GDB intends to support the following hardware debug features of BookE processors:

4 hardware breakpoints (IAC)
2 hardware watchpoints (read, write and read-write) (DAC)
2 value conditions for the hardware watchpoints (DVC)

For that, we need to extend ptrace so that GDB can query and set these resources. Since we're extending, we're trying to create an interface that's extendable and that covers both BookE and server processors, so that GDB doesn't need to special-case each of them. We added the following 3 new ptrace requests.

1. PTRACE_PPC_GETHWDEBUGINFO

Query for GDB to discover the hardware debug features. The main info to be returned here is the minimum alignment for the hardware watchpoints. BookE processors don't have restrictions here, but server processors have an 8-byte alignment restriction for hardware watchpoints. We'd like to avoid adding special cases to GDB based on what it sees in AUXV.

Since we're at it, we added other useful info that the kernel can return to GDB: this query will return the number of hardware breakpoints, hardware watchpoints and whether it supports a range of addresses and a condition. The query will fill the following structure provided by the requesting process:

```
struct ppc_debug_info {
       unit32_t version;
       unit32_t num_instruction_bps;
       unit32_t num_data_bps;
       unit32_t num_condition_regs;
       unit32_t data_bp_alignment;
       unit32_t sizeof_condition; /* size of the DVC register */
       uint64_t features; /* bitmask of the individual flags */
};
```

features will have bits indicating whether there is support for:

```
#define PPC_DEBUG_FEATURE_INSN_BP_RANGE         0x1
#define PPC_DEBUG_FEATURE_INSN_BP_MASK          0x2
#define PPC_DEBUG_FEATURE_DATA_BP_RANGE         0x4
#define PPC_DEBUG_FEATURE_DATA_BP_MASK          0x8
```

2. PTRACE_SETHWDEBUG

Sets a hardware breakpoint or watchpoint, according to the provided structure:

```
struct ppc_hw_breakpoint {
        uint32_t version;
#define PPC_BREAKPOINT_TRIGGER_EXECUTE  0x1
#define PPC_BREAKPOINT_TRIGGER_READ     0x2
#define PPC_BREAKPOINT_TRIGGER_WRITE    0x4
        uint32_t trigger_type;       /* only some combinations allowed */
#define PPC_BREAKPOINT_MODE_EXACT               0x0
#define PPC_BREAKPOINT_MODE_RANGE_INCLUSIVE     0x1
#define PPC_BREAKPOINT_MODE_RANGE_EXCLUSIVE     0x2
```

```
#define PPC_BREAKPOINT_MODE_MASK                0x3
        uint32_t addr_mode;          /* address match mode */

#define PPC_BREAKPOINT_CONDITION_MODE   0x3
#define PPC_BREAKPOINT_CONDITION_NONE   0x0
#define PPC_BREAKPOINT_CONDITION_AND    0x1
#define PPC_BREAKPOINT_CONDITION_EXACT  0x1     /* different name for the same
thing as above */
#define PPC_BREAKPOINT_CONDITION_OR     0x2
#define PPC_BREAKPOINT_CONDITION_AND_OR 0x3
#define PPC_BREAKPOINT_CONDITION_BE_ALL 0x00ff0000      /* byte enable bits */
#define PPC_BREAKPOINT_CONDITION_BE(n)  (1<<((n)+16))
        uint32_t condition_mode;     /* break/watchpoint condition flags */

        uint64_t addr;
        uint64_t addr2;
        uint64_t condition_value;
};
```

A request specifies one event, not necessarily just one register to be set. For instance, if the request is for a watchpoint with a condition, both the DAC and DVC registers will be set in the same request.

With this GDB can ask for all kinds of hardware breakpoints and watchpoints that the BookE supports. COMEFROM breakpoints available in server processors are not contemplated, but that is out of the scope of this work.

ptrace will return an integer (handle) uniquely identifying the breakpoint or watchpoint just created. This integer will be used in the PTRACE_DELHWDEBUG request to ask for its removal. Return -ENOSPC if the requested breakpoint can't be allocated on the registers.

Some examples of using the structure to:

- set a breakpoint in the first breakpoint register

```
  p.version         = PPC_DEBUG_CURRENT_VERSION;
  p.trigger_type    = PPC_BREAKPOINT_TRIGGER_EXECUTE;
  p.addr_mode       = PPC_BREAKPOINT_MODE_EXACT;
  p.condition_mode  = PPC_BREAKPOINT_CONDITION_NONE;
  p.addr            = (uint64_t) address;
  p.addr2           = 0;
  p.condition_value = 0;
```

- set a watchpoint which triggers on reads in the second watchpoint register

```
  p.version         = PPC_DEBUG_CURRENT_VERSION;
  p.trigger_type    = PPC_BREAKPOINT_TRIGGER_READ;
  p.addr_mode       = PPC_BREAKPOINT_MODE_EXACT;
  p.condition_mode  = PPC_BREAKPOINT_CONDITION_NONE;
  p.addr            = (uint64_t) address;
  p.addr2           = 0;
  p.condition_value = 0;
```

- set a watchpoint which triggers only with a specific value

```
  p.version         = PPC_DEBUG_CURRENT_VERSION;
  p.trigger_type    = PPC_BREAKPOINT_TRIGGER_READ;
  p.addr_mode       = PPC_BREAKPOINT_MODE_EXACT;
  p.condition_mode  = PPC_BREAKPOINT_CONDITION_AND |
PPC_BREAKPOINT_CONDITION_BE_ALL;
  p.addr            = (uint64_t) address;
  p.addr2           = 0;
  p.condition_value = (uint64_t) condition;
```

- set a ranged hardware breakpoint

```
  p.version         = PPC_DEBUG_CURRENT_VERSION;
  p.trigger_type    = PPC_BREAKPOINT_TRIGGER_EXECUTE;
  p.addr_mode       = PPC_BREAKPOINT_MODE_RANGE_INCLUSIVE;
  p.condition_mode  = PPC_BREAKPOINT_CONDITION_NONE;
  p.addr            = (uint64_t) begin_range;
  p.addr2           = (uint64_t) end_range;
  p.condition_value = 0;
```

3. PTRACE_DELHWDEBUG

Takes an integer which identifies an existing breakpoint or watchpoint
(i.e., the value returned from PTRACE_SETHWDEBUG), and deletes the
corresponding breakpoint or watchpoint..