Subject: [PATCH] Priority heap infrastructure enhancements
Posted by Balbir Singh on Wed, 18 Jun 2008 13:48:06 GMT
View Forum Message <> Reply to Message

This patch enhances the priority heap infrastructure to add a delete_max
routine. This patch and routines are helpful as

1. They allow me to delete nodes from the prio_heap (max heap), which is
   currently missing
2. This infrastructure would be useful for the soft limit patches I am working
   on for the memory controller

Some of the common code has been factored into heap_adjust() a.k.a heapify
in data structures terminology.

I am sending out this patch indepedent of the memory controller changes as
they deserve to be reviewed independently.

One limitation of the current heap_insert() routine is that it does not
insert an element which is greater than the root, when the heap slots
are fully used. I'll work on and review that interface and find a suitable
way to address that issue

I've tested them by porting the code to user space (very easy to do) and
I wrote a simple test routine, that ensures that elements are removed
from the heap in descending order.

Comments, Flames? Please do review closely!

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

```
---
 include/linux/prio_heap.h |   10 +++++++-
 lib/prio_heap.c           |   56 ++++++++++++++++++++++++++++++-------------
 2 files changed, 48 insertions(+), 18 deletions(-)

diff --git a/include/linux/prio_heap.h b/include/linux/prio_heap.h
index 0809435..a3578bd 100644
--- a/include/linux/prio_heap.h
+++ b/include/linux/prio_heap.h
@@ -53,6 +53,14 @@ void heap_free(struct ptr_heap *heap);
  */
 extern void *heap_insert(struct ptr_heap *heap, void *p);


-
+/**
+ * heap_delete_max - delete the maximum element from the top of the heap
+ * @heap: The heap to be operated upon
+ *
```

```
+ * The top of the heap is removed, the last element is moved to the
+ * top and the entire heap is adjusted, so that the largest element bubbles
+ * up again
+ */
+extern void *heap_delete_max(struct ptr_heap *heap);

 #endif /* _LINUX_PRIO_HEAP_H */
diff --git a/lib/prio_heap.c b/lib/prio_heap.c
index 471944a..4476bc9 100644
--- a/lib/prio_heap.c
+++ b/lib/prio_heap.c
@@ -23,11 +23,33 @@ void heap_free(struct ptr_heap *heap)
  kfree(heap->ptrs);
 }

+static void heap_adjust(struct ptr_heap *heap)
+{
+ int pos = 0;
+ void **ptrs = heap->ptrs;
+ void *p = ptrs[pos];
+
+ while (1) {
+  int left = 2 * pos + 1;
+  int right = 2 * pos + 2;
+  int largest = pos;
+  if (left < heap->size && heap->gt(ptrs[left], p))
+   largest = left;
+  if (right < heap->size && heap->gt(ptrs[right], ptrs[largest]))
+   largest = right;
+  if (largest == pos)
+   break;
+  /* Push p down the heap one level and bump one up */
+  ptrs[pos] = ptrs[largest];
+  ptrs[largest] = p;
+  pos = largest;
+ }
+}
+
 void *heap_insert(struct ptr_heap *heap, void *p)
 {
  void *res;
  void **ptrs = heap->ptrs;
- int pos;

  if (heap->size < heap->max) {
   /* Heap insertion */
@@ -49,22 +71,22 @@ void *heap_insert(struct ptr_heap *heap, void *p)
  /* Replace the current max and heapify */
```

```
  res = ptrs[0];
  ptrs[0] = p;
- pos = 0;
+ heap_adjust(heap);
+ return res;
+}
+
+void *heap_delete_max(struct ptr_heap *heap)
+{
+ void **ptrs = heap->ptrs;
+ void *res;
+
+ if (heap->size == 0)
+  return NULL;  /* The heap is empty */
+
+ res = ptrs[0];
+ heap->size--;
+ ptrs[0] = ptrs[heap->size]; /* Put a leaf on top */
+ heap_adjust(heap);

- while (1) {
-  int left = 2 * pos + 1;
-  int right = 2 * pos + 2;
-  int largest = pos;
-  if (left < heap->size && heap->gt(ptrs[left], p))
-   largest = left;
-  if (right < heap->size && heap->gt(ptrs[right], ptrs[largest]))
-   largest = right;
-  if (largest == pos)
-   break;
-  /* Push p down the heap one level and bump one up */
-  ptrs[pos] = ptrs[largest];
-  ptrs[largest] = p;
-  pos = largest;
- }
  return res;
 }
--
1.5.5.2
```