Fix Multi Thread debug fix for AIX

The bug:- In the recent commit 98ed24fb35d89eb20179edf6c12f599c7a9e228e there is a change in aix-thread.c file that changes

static_cast<aix_thread_info *> in gdb to
gdb::checked_static_cast<aix_thread_info *>

AIX folks using the latest version thus will not be able to debug multi thread programs as a result of it. The error in AIX is as follows: - internal error: checked_static_cast: Assertion `result! = nullptr' failed.

The root cause of the issue:- The private data was not set for the first thread or the main thread of a process. In AIX when we run an “info threads” command, we showed main process as “process <pid>” without private data set and added a new thread Thread<Tid> representing the same with private set. When we iterate_over_threads () we call get_aix_thread_info (). This leads to the crash as we had the main process thread “process <pid>” with no private data. Hence the checked static cast will not allow us to debug any further which is rightly so as we had a thread with no private data.

What should be the fix: - Removing the main process thread i.e. “process <pid> “was the first proposed solution as the “Thread <tid>” representing the same already exists with private data set. This was happening in the sync_threadlists () code of AIX.

Solution Part 1: -

Why the change?

The delete_thread () with the cmp_result > 0 block of the for loop in the sync_threadlists () function which applies the difference between the pthread and GDB threadlist, **will fail to delete the main process thread**. The reason is that it “process <pid>” is the current process and thus GDB core will not delete it despite we are calling it. Hence even if we add the “thread <tid>” representing the same “process <pid>” in the next iteration of the for loop we will not be successful.

Hence this forces us to change the main process thread “process <pid>” to “thread <tid>” via the thread_change_ptid () and the private data set. These changes can be seen in the sync_threadlists () part.

However, we also need to keep in mind that before we think this will work, our libpthread library is only ready when the following condition in the wait () of aix-thread.c is satisfied.

```
/* Check whether libpthdebug might be ready to be initialized.  */
  if (!data->pd_active && status->kind () == TARGET_WAITKIND_STOPPED
      && status->sig () == GDB_SIGNAL_TRAP)
```

Until then changing the “process <pid>” to “thread <tid>” is incorrect. Even though the session is ready and initalised via pd_enable () and pd_activate () functions respectively. Therfore this made us to keep a variable **pthdebugready** in all functions that lead to sync_threadlists () so that we change the process thread to a thread with private data only when libpthdebug is initialised for a particular process.

The first if condition below this paragraph change in the sync_threadlists () as shown below means the pthread debug library is not intialised. This is just to set priv to main process thread.

```
if (gbuf[0]->ptid.is_pid () && !pthdebugready)
   {
     aix_thread_info *priv = new aix_thread_info;
     tp->priv.reset (priv);
   }
```

The second if condition below this paragraph change is for changing “process <pid>” to “thread <tid>” as the pthread debug library is intialised.

```
if (gptid.is_pid () && pthdebugready)
            {
              thread_change_ptid (proc_target, gptid, pptid);
              aix_thread_info *priv = new aix_thread_info;
              priv->pdtid = pbuf[pi].pdtid;
              priv->tid = pbuf[pi].tid;
              tp->priv.reset (priv);
              gi++;
              pi++;
            }
```

Failing to do so leads us to two problems. One while we fetch_registers () our regcache-> ptid though changed to ptid_t (pid, 0, tid) will not be able to get the private data in a case where we switch to a child process from the parent process via “inferior 2” command leading to the crash that private data was not set for a thread. Because we incorrectly changed the “process <pid>” to “thread <tid>” before the process itself could raise a trap and tell the debugger we are now ready to debug threads.

Example of the crash:-

(gdb) set detach-on-fork off
(gdb) r
Starting program:
[New Thread 258]
[New Thread 515]
[New inferior 2 (process 21627386)]
I am parent
[New inferior 3 (process 9372064)]
I am parent
^C
Thread 1.1 received signal SIGINT, Interrupt.
[Switching to Thread 1]
0xd0595fb0 in _p_nsleep () from /usr/lib/libpthread.a(shr_xpg5.o)
(gdb) inferior 2
[Switching to inferior 2 [process 21627386] (/home /gdb_tests/ultimate-multi-thread-fork)]
[Switching to thread 2.1 (Thread 515)]
#0 0xd0594fc8 in _sigsetmask () from /usr/lib/libpthread.a(shr_xpg5.o)
(gdb) c
Continuing.
./../gdbsupport/gdb-checked-static-cast.h:58: internal-error: checked_static_cast: Assertion `result != nullptr' failed.

The process stack of the crash due to the is as below: -

0x000000010059ef60 aix_thread_info* gdb::checked_static_cast<aix_thread_info*, private_thread_info>(private_thread_info*)(0x0) + 0x7c
0x0000000100596ea0 get_aix_thread_info(thread_info*)(0x0) + 0x34
0x000000010059b778 aix_thread_target::fetch_registers(regcache*, int)(0x11001f3f8, 0x1107c5030, 0x4000000000) + 0xf8
0x00000001003675f0 target_fetch_registers(regcache*, int)(0x1107c5030, 0x40e0ddf00d) + 0x6c
0x00000001005817c0 regcache::raw_update(int)(0x1107c5030, 0x401001f3f8) + 0x94
0x0000000100581904 readable_regcache::raw_read(int, unsigned char*)(0x1107c5030, 0x4000000203, 0xfffffffffffebc0) + 0x8c
0x0000000100581f54 readable_regcache::cooked_read(int, unsigned char*)(0x1107c5030, 0x40ffffeb90, 0xfffffffffffebc0) + 0xec
0x0000000100daba10 register_status readable_regcache::cooked_read<unsigned long, void>(int, unsigned long*)(0x1107c5030, 0x40ffffec50, 0xffffffffffffed10) + 0xd4
0x00000001005826a0 regcache_cooked_read_unsigned(regcache*, int, unsigned long*)(0x1107c5030, 0x40ffffecd0, 0xffffffffffffed10) + 0x70
0x0000000100584e2c regcache_read_pc(regcache*)(0x1107c5030) + 0xa4
0x0000000100387614 handle_signal_stop(execution_control_state*)(0xfffffffffffff3a8) + 0x158
0x00000001003864e4

Secondly in a case where, if we follow the child instead of the parent and we end up changing our “process <pid>” to “thread <tid>” before the process itself raises a trap

and tells the debugger “I am ready for threads”, then when we switch_to_thread in the follow_fork () we end up not finding the “process <pid>” and thus leading to an assertion failure as shown below and rightly so, because we changed threads without the library being initialised. This happens when the follow_fork () is called, and we switch to the child thread there.

(gdb) set detach-on-fork off
(gdb) set follow-fork-mode child
(gdb) r
Starting program:
[New Thread 258]
[New Thread 515]
[Attaching after Thread 515 fork to child process 18809098]
[New inferior 2 (process 18809098)]
thread.c:1337: internal-error: switch_to_thread: Assertion `thr != NULL' failed.

The process stack is as follows:-
0x0000000100036590 internal_error_loc(char const, int, char const*, ...)(0x10192ba70, 0x53900000000, 0x10192b970) + 0x58*
0x0000000100619918 switch_to_thread(thread_info)(0x0) + 0x48*
0x000000010037635c follow_fork()() + 0x4c8
0x0000000100385af8 handle_inferior_event(execution_control_state)(0xfffffffffff3a8) + 0xda8*
0x00000001003809d0 fetch_inferior_event()() + 0x2f8
0x0000000100719a0c inferior_event_handler(inferior_event_type)(0x10207a50) + 0x38
0x000000010039228c infrun_async_inferior_event_handler(void*)(0x0) + 0x30
0x0000000100671d18 check_async_event_handlers()() + 0x94
0x000000010066e32c gdb_do_one_event(int)(0xfffffffffffff840) + 0xb4
0x0000000100001dcc start_event_loop()() + 0x28
0x0000000100001fd4 captured_command_loop()() + 0x58
0x000000010000414c captured_main(void*)(0xfffffffffffa60) + 0x2c
0x0000000100004220 gdb_main(captured_main_args*)(0xfffffffffffa60) + 0x20

So, the changes in the sync_threadlists () with parameter and the for loop justifies the same.

Also, we now do not use iterate_over_threads to count our GDB threads. We instead do it via for (thread_info *tp : all_threads (proc_target, ptid_t (pid))) inline.

Solution Part 2: -

Since we switch_to_no_thread before a wait (), on an event of a thread detection or any other event which makes us use the thread call-backs, we need to be in the right context while we read and write data for threads. That is why we switch our inferior_ptid, current_inferior and program space in pdc_read_data () and pdc_write_data and now pdc_write_data.

So why did we make this change

```
- if (user_current_pid != 0)
-  switch_to_thread (current_inferior ()->process_target (),
-                    ptid_t (user_current_pid));
```

in pdc_read_data and change our user variable which was the process ID to a thread? Wasn't it already doing the job?

Consider an event where the parent process is threaded, and we have a fork (). When we do a pd_update () after the beneath->wait () in thread wait () we call sync_threadlists () as well. Over there we call pthdb_pthread (data->pd_session, &pdtid, cmd);

This now will use the ptid_t (user_current_pid) to switch the thread (). However, our parent process or main thread of it, is threaded i.e is ptid_t (user_current_pid, 0, tid). Hence, we will crash with an assertion failure that thread ptid_t (user_current_pid) has not been found.

In order to avoid the same, we now pass the thread directly. So, on any event after the main process looks like a main thread, there will be no confusion on which thread space or inferior_ptid or program space to switch, especially when a process is multi-threaded.

Solution Part 3: - In AIX we use a lot of variables for different purposes like pd_active, pd_able, arch64, pd_brk_addr and pd_session. These variables are unique per inferior. Hence, we need to keep them in a map <inferior, structure> where structure can hold all these variables per inferior. This is where we use the inbuilt GDB registry for every inferior. This change exists in this patch.

Solution Part 4: -

We figured out that the top target for a new inferior born after the main inferior was incorrect post the process being threaded.

The root cause was that the shared library was not being loaded for new process. The reason being we change our shared library file name in the BFD registry from member name to path(member_name).

Hence the changes in solib-aix takes care of the new pattern so that the shared library can be loaded correctly for every new inferior born as well via pattern matching the ‘(‘character and checking if the member_name exists after that in the new pattern registered in the BFD registry as shown in solib-aix.c changes in this patch.

These 4 solution parts together fixes the bug.