offset = symbol offset + addend
Clang: jump table <code>.LJTI3_0</code> in <code>.rodata</code>	table not cloned; base hit name strings → <code>jr</code> to garbage	data label → section + full offset

Same address, different name: normalize the name, keep the address.

2 · One reference, two relocations

Recorded Clang input: the external static key in `__jump_table`

```
offset 0x18: ADD64    __tracepoint_netif_rx + 8
offset 0x18: SUB64    __jump_table + 0x18

                (S + A) - P
                ↓
offset 0x18: 64_PCREL __tracepoint_netif_rx + 8
```

- Cloning failed: `elf_create_reloc()` accepts **one relocation per offset**.
- Fold only a matching ADD/SUB pair whose SUB target is **the field itself**.

LLVM fix: <https://github.com/llvm/llvm-project/pull/211754> — in main (Clang 23);
Clang 22 still emits the pair.

3 · The target can be out of reach

running kernel

livepatch module

task_state_array[]

proc_pid_status()

original static data ←———— X ————— pcalau12i + addi.d

two-instruction PCALA reach: about ± 2 GiB

module-to-kernel distance: far beyond that range

Reference	Fix
same-unit or external global	compiler flags: <code>-fPIC</code> , no direct external-data access
file-local <code>static</code> data	objtool rewrites it into a GOT load (next slide)

3 · Rewrite the address calculation into a GOT load

```
# Before: compute the address directly
pcalau12i $a1,      %pc_hi20(sym)
addi.d    $a1, $a1, %pc_lo12(sym)

# After: load the full address from a nearby GOT slot
pcalau12i $a1,      %got_pc_hi20(sym)
ld.d      $a1, $a1, %got_pc_lo12(sym)
```

module code → nearby GOT slot → full 64-bit kernel address

Ask 1 · Direct loads: compiler help, thunks, or reject?

PCALA low instruction	What it does	Today
<code>addi.d</code>	Materializes an address	Rewritten to a GOT load
<code>ld.d</code> / store	Reads or writes the value directly	Build fails
<code>addi.d</code> , high separated or shared	Same job, other shape	Build fails

- Through the GOT a direct load needs **one more instruction**; a cloned function cannot grow in place.
- `-mcmmodel=extreme` also turns direct calls indirect and confuses objtool (B8).

Which direction would you maintain? Is there a compiler option we missed?

Ask 2 · Where should normalization live?

Today	What it handles
generic klp-diff	local labels → owner + offset; entry offsets
architecture hooks	ADD/SUB fold; PCALA → GOT rewrite

Is this the right boundary for the next port — or should klp diff carry relocation expressions (such as `s1 - s2`) natively, so each architecture does not write its own fold?

Ask 3 · What should a new port prove?

- A successful module build **missed** several of these failures.
- Proposal: relocation examples per supported toolchain · checks that the right metadata is retained · runtime tests that exercise references and rollback.
- Should these become shared klp-build tests?

Toolchain side question: GAS keeps `.L*` labels for special-section entries even though the kernel builds with `-mno-relax`. Does anyone know why?

Thank you

George Guo · guodongtai@kylinos.cn

Patches: <https://lore.kernel.org/all/20260807061510.26138-1-dongtai.guo@linux.dev/>

Identify the target. Preserve the expression. Check reachability.

Thanks to Kexin Liu and WangYuli for co-development,
and Joe Lawrence, Xi Ruoyao, Huacai Chen, and Josh Poimboeuf
for review, feedback, and the tooling this work builds on.

B1 · Patch map: first seven commits

Commit	Change	Main slide
5cd7ddd4652b	Normalize local-label references in text and data	5
75d8cde06a07	Include symbol offsets in annotation boundaries	5
31925d1eff50	Mask branch displacement bits for checksums	B6
c303c49bdf0a	Fold supported ADD/SUB pairs into PCREL	6
ec437cbee558	Retain tablejump words referring to cloned targets	B3
acc94058e671	Rewrite supported PCALA KLP references to GOT	7, 8, 9
af43b8e71703	Add <code>arch_adjusted_addend()</code> , returning the plain addend	B6

Exact series supplied for this talk; ordered oldest to newest.

B2 · Patch map: remaining seven commits

Commit	Change	Main slide / backup
e2b2dfa882b6	Enable inline alternative cloning	B4
4b11096b16e4	Use <code>-fPIC</code> ; disable direct external-data access	7, B5
4ed7542260e1	Add the LoongArch syscall patching macro	Build prerequisite
890d3daf1a3b	Mark special-section entries and alternative blocks	5, B4
7977c090e948	Collect EFI <code>.init.bss.*</code> under <code>-fdata-sections</code>	Build prerequisite
460896b12bff	Select <code>HAVE_KLP_BUILD</code> ; enable objtool KLP build	Build prerequisite
5f7146699384	Select patchable function entries for ftrace	Build prerequisite

14 commits · 15 files · 473 insertions · 14 deletions

B3 · Tablejump: preserve the table-side words

Input / fixed output	Broken output
0x00 proc_pid_status + 0xc70	0x00 proc_pid_status + 0xc70
0x08 .rodata.proc_pid_status	0x08 proc_pid_status + 0xd2c
0x10 proc_pid_status + 0xd2c	
0x18 .rodata.proc_pid_status+0x80	

- Fake symbols remain **8-byte words**; a logical entry remains **16 bytes**.
- The previous rule required a cloned **function** target.
- This patch also accepts cloned **section** targets for this annotation section.
- It changes `should_keep_special_sym()`, not the annotation reader's iteration.

B4 · Mark entry boundaries and replacement code

Data to preserve	Kernel-side description
<code>.altinstructions</code> , <code>__ex_table</code>	Read-only "aM" sections with explicit entry size
<code>__bug_table</code> , <code>__jump_table</code>	Writable "aw" sections + <code>ANNOTATE_DATA_SPECIAL</code>
Inline alternative replacement	<code>ANNOTATE_DATA_SPECIAL</code> / <code>_END</code> around the block

`ARCH_HAS_INLINE_ALTS` enables the existing replacement-cloning path.

- A replacement is cloned as a separate fake symbol, with its relocations.
- Recorded checks verify **nop replacement** layout for new and changed functions.
- Relocation-bearing replacements and `ALTERNATIVE_2` : **still pending in the notes.**

B5 · Exact compiler options and configuration

```
KCFLAGS += -fPIC
```

```
KBUILD_CFLAGS_KERNEL = -mno-direct-extern-access      (GCC)  
KBUILD_CFLAGS_KERNEL = -fno-direct-access-external-data (Clang)
```

- The command-line assignment **replaces** the architecture value, removing `-fPIE`.
- klp-build also uses `-ffunction-sections -fdata-sections` for its rebuilds.
- LoongArch validation requires `CONFIG_RELOCATABLE=y` and `CONFIG_AS_HAS_EXPLICIT_RELOCS=y`.
- Inspect emitted relocations: a flag name alone does not prove GOT coverage.

B6 · Branch references also affect change detection

`arch_jump_opcode_bytes()` masks embedded displacement bits:

Instruction format	Retained bits
B / BL	Opcode: <code>0xfc000000</code>
BEQZ / BNEZ / BCEQZ / BCNEZ	Opcode + register: <code>0xfc0003e0</code>
JIRL / two-register conditional branches	Opcode + registers: <code>0xfc0003ff</code>

- Generic checksum code also hashes a representation of the destination.
- This handles decoded branch/call targets **without an instruction relocation**.
- Separate hook: `arch_adjusted_addend()` returns the LoongArch relocation addend.

B7 · Where a KLP relocation goes

```
klp diff:      .klp.sym.<object>.<symbol>,<sympos>
               __klp_relocs entry: object, symbol, sympos, addend, type
post-link:     .klp.rela.<object>.<section>
module load:   resolve vmlinux targets; apply architecture relocations
target module: apply its deferred relocations when it is available
```

- Existing unexported targets use the KLP symbol-resolution path.
- In the current code, **vmlinux exports** can use ordinary module relocations.
- Post-link turns the ordinary reloc into `R_LARCH_NONE`; the real type is in `.klp.rela.*`.
- GOT capacity is reserved from relocation types before addresses are applied.

B8 · Why the extreme code model was not adopted

Recorded A/B build of `drivers/video/backlight/backlight.o` (Clang 21):

Observation	Baseline	With <code>-mcmodel=extreme</code>
<code>R_LARCH_B26</code> relocation count	88	0
objtool warnings	None	Five "unexpected end of section" warnings

- Wider address materialization also changed call sequences in this experiment.
- Losing direct callee information interfered with objtool's analysis.
- The series keeps the existing call path and converts supported data references.