1 Design Process & Implementation

Design

In designing the best-fit algorithm for the slob allocator, the overall design was to traverse the allocated memory looking for the best fitting area for the needed memory.

With this end goal in mind, the first step was to understand the slob list structure, and find a way to traverse the list of allocated memory and get the size of each space not occupied. Once we found this value, we will compare it to the size needed to be allocated and see if the current list position is the closest in size to the memory the needs to be allocated.

The biggest design choice that helped us in this project was to try to understadn the memory subsystem and the slob allocator. This would allow for a visualization mentally of how the memory was allocated and arranged, allowing for us to better implment the best-fit algorithm.

Implementation

In implementing the encrypted ramdisk driver, our group first needed to understand the different functions required for the block device driver to be usable for the kernel. In order to encrypt or decrypt the data from the device, the group needed to understand specifically the encryption and decryption of block and bytes.

In implmenting the slob best-fit algorithm, the group needed to understand the way in which slob allocates memory. With this understanding, we created two functions that searched for the best place to put the allocated memory, slob_set_better_fit() and slob_alloc_from_best_fit(). The first function compares the slob page's best fit place to ss if it can find a better place to fit the page. The second function allocates space for the given page at the best-fit position.

Testing

In testing our best-fit implementation, our group created two seperate kernels: one with the first fit slob, and 1 with the best-fit slob. After booting up and running some programs to thrash the memory, we compared the output from /proc/buddyinfo to compare the memory usage and fragmentation.

This is the sample output from out test:
best fit
Node 0, zone DMA 0 1 1 1 2 1 1 0 1 1 3
Node 0, zone Normal 2 3 4 3 1 2 2 7 2 3 206
Node 0, zone HighMem 1 1 1 1 0 1 1 1 1 2 279

after compile main.c
Node 0, zone DMA 0 1 1 1 2 1 1 0 1 1 3
Node 0, zone Normal 1 3 1 3 2 1 0 7 2 3 206
Node 0, zone HighMem 23 23 19 8 1 3 2 1 0 1 275

first fit
Node 0, zone DMA 0 1 1 1 2 1 1 0 1 1 3
Node 0, zone Normal 2 4 0 3 0 1 4 6 2 3 206
Node 0, zone HighMem 2 1 1 1 2 2 1 2 2 1 280

after compile main.c
Node 0, zone DMA 0 1 1 1 2 1 1 0 1 1 3
Node 0, zone Normal 2 9 16 11 6 8 6 10 3 4 204
Node 0, zone HighMem 62 34 14 10 2 6 2 1 2 1 276

Some weird results we noticed while trying to thrash our system memory, we noticed that malloc() didn't have as much effect on the memory fragemnetation as we expected. This is due to the behavior of malloc, so we switched to using sbrk() (which malloc uses) to allocate larger sections of memory. This resulted in a more noticeable memory effect. We also notice that there was not as much memory difference when comparing the mbuddyinfo before and buddyinfo after out memory thrashing program. But there was a difference after compiling a program.

Issues

The biggest issue we faced on this project was trying to understand the behavior of the memory system. This issue caused a lot of confusing when writing out memory thrashing program as to why the values weren't changing as much as expected.

2 Code

```
--- ../linux/mm/slob.c        2012-10-08 07:32:26.228555000 -0700
+++ project4/mm/slob.c        2013-05-27 21:05:23.468839869 -0700
@@ -5,6 +5,8 @@
  *
  * NUMA support by Paul Mundt, 2007.
  *
+ * TODO: best fit stuff by group 13.
+ *
  * How SLOB works:
  *
  * The core of SLOB is a traditional K&R style heap allocator, with
@@ -109,6 +111,35 @@
                struct page page;
        };
 };
+
+/* The current best fitting page's slob_t and how well it fits, as well as probably some
+ * other info we need to tell the allocation function. */
+/* TODO: Add stuff we need to this to tell slob_alloc_from_best_fit(). */
+/* FIXME: rename eveyrthing in here! :v */
+/* FIXME: This stores a ridiculous amount of state info. */
+/* TODO: Rename fit_info to alloc_info */
+typedef struct fit_info {
+        slob_t    *best;          /* The current best fitting block. */
+        slob_t    *previous;      /* The previous slob_t in the list. */
+        slob_t    *aligned_best;  /* The current, but aligned. */
+        struct slob_page *best_page; /* The page that best is on. */
+        size_t    best_leftover; /* The unused space that will result from alloc-ing it. */
+        int       units;         /* The amount of units this allocation takes up. */
+        int       delta;         /* The delta from the thing. */
+        int       avail;         /* The available ... */
```

```
+} fit_info;
+
+/* Sets a fit_info to its default values. */
+static void init_fit_info(fit_info *fit)
+{
+        fit->best = NULL;
+        fit->previous = NULL;
+        fit->best_page = NULL;
+        fit->best_leftover = 0;
+        fit->aligned_best = NULL;
+        fit->units=fit->delta=fit->avail=0;
+}
+
 static inline void struct_slob_page_wrong_size(void)
 { BUILD_BUG_ON(sizeof(struct slob_page) != sizeof(struct page)); }

@@ -264,68 +295,115 @@
         free_pages((unsigned long)b, order);
 }

-/*
- * Allocate a slob block within a given slob_page sp.
- */
-static void *slob_page_alloc(struct slob_page *sp, size_t size, int align)
-{
-        slob_t *prev, *cur, *aligned = NULL;
-        int delta = 0, units = SLOB_UNITS(size);
+/* Compares the current page's best fit with the specified best fit. If this page has a
+ * better fit, out_fit is set to the fit_info coorosponding to this page's best fit. */
+/*FIXME: rename this to slob_best_fit_compare_page*/
+static void slob_set_better_fit(struct slob_page *sp, size_t alloc_size, int align, fit_info *out_f
+{
+        slob_t *prev, *cur, *aligned;
+        int delta = 0, units = SLOB_UNITS(alloc_size);
+        fit_info current_fit;
+
+        /* Initialize the thing. :v */
+        init_fit_info(&current_fit);
+        current_fit.best_page = sp;

+        /* Go thru and try and stuff the thing in. */
         for (prev = NULL, cur = sp->free; ; prev = cur, cur = slob_next(cur)) {
                 slobidx_t avail = slob_units(cur);
-
                 if (align) {
                         aligned = (slob_t *)ALIGN((unsigned long)cur, align);
                         delta = aligned - cur;
                 }
-                if (avail >= units + delta) { /* room enough? */
-                        slob_t *next;
```

```
-                       if (delta) { /* need to fragment head to align? */
-                               next = slob_next(cur);
-                               set_slob(aligned, avail - delta, next);
-                               set_slob(cur, delta, aligned);
-                               prev = cur;
-                               cur = aligned;
-                               avail = slob_units(cur);
-                       }
-
-                       next = slob_next(cur);
-                       if (avail == units) { /* exact fit? unlink. */
-                               if (prev)
-                                       set_slob(prev, slob_units(prev), next);
-                               else
-                                       sp->free = next;
-                       } else { /* fragment */
-                               if (prev)
-                                       set_slob(prev, slob_units(prev), cur + units);
-                               else
-                                       sp->free = cur + units;
-                               set_slob(cur + units, avail - units, next);
+               if (avail >= units + delta) { /* room enough? */
+                       /* It fits! Now is it the best in this page? This should short-circuit I th
+                       if (current_fit.best == NULL || current_fit.best_leftover > (avail - units
+                               current_fit.best_leftover = (avail - units + delta);
+                               current_fit.previous = prev;
+                               current_fit.best = cur;
+
+                               /* HACK: We shouldn't be really storing a lot of this. */
+                               current_fit.aligned_best = aligned;
+                               current_fit.units = units;
+                               current_fit.delta = delta;
+                               current_fit.avail = avail;
                        }
-
-                       sp->units -= units;
-                       if (!sp->units)
-                               clear_slob_page_free(sp);
-                       return cur;
                }
+
                if (slob_last(cur))
-                       return NULL;
+                       break;
+       }
+
+       /* If we found a thing in this page, copy it to out_fit if it's better than it. */
+       if (current_fit.best != NULL) {
+               out_fit->best = current_fit.best;
+               out_fit->previous = current_fit.previous;
+               out_fit->best_page = sp;
```

```
+                out_fit->best_leftover = current_fit.best_leftover;
+                out_fit->aligned_best = current_fit.aligned_best;
+                out_fit->units = current_fit.units;
+                out_fit->delta = current_fit.delta;
+                out_fit->avail = current_fit.avail;
         }
 }

+/* Allocates a block given the specified fit_info struct. */
+static void *slob_alloc_from_best_fit(fit_info *fit, size_t alloc_size, int align)
+{
+        /* Copied mercilessly from slob_page_alloc. */
+        slob_t *next;
+
+        /* There will always be room; we've determined that. */
+        BUG_ON(fit->avail < fit->units + fit->delta);
+
+        if (fit->delta) {
+                next = slob_next(fit->best);
+                set_slob(fit->aligned_best, fit->avail - fit->delta, next);
+                set_slob(fit->best, fit->delta, fit->aligned_best);
+                fit->previous = fit->best;
+                fit->best = fit->aligned_best;
+                fit->avail = slob_units(fit->best);
+        }
+
+        next = slob_next(fit->best);
+        if (fit->avail == fit->units) { /* exact fit? unlink. */
+                if (fit->previous)
+                        set_slob(fit->previous, slob_units(fit->previous), next);
+                else
+                        fit->best_page->free = next;
+        } else {  /* fragment. */
+                if (fit->previous)
+                        set_slob(fit->previous, slob_units(fit->previous), fit->best + fit->units);
+                else
+                        fit->best_page->free = fit->best + fit->units;
+                set_slob(fit->best + fit->units, fit->avail - fit->units, next);
+        }
+
+        fit->best_page->units -= fit->units;
+
+        if (!fit->best_page->units)
+                clear_slob_page_free(fit->best_page);
+        return fit->best;
+}
+
 /*
  * slob_alloc: entry point into the slob allocator.
  */
 static void *slob_alloc(size_t size, gfp_t gfp, int align, int node)
```

```
 {
+        fit_info current_best;
         struct slob_page *sp;
         struct list_head *prev;
         struct list_head *slob_list;
         slob_t *b = NULL;
         unsigned long flags;

+        /* Set up the fit_info. */
+        init_fit_info(&current_best);
+
+        /* early_printk("woop woop"); //no more of this sillyness */
+
         if (size < SLOB_BREAK1)
                 slob_list = &free_slob_small;
         else if (size < SLOB_BREAK2)
@@ -334,7 +412,7 @@
                 slob_list = &free_slob_large;

         spin_lock_irqsave(&slob_lock, flags);
-        /* Iterate through each partially free page, try to find room */
+        /* Iterate through each partially free page, find the one with the best fit. */
         list_for_each_entry(sp, slob_list, list) {
 #ifdef CONFIG_NUMA
                 /*
@@ -348,24 +426,22 @@
                 if (sp->units < SLOB_UNITS(size))
                         continue;

-                /* Attempt to alloc */
+                prev = sp->list.prev; /* FIXME: Necessary? */
+                slob_set_better_fit(sp, size, align, &current_best);
+        }
+        if (current_best.best != NULL) {
+                /* We have our best fit! */
                 prev = sp->list.prev;
-                b = slob_page_alloc(sp, size, align);
-                if (!b)
-                        continue;
-
-                /* Improve fragment distribution and reduce our average
-                 * search time by starting our next search here. (see
-                 * Knuth vol 1, sec 2.5, pg 449) */
-                if (prev != slob_list->prev &&
-                                slob_list->next != prev->next)
-                        list_move_tail(slob_list, prev->next);
-                break;
+                b = slob_alloc_from_best_fit(&current_best, size, align);
+                BUG_ON(!b); /* We've checked this before, so this can't happen. */
         }
+
```

```
         spin_unlock_irqrestore(&slob_lock, flags);

         /* Not enough space: must allocate a new page */
-        if (!b) {
+        if (!current_best.best) {
+                fit_info new_fit;
+                init_fit_info(&new_fit);
                 b = slob_new_pages(gfp & ~__GFP_ZERO, 0, node);
                 if (!b)
                         return NULL;
@@ -378,7 +454,10 @@
                 INIT_LIST_HEAD(&sp->list);
                 set_slob(b, SLOB_UNITS(PAGE_SIZE), b + SLOB_UNITS(PAGE_SIZE));
                 set_slob_page_free(sp, slob_list);
-                b = slob_page_alloc(sp, size, align);
+
+                /* HACK: we got rid of slob_page_alloc, so we have to replicate it. */
+                slob_set_better_fit(sp, size, align, &new_fit);
+                b = slob_alloc_from_best_fit(&new_fit, size, align);
                 BUG_ON(!b);
                 spin_unlock_irqrestore(&slob_lock, flags);
         }
```