Building OpenMPI with a custom Glibc

The following has been tested on a Centos 7.4 system with GCC 8.1.0. Glibc 2.27 is compiled and installed in a custom directory `$GLIBC_DIR` as follows:

```
$ tar xaf glibc-2.27.tar.bz2
$ mkdir glibc-2.27/build
$ cd glibc-2.27/build
$ ../configure --prefix=$GLIBC_DIR
$ make -j 32
$ make install
```

By default the custom dynamic loader will look for libraries in `$GLIBC_DIR/lib`, and - if present - it will use `$GLIBC_DIR/etc/ld.so.cache`. Since we want the custom Glibc to have access to the usual system libraries, we need to update the cache:

```
$ echo /lib64 > $GLIBC_DIR/etc/ld.so.conf
$ echo /usr/lib64 >> $GLIBC_DIR/etc/ld.so.conf
$ $GLIBC_DIR/sbin/ldconfig
```

In fact, it a good idea is to copy the entire contents of the system `/etc/ld.so.conf` and `/etc/ld.so.conf.d` to `$GLIBC_DIR/etc`.

Compiling programs against a custom Glibc requires using some non-standard GCC compilation flags. One needs to tell GCC to ignore the native Glibc, which is done with the `-nostdinc` switch. Also, the custom Glibc include path needs to be specified, together with internal GCC paths:

```
$ gcc -O3 -g -nostdinc -I$GLIBC_DIR/include
   -I$GCC_DIR/include
   -I$GCC_DIR/lib/gcc/x86_64-pc-linux-gnu/$GCC_VER/include
   -I$GCC_DIR/lib/gcc/x86_64-pc-linux-gnu/$GCC_VER/include-fixed
   -c test.c
```

The latter can be obtained on the command line, e.g.:

```
$ gcc -xc -v -
[...]
#include <...> search starts here:
 <GCC_DIR>/lib/gcc/x86_64-pc-linux-gnu/<GCC_VER>/include
 /usr/local/include
 <GCC_DIR>/include
 <GCC_DIR>/lib/gcc/x86_64-pc-linux-gnu/<GCC_VER>/include-fixed
 /usr/include
End of search list.
```

When linking programs against a custom Glibc GCC needs to be told to ignore the standard libraries, use correct Glibc CRT startup / finish functions, and to use the custom dynamic loader. Also, specyfying `RUNPATH` that points to the custom Glibc is useful. For example, when linking a dynamic ELF executable:

```
BEGFILES="$GLIBC_DIR/lib/crt1.o $GLIBC_DIR/lib/crti.o \
            $(gcc --print-file-name=crtbegin.o)"
ENDFILES="$GLIBC_DIR/lib/crtn.o $(gcc --print-file-name=crtend.o)"
LDFLAGS="-nostdlib -L$GLIBC_DIR/lib $GLIBC_DIR/lib/libc.so \
         -Wl,--start-group \
             -lgfortran -lgcc_s -lgcc -lgcc_eh -lstdc++ \
         -Wl,--end-group \
         -Wl,--rpath=$GLIBC_DIR/lib \
         -Wl,--rpath=$GCC_DIR/lib64 \
         -Wl,--enable-new-dtags \
         -Wl,--dynamic-linker=$GLIBC_DIR/lib/ld-linux-x86-64.so.2"

$ gcc $BEGFILES $ENDFILES $LDFAGS test.o -o test
```

Since different start / end files are used when linking shared objects (the `-shared` flag), having a general-purpose compilation environment requires writing a simple wrapper for GCC, G++, and GFORTRAN. Here we call those wrappers `gccwrap, g++wrap, gfortranwrap`, respectively. A draft of the wrapper, with details left out for brevity, is as follows:

```
#!/bin/bash

ARGS="$@"

# gcc information
GCC_DIR=$(dirname $(dirname `which gcc`))
GCC_VER=$(gcc --version | grep ^gcc | sed 's/^.* //g')

# what are we wrapping?
prog=$(echo $0 | sed -e 's/wrap//')
prog=$(basename $prog)

# analyze the arguments, check if we are compiling, linking, or other
[...]
if test $compile == 0; then
   # not compiling - do not wrap
   $prog $ARGS
   exit $?
fi

# update CFLAGS with -nostdinc etc.
CFLAGS=[...]

# linking - different setup for shared and runnable objects
if test $link != 0; then
    if test $shared != 0; then
       # shared library start / end files
       BEGFILES=[...]
       ENDFILES=[...]
       LDFLAGS=[...]
    else
       # runnable start / end files
       BEGFILES=[...]
       ENDFILES=[...]
       LDFLAGS=[...]
    fi
fi

# fix double quotation marks around arguments, e.g., -DVAR="value"
ARGS=$(echo $ARGS | sed -e 's/"\([^"]*\)"/\\\""\1"\\\"/'g)

# execute the wrapped program with modified environment
eval $prog $CFLAGS $ARGS $LDFLAGS
```

Using the wrapper we can now compile and install OpenMPI, which uses the custom Glibc. On a Mellanox Infiniband system with HPCX libraries this can be done as follows:

```
$ tar xaf openmpi-3.1.0.tar.bz2
$ cd openmpi-3.1.0
$ ./configure CC=gccwrap CXX=g++wrap FC=gfortranwrap \
--prefix=$OMPI_DIR \
--with-knem=${HPCX_HOME}/knem \
--with-mxm=${HPCX_MXM_DIR} \
--with-hcoll=${HPCX_HCOLL_DIR} \
--with-ucx=${HPCX_UCX_DIR} \
--with-platform=contrib/platform/mellanox/optimized \
$ make -j 32
$ make install
```

For `configure` to succeed a number of external development packages are needed - those can be simply installed from native OS package manager. One problem on newer ConnectX-4 systems is that the custom OpenMPI runtime cannot find one library:

```
$ mpicc test.c -o test
$ mpirun -np 2 ./test
libibverbs: Warning: couldn't load driver 'mlx5':
  libmlx5-rdmav2.so: cannot open shared object file:
  No such file or directory
[...]
```

The file is present in `/usr/lib64`, and is a link to `libmlx5.so.1.0.0`. For some reason it is not picked up by the custom dynamic loader and has to be manulaly copied into `$GLIBC_DIR/lib`. The installation is then complete.

OpenMPI compiled in the above way will use the wrapper inside `mpicc` to compile user code, hence nothing needs to be changed on the user-side to be able to use the custom Glibc.