Static linking symbol conflicts

PR	Description	Notes	JDK 11 submission date
https://github.com/openjdk/jdk/commit/5c1d7cd99551661d772c0289fb2bd8ca5e6cbc05	jianglizhou: Redefine 'BaseThread' to 'HotspotBaseThread' in hotspot	JDK-8311846	2022/01/18-12:1
https://github.com/openjdk/jdk/commit/0fc04423e7d86553bdae39cad8618d13dea919ce	rasbold: Fix Thread symbols in _ZN10JavaThread25aarch64_get_thread_helperEv		2022/01/20-16:1

Statically linked built-in native library support

PR	Description	Notes	JDK 11 submissi
https://github.com/openjdk/jdk/commit/294dca0eef57bb0f9d1748e4a8c316c88e718df2	jianglizhou: Resolve some crashes and failures during startup.	I changed to not use weak symbols in the PR for portability consideration.	2021/11/01-17:0
https://github.com/jianglizhou/jdk/commit/1a045726cced9d00a11743aa205093d1724327b0	jianglizhou: Fix crash with jtreg tests on dynamic JDK		
https://github.com/jianglizhou/jdk/commit/b32b30b1d88856832c91cba5e647199cb1be3041 https://github.com/jianglizhou/jdk/commit/e45f8f175428b81295ba2e6226204b20a7295100	jianglizhou: Rename set_static_jdk and is_static_jdk		
https://github.com/openjdk/jdk/commit/d6aa234e993b4c6c4f69533a71929b4e87c8cc5f	jianglizhou: In findJniFunction(), only try to trim JNI_LIB_PREFIX and JNI_LIB_SUFFIX from 'cname' if we find any path components. Changed os::find_agent_function() to always look up Agent_On(Un)Load/Attach<_lib_name> first.		2021/11/01-16:2
https://github.com/openjdk/jdk/commit/63354ca43bf90ebb15a5fe3a434a0123c75b7b05	jianglizhou: Clean-up the changes for using unique JNI_OnLoad|JNI_OnUnload|Agent_OnLoad|Agent_OnUnload|Agent_OnAttach symbols in different JDK JNI libraries by default.		2021/11/09-08:4
https://github.com/openjdk/jdk/commit/f2ac77cbbe59821284f51a6dcc2563b4bb9c9311 https://github.com/openjdk/jdk/commit/9e3c81bc3a80117ca99867fa47f64aee10387aa4	jianglizhou: Use runtime check for static JDK. Remove STATIC_BUILD usage in awt_LoadLibrary.c; Add JNIEXPORT to JLI_IsStaticJDK; Fix errornous dlsym call for isStaticJDK check in AWT_OnLoad.	In JDK head, dlopen() is under #ifndef STATIC_BUILD.	2021/12/03-16:2
https://github.com/jianglizhou/jdk/commit/31f8e721e1fe763a9bddb44a7bd731f93223a6b6	jianglizhou: Change libjdwp and libawt natives to use JVM_IsStaticJDK instead of \`JLI_IsStaticJDK\` to check for static JDK at runtime.		
https://github.com/openjdk/jdk/commit/f2ac77cbbe59821284f51a6dcc2563b4bb9c931	jianglizhou: Use runtime check for static JDK.		2022/01/11-10:2
https://github.com/openjdk/jdk/commit/ad90961e5bdf43aef815a914fa150d451e5bc088	jianglizhou: Remove STATIC_BUILD usages. Support static linking and dynamic linking libjdwp (and libdt_socket) with same .o files.	Will discuss the usages of STATIC_BUILD .	2022/09/06-11:0

https://github.com/jianglizhou/jdk/commit/e1342c04e6415b9f11e13ecec72e77fad871bae0	jianglizhou: Don't report any error and bail out too early in lookup_JVM_OnLoad_entry_point if it does not succeed, since we want to try lookup_Agent_OnLoad_entry_point for Agent_OnLoad as well. With static linking support for built-in library, if we cannot find the JVM_OnLoad_<libName> symbol and determine that the library is built-in, we also try loading the shared library. However, we don't want to report error if the requested shared library cannot be loaded. Instead we let lookup_Agent_OnLoad_entry_point to report any error if there is any failure.		2022/10/17-10:5
Remove STATIC_BUILD macro			
PR	Description	Notes	JDK 11 submissi
https://github.com/openjdk/jdk/commit/c635ac1006a29afc03d5e67dcfc708af23e9ddea https://github.com/openjdk/jdk/commit/105774e6f28169c46a7f58c84152351d654d6085	jianglizhou: Define the JNI_OnLoad entries for libjimage to make sure the builtin library work properly with the static builds; Define DEF_STATIC_JNI_OnLoad for 'zip' library by default. Remove the usage of STATIC_BUILD macro.		2021/11/29-20:3
Hermetic deploy JAR packaged runtime			
PR	Description	Notes	JDK 11 submissi
https://github.com/openjdk/jdk/commit/8b5526c4ba8a852d67453d199634c80f5f3b2da7	jianglizhou: JDK support for hermetic JAR packaged jimage (aka runtime image). This handles the JDK runtime image that's packaged within the hermetic JAR at a specific file offset (page aligned). In hermetic JAR, the <JDK>/lib/'modules' data is packed within the JVM data section, which is between the ELF section and the JAR section in a deploy JAR. The start offset of the 'modules' data is page aligned. Special launcher argument that can be passed to the VM: -XX:UseHermeticJDK=<deploy_jar_path>,<jimage_start_offset> The jimage start offset is used when opening/mapping and reading the jimage.		2022/03/29-11:1
https://github.com/openjdk/jdk/commit/c13287f669b77177b2e5ee83b976b2517abc070c	jianglizhou: Map hermetic packaged modules using the modules size correctly. With the hermetic Java support, we embed the JDK 'modules' image in the hermetic JAR at page aligned offset. The offset is recorded in the deploy JAR manifest 'JDK-Lib-Modules-Offset' attribute. At runtime, the recorded offset information is used by the VM to mmap the modules image. Also, record the 'modules' file size in deploy JAR manifest 'JDK-Lib-Modules-Size' attribute. The VM option, -XX:UseHermeticJDK is extended to include the image size: -XX:UseHermeticJDK:<executable_image_file_path>,<jdk_runtime_image_start_offset>,<jimage_size> At runtime, modules size from the manifest attribute may be retrieved and passed to the VM via -XX:UseHermeticJDK option. Both the hermetic 'modules' image offset and size are used when mmap'ing the 'modules' data.	https://github.com/openjdk/jdk/commit/c13287f669b77177b2e5ee83b976b2517abc070c	2022/10/18-14:3
https://github.com/jianglizhou/jdk/commit/9772972839926dd816a6e78776741f6b5cb46099.patch	jianglizhou: Fix linux-x86 build failure caused by error 'cannot convert 'size_t*' {aka 'unsigned int*'} to 'julong*' {aka 'long long unsigned int*'}'.		
https://github.com/jianglizhou/jdk/commit/325ba6661515e3a3ec9fd8a066b99dac3f8f20aa	jianglizhou: Handle the case for hermetic JAR embedded modules in skip_first_path_entry(), when checking the shared classpath. In a hermetic JAR, the 'modules' image is part of the JAR and there is no separate modules file. In that case, we set up the JAR instead of the modules file as the first path entry. So skip_first_path_entry() needs to take that into consideration. The other place that checks for MODULES_IMAGE_NAME (modules) is in ClassLoader::setup_boot_search_path(), which is already fixed by openjdk@8b5526c.	I haven't seen the assertion with JDK@head testing when running HelloWold using hermetic deploy JAR yet. It's okay to port this first. Ported for JDK@head: https://github.com/jianglizhou/jdk/commit/325ba6661515e3a3ec9fd8a066b99dac3f8f20aa	2022/04/12-11:1
https://github.com/jianglizhou/jdk/commit/1b113437022a93f8697f1b05792316424b416a06	jianglizhou: Initial support for accessing hermetic JAR packaged JDK resource files via java.home in JDK.		

https://github.com/jianglizhou/jdk/commit/d4f10355f7b4f6b2e1ee367effaabf558630e056	jianglizhou: Support hermetic packaged lib/security/cacerts in sun.security.ssl.TrustStoreManager$TrustStoreDescriptor and sun.security.util.AnchorCertificates. Use jdk.internal.misc.JavaHome to access JDK default store, lib/security/cacerts in TrustStoreDescriptor. If 'javax.net.ssl.trustStore' property is set and the value is not 'NONE', the specified store is accessed as a regular file using Path.of() API. The original semantics should not be affected by the change.		2023/03/15, 8:18
https://github.com/jianglizhou/jdk/commit/a00913671b747ddb108d461bda9644177462568b	jianglizhou: Support hermetic JAR packaged conf/security/policy/{limited|unlimited} cryptography extension policy files.	Ported to JDK@head: https://github.com/jianglizhou/jdk/commit/a00913671b747ddb108d461bda9644177462568b	2022/06/01-10:5
https://github.com/jianglizhou/jdk/commit/bfdbe012a2c50d56bcb62be3c2b3e1902228f482	jianglizhou: Support hermetic JAR packaged lib/ct.sym file. Removed the use of the static final 'symbolFileLocation' field, which defines the path subcomponents of the 'lib/ct.sym' file. Instead, the path elements are passed as the arguments of JavaHome.getJDKResource(). That provides more complete and precise logging information for verifying the runtime access of ct.sym file path.	**This change requires JavaHome class in the boot JDK. Commented out and reverted in the github branch. Reverted.**	2022/06/21-14:3
https://github.com/jianglizhou/jdk/commit/8e8c8efff32c92eb40c99eb270bfbc3fdc8626cd	jianglizhou: Support runtime accessing for hermetic JAR packaged JDK-bundled fonts (<jdk>/lib/fonts/*.ttf) and <jdk>/lib/fontconfig.<os>.properties.	Ported to JDK@head: https://github.com/jianglizhou/jdk/commit/8e8c8efff32c92eb40c99eb270bfbc3fdc8626cd	2022/07/21-10:1
https://github.com/jianglizhou/jdk/commit/a4de3f83b33145a46878b37dd25479506534c75b	jianglizhou: Support POSIX_SPAWN launch mechanism for ProcessBuilder.start() on hermetic Java. Moved the code from jspawnhelper.c to childproc.c and renamed the original main() (in jspawnhelper.c) to JDK_spawn_process() in childproc.c. The jspawnhelper.c main() is now a simple wrapper of JDK_spawn_process(). JDK_spawn_process() can be shared by jspawnhelper and launcher for creating child process using POSIX_SPAWN launch mechanism.	In JDK@head spawnChild(), the hlpargs[] takes an additional argument as argv[0], which is the path to jspawnhelper. The change was done for https://bugs.openjdk.org/browse/JDK-8310265.	2022/09/28-11:5
https://github.com/jianglizhou/jdk/commit/c5237edb54ca78b8c4b3af68316961d0c3814a52	jianglizhou: Use JavaHome.EXECUTABLE field for storing the hermetic executable name/path. The hermetic executable is the hermetic JAR (as the JAVA_HOME) by default. Launcher may set jdk.internal.misc.hermetic.executable property value, which is retrieved and stored in JavaHome.EXECUTABLE.		Jun 2, 2023, 12:

https://github.com/jianglizhou/jdk/commit/fcd0db731db38e85314d4efbbba5792fcd0d8a9d	jianglizhou: Delay the initialization of 'jarFileSystem' field in JavaHome.
	The earlier a4de3f8 change in ProcessImpl.java causes JavaHome class initialization occur early before the module system initialization. That in turn causes the loading of the "Jar" provider happen before the module system initialization. When running on a hermetic Jar, the system fails to start due to "java.nio.file.ProviderNotFoundException: Provider "jar" not found".
	
	This change delays the initialization of JavaHome.jarFileSystem. It's no longer initialized during JavaHome <clinit>. The initialization of JavaHome.jarFileSystem now happens when the system first tries to access a hermetic JAR packaged JDK resource/property file.
		Fix issue:
		
		`Error occurred during initialization of boot layer`
		`java.lang.ExceptionInInitializerError`
		`Caused by: java.nio.file.ProviderNotFoundException: Provider "jar" not found`
		
		I think it's worth discussing upstream is if jdk.nio.zipfs.ZipFileSystemProvider (which is the provider that fails to load if loading occurs before the module system is initialized) could be moved to the module boot layer and be loaded by the null class loader. We can bring this up as part of the hermetic discussion.	

https://github.com/jianglizhou/jdk/commit/510c760d1ba86590ea5b72dbe4019c5d1bae0683	jianglizhou: Check isHermetic for isHermetic() as jarFileSystem may not initialized yet when `isHermetic()` is called. This bug was causing java.nio.file.FileSystemException when loading conf/security/java.security.		

https://github.com/jianglizhou/jdk/commit/8d1881461f468e77d415c06bf2e3b1d1d2ffe151	jianglizhou: In j.u.ServerLoader, use the platform classloader after the module system is initialized but before the VM initialization is completed. Don't use the system classloader to find resources if the loader is null when finding service provider. There are two issues uncovered when FileSystemProvider.loadInstalledProviders() is called early during system startup before the VM is initialized. The VM is considered in booted state after System.initPhase3() completes. 1) In nextProviderClass(), when the loader is null, ClassLoader.getSystemResources() is called to find the service. The result can include service providers that can only be loaded by the system class loader, e.g. in JAR files on the -classpath. That can cause failure when the null classloader is trying to load the provider class. This issue is addressed by changing to call 'BootLoader.findResources(fullName)' instead, if the 'loader' is null. 2) When trying to load installed FileSystemProvider during early start up before the VM is booted, it fails to find the 'jar' provider. That's because the boot loader (a.k.a. the null classloader) is used by ServiceLoader, which tries to only use the code in java.base at the time. The ZipFileSystemProvider and JarFileSystemProvider are in jdk.zipfs module. The JavaHome is trying to use the JarFileSystem during initPhase3, which is after system module initialization. During that phase, it can use the platform classloader to load the installed provider, which would able to find the ZipFileSystemProvider and JarFileSystemProvider. These issues are found by hermetic Java testing, however I think these are not specific to hermetic Java.		Jun 8, 2023, 11:

Makefile changes for static libs and static linking

PR	Description	Notes	JDK 11 submissi
https://github.com/openjdk/jdk/commit/10c2a5eae6d8245ea484364963e8a2d10c949fd3 https://github.com/openjdk/jdk/commit/e123b4f52c8abbb6b9d4e9f45e5ee99a194ae49c	jianglizhou: Makefile changes from #13709 to demonstrate fully statically linked JDK build, i.e. building java launcher executable statically linked with JDK and hotspot native code. Don't link with the extra libs for awt headfull and jsound for now.	To build static JDK: - bash configure --with-boot-jdk=<jdk_path> --with-static-java=yes - make static-java-image	

Misc

PR	Description	Notes	JDK 11 submissi
https://github.com/jianglizhou/jdk/commit/34027e5d16ed89291d9b1fc683406f95debd05ee	jianglizhou: Hermetic Java related logging can be enabled by -Xlog:hermetic (or -Xlog:hermetic=info).		2022/09/06-13:2
https://github.com/jianglizhou/jdk/commit/71ca42c919947cf1d858af8fc13e9e7faa991af8	jianglizhou: Add DisableHermetic flag for disabling hermetic Java runtime. If a hermetic JAR is used, setting DisableHermetic to true disables the hermetic Java runtime (the JDK modules image and JDK resources) packaged in JAR. The flag has no effect, if the JAR is non-hermetic. It does not affect the JDK static support either.		Feb 8, 2023, 10:

https://github.com/jianglizhou/jdk/commit/dcd0659b735ce7d2135d06bcd249cc67fd928def	jianglizhou: Ignore -server|-client options in Arguments::parse_each_vm_init_arg when executing in hermetic Java mode. As part of the CreateExecutionEnvironment operations, CheckJvmType checks for VM types (known types specified in lib/jvm.cfg) for normal non-hermetic Java execution mode. A side effect of CheckJvmType is the removal of -server|-client options if they exist in the command-line options. Arguments::parse_each_vm_init_arg would report 'Unrecognized option' error when encounters these options. With hermetic Java/JDK static linking, CreateExecutionEnvironment operations are skipped as they are not necessary. As a result, Arguments::parse_each_vm_init_arg may see -server|-client if the options exist in the command line when running in hermetic Java mode. This fix changes Arguments::parse_each_vm_init to not report error for -server|-client options.		Thu, Jan 12, 202
https://github.com/jianglizhou/jdk/commit/0c95ee8f4f4848ee6306cc9564549995123dbee2	jianglizhou: Remove runtime archived heap oopmap check (non-product only code). The code was only enabled for non-product binary in ArchiveHeapLoader::patch_embedded_pointers. ArchiveHeapLoader::patch_embedded_pointers is called during MetaspaceShared::initialize_shared_spaces to patch all archived Java heap pointers when runtime relocation occurs (e.g. archived Java heap regions cannot be mmap at the desired addresses due to runtime Java heap size difference). That's done early during VM initialization and before SystemDictionary::resolve_well_known_classes. The calculate_oopmap operations may access some of the well-known klasses during oop iteration. That could cause crashes since the well-known klasses are not loaded/resolved at the time. When loading/resolving a shared well-known klass, SystemDictionary::resolve_wk_klass loads and restores the archived klass and mirror object. So it's not feasible to move HeapShared::patch_archived_heap_embedded_pointers to a later point after resolving some of the needed well-known classes during VM initialization. Hence removing the runtime sanity check.		Wed, Jan 18, 20