Relatório: Implementação das Funções `kalloc2`, `kfree` e `kstrcpy`

Introdução

A implementação das funções `kalloc2`, `kfree` e `kstrcpy` foi projetada para permitir a alocação, liberação de memória e cópia de strings no heap do kernel, utilizando um esquema de gerenciamento baseado em descritores de páginas. Neste relatório, detalhamos como essas funções foram implementadas, explicando a lógica utilizada para gerenciar a memória de forma eficiente.

---

1. Implementação da Função `kalloc2`

A função `kalloc2` permite a alocação de um número arbitrário de páginas contíguas na memória. Sua implementação segue os seguintes passos:

Passo 1: Inicialização das Variáveis

```
uint8 *ptr;
uint8 *fp_desc = 0; // Ponteiro para o primeiro descritor de página livre
int count = 0; // Contador de páginas livres encontradas
```

- `ptr` armazena o ponteiro para o descritor da página atual.
- `fp_desc` será usado para armazenar o primeiro descritor da região contígua de páginas livres.
- `count` rastreia quantas páginas livres consecutivas foram encontradas.

Passo 2: Busca por um Bloco Contíguo de `pages` Páginas Livres

```
for (int i = 0; i < total_pages; i++) {
    ptr = (uint8*)HEAP_START + i;
    if (free_page(*ptr)) {
        count++;
        if (count == pages) {
            fp_desc = ptr - (pages - 1);
            break;
        }
    } else {
        count = 0; // Reinicia a contagem se encontrar uma página ocupada
    }
}
```

- Percorre todas as páginas disponíveis em busca de um bloco contíguo de `pages` páginas livres.
- Se uma página estiver livre, incrementa `count`.
- Se `count` atingir `pages`, armazena o descritor da primeira página do bloco.

Passo 3: Verificação e Marcar as Páginas como Alocadas

```
if (fp_desc == 0) {
    return 0; // Se não encontrou páginas livres suficientes, retorna NULL
} else {
    for (int i = 0; i < pages; i++) {
        ptr = fp_desc + i;
        set_free_page_flag(ptr, !FREEPG); // Marca a página como alocada
        if (i == pages - 1) {
            set_last_page_flag(ptr, LASTPG); // Marca como última página
        }
    }
}
```

- Se `fp_desc` for `NULL`, a alocação falha.
- Caso contrário, percorre o bloco de páginas alocadas e:
    - Marca cada uma como ocupada.
    - Marca a última página como sendo a última do bloco.

Passo 4: Cálculo e Retorno do Endereço Alocado

```
int desc_pos = (uint64)fp_desc - (uint64)HEAP_START;
return (void*)(desc_pos * PAGE_SIZE + alloc_start);
```

Converte a posição do descritor para o endereço real da memória alocada.

---

2. Implementação da Função `kfree`

função `kfree` é responsável por liberar um bloco previamente alocado por `kalloc` ou `kalloc2`. A implementação original liberava apenas uma única página, mas foi corrigida para liberar corretamente todas as páginas alocadas.

Passo 1: Obter o Ponteiro para o Descritor da Página

```
uint8 *descritor = (uint8 *)HEAP_START + ((uint64)ptr - (uint64)alloc_start) / PAGE_SIZE;
```

Converte o endereço da memória alocada no respectivo descritor no heap.

Passo 2: Verificar se a Página é a Última e Liberar

```
if (descritor < (uint8 *)HEAP_START || descritor >= (uint8 *)HEAP_END) {
    return;
}
```

Passo 3: Verificar se a Página Já Está Livre

```
if (free_page(*descritor)) {
    return;
}
```

Passo 4: Percorrer e Liberar Todas as Páginas do Bloco

```
while (!last_page(*descritor)) {
    set_free_page_flag(descritor, FREEPG); // Marca a página como livre
    descritor += 1; // Avança para o próximo descritor de página
}
```

```
set_free_page_flag(descritor, FREEPG); // Libera a última página
set_last_page_flag(descritor, 0); // Remove a flag de última página
```

---

3. Implementação da Função `kstrcpy`

A função `kstrcpy` copia uma string para um bloco de memória previamente alocado. Segue os seguintes passos:

Passo 1: Inicialização de Ponteiros

```
char *p = (char*)ptr; // Ponteiro para a memória alocada
```

Passo 2: Copiar a String Fonte para o Destino

```
while (*s) {
    *p++ = *s++; // Copia caractere por caractere
}
*p = '\0'; // Adiciona caractere nulo ao final
```

- O loop copia caractere por caractere de `s` para a memória alocada em `ptr`.
- No final, adicionamos `\0` para garantir que a string seja terminada corretamente.

---

4. Conclusão

A implementação das funções `kalloc2`, `kfree` e `kstrcpy` foi projetada para fornecer um gerenciamento de memória eficiente no heap do kernel. A estratégia de uso de descritores permite a alocação de blocos contíguos de páginas e facilita a liberação correta da memória alocada.

A função `kfree` garante que um bloco inteiro de memória seja liberado corretamente, permitindo a reutilização eficiente dos recursos.

A adição de `\0` em `kstrcpy` assegura que a string copiada seja corretamente terminada, evitando comportamentos indefinidos. Essa abordagem permite que o kernel manipule a memória de maneira eficiente e escalável, garantindo que os recursos sejam utilizados de maneira ótima.