Ep2 – SO – Ciro S. Costa 8071488

Arquitetura - simulador

- Responsável pelo contexto
- Armazena:
  - Memoria física e virtual (wrappers p/ manipular os binarios)
  - Indicação dos algoritimos de subst de pag e mem livre
  - Processos
  - Segmentos
  - MMU

Arquitetura - segmentos

- Responsável por manusear memória livre e memória de processos
- Armazena:
  - Duas listas: uma para processos, outra para segmentos livres
  - Algóritimo de gerenciamento
- Métodos principais:
  - seglist_add_process()
    - Onde o algorítimo de gerenciamento age!
  - seglist_free_process()

seglist_add_process()

```
16 mm_segment_t* mm_seglist_add_process(mm_seglist_t* list,
15                                      mm_process_t* process)
14 {
13   mm_segment_t* process_seg = NULL;
12   mm_segment_t* free_seg = list->algorithm(list->holes->next, process->b);
11
10   ASSERT(free_seg != NULL, "virtual memory must not be full");
 9   process_seg = mm_segment_create(free_seg->start, process->b);
 8   process_seg->process = process;
 7
 6   mm_dllist_t* proc_b4 = _search_b4_start(list->processes, process_seg->start);
 5   mm_dllist_insert_after(proc_b4, process_seg);
 4
 3   free_seg->length -= process->b;
 2   free_seg->start += process->b;
 1
229   return process_seg;
  1 }
```

Arquitetura - mmu

- Responsável pela tradução (vaddr)->phys_pos
- Armazena:
  - Qtd page frames; page frames vazios
  - Qtd pages; pages;
  - Tamanhos em bits (offset, page, frame)
  - Algorítimo de substituição de página (`pagesubst_alg`)
    - Inicialização das estruturas de dados
    - Destruição das estruturas de dados
    - Algorítimo *per se*

Arquitetura – mmu - pagesubst_alg

```
 2 struct mm_mmu_t;
 1 typedef void (*mm_mmu_map_cb)(mm_vpage_t*, void* data);
13
 1 typedef struct mm_pagesubst_alg_t {
 2   void (*init)(struct mm_mmu_t* mmu);
 3   void (*destroy)(struct mm_mmu_t* mmu);
 4   mm_vpage_t* (*algorithm)(struct mm_mmu_t* mmu, uint8_t page);
 5 } mm_pagesubst_alg_t;
 6
 7 typedef struct mm_mmu_t {
 8   unsigned pages_count;
 9   unsigned pageframes_count;
10   mm_vpage_t* pages;
11
12   mm_pagesubst_alg_t* pagesubst_alg;
13
14   uint8_t offset_size_bits;
15   uint8_t page_size_bits;
16   uint8_t frame_size_bits;
17   uint8_t offset_mask;
18
19   unsigned char* free_pageframes; // 0 ==> not free. 1 ==> free
20   unsigned free_pageframes_count;
21 } mm_mmu_t;
```

Arquitetura – exemplo fifo_alg

```
5 static mm_pagesubst_alg_t mm_fifo_alg = {
4   .init = mm_fifo_init,
3   .destroy = mm_fifo_destroy,
2   .algorithm = mm_fifo_algorithm,
1 };
```

Mmu – principal função: mmu_access(position)

```
16 unsigned mm_mmu_access(mm_mmu_t* mmu, unsigned position, int* mb)
15 {
14   uint8_t page = position >> mmu->offset_size_bits;
13   uint8_t offset = position & mmu->offset_mask;
12   unsigned ppage;
11
10   if (mmu->pages[page].p) {
 9     mmu->pages[page].r = 1;
 8     return (mmu->pages[page].phys_page << mmu->offset_size_bits) + offset;
 7   }
 6
 5   ppage = mmu->pagesubst_alg->algorithm(mmu, page)->phys_page;
 4   if (mb)
 3     *mb = ppage;
 2
 1   return mm_mmu_access(mmu, position, mb);
111 }
```

Resultados

- Implementação de 1 algorítimo de gerênciamento de espaço livre (:( )
- Implementação de 3 algorítimos de substituição de página (nru, fifo, scp)
- Comparação dos resultados com base na quantidade de *page faults* necessárias para o acesso em 2 cenários:
  - Poucos acessos por processo
  - Muitos acessos por processos

Resultados – Poucos Acessos

Resultados – Vários Acessos